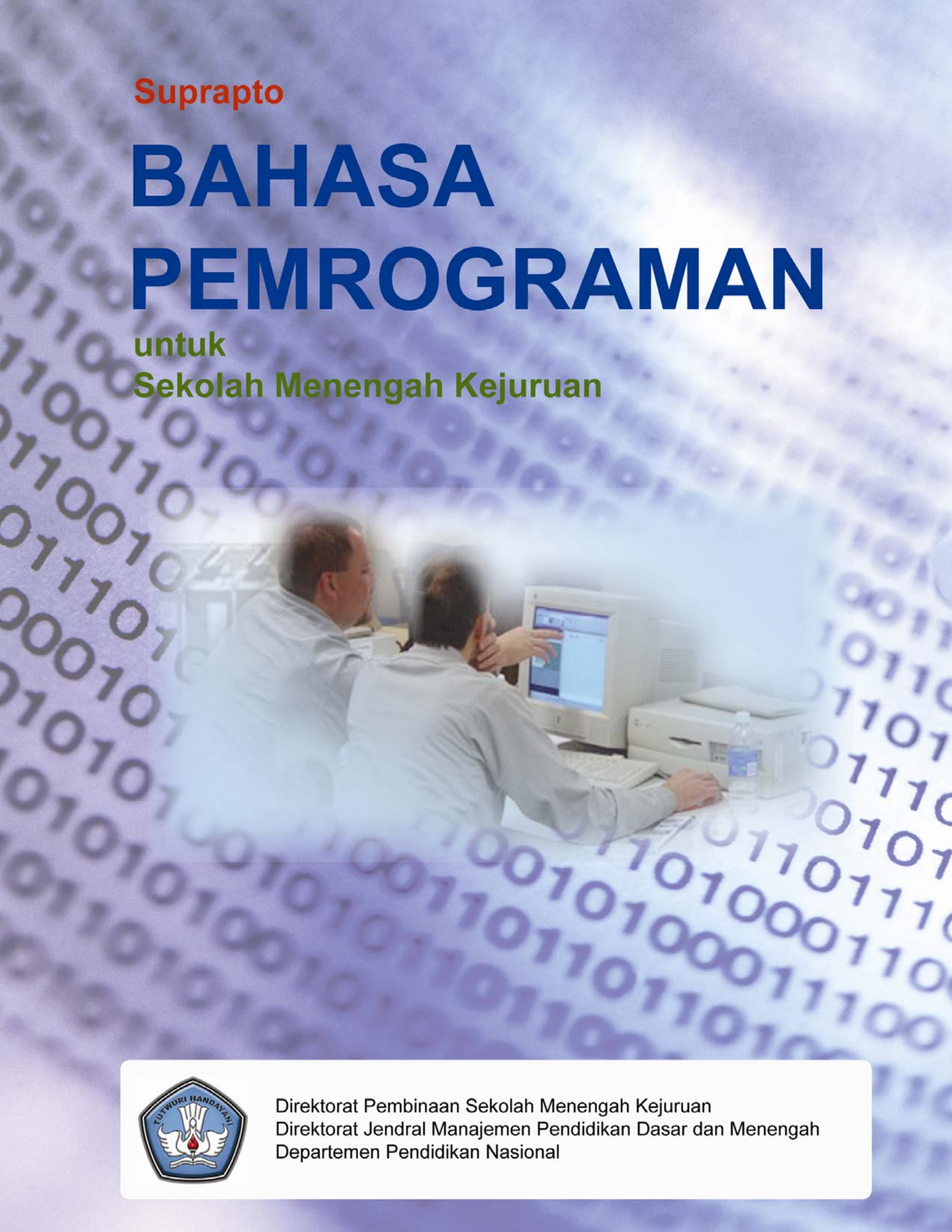
Suprapto

BAHASA PEMROGRAMAN

untuk

Sekolah Menengah Kejuruan



Direktorat Pembinaan Sekolah Menengah Kejuruan
Direktorat Jendral Manajemen Pendidikan Dasar dan Menengah
Departemen Pendidikan Nasional

BAHASA PEMROGRAMAN

Untuk SMK

Penulis : Suprapto
Kadarisman Tejo Yuwono
Totok Sukardiyono
Adi Dewanto

Editor : Ratu Amilia Avianti

Perancang Kulit : Tim

Ukuran Buku : 18,2 x 15,7 cm

Direktorat Pembinaan Sekolah Menengah Kejuruan
Direktorat Jenderal Manajemen Pendidikan Dasar dan Menengah
Departemen Pendidikan Nasional
Tahun 2008

KATA PENGANTAR

Pertama kali kami memanjatkan rasa syukur ke hadirat Allah Subhanahu Wata'la bahwasanya penyusunan buku dengan judul "Bahasa Pemrograman " ini dapat diselesaikan. Kerja keras yang telah dilakukan dalam penulisan ini telah membuahkan hasil dengan baik. Buku ini sangat berarti bagi para siswa Sekolah Menengah Kejuruan (SMK) terutama mereka yang mempelajari bidang teknologi informatika dan komunikasi atau bidang lain yang sejenis.

Selain itu, dengan ditulisnya buku ini, akan menambah perbendaharaan pustaka yang dapat dijadikan pegangan bagi para guru. Kami menyadari bahwa ketersediaan buku yang memadai bagi para siswa dan guru sekarang ini dirasakan masih kurang. Sejalan dengan kemajuan jaman dan teknologi yang ada, maka sudah sepantasnya perlu ada upaya untuk mencerdaskan para siswa dengan kampanye penulisan buku.

Buku yang ditulis ini diharapkan dapat menjembatani kebutuhan siswa dan guru terhadap materi-materi pelajaran yang diajarkan di sekolah. Dengan demikian keluhan sulitnya mencari buku bermutu yang ditulis dalam bahasa Indonesia sudah tidak akan didengar lagi. Sebagaimana yang ditulis dalam pengantar Buku Standar Kompetensi Nasional Bidang teknologi Informasi dan Telekomunikasi bahwa demikian luasnya bidang TIK, prioritas utama dalam penyusunan standar kompetensi ditujukan untuk bidang-bidang pekerjaan yang berhubungan dengan Teknologi Informasi dan Telekomunikasi. Namun buku pegangan "bahasa Pemrograman" ini akan memuat pengetahuan mendasar tentang bahasa Pemrograman khususnya bahasa Prosedural dan OOP. Selanjutnya bagi yang berkepentingan dengan buku ini dapat mengimplementasikannya dalam pemberdayaan proses belajar mengajar yang berlangsung di SMK.

Dalam kesempatan ini ucapan terima kasih yang sebesar-besarnya disampaikan kepada para anggota Tim Penulis, para konstributor materi yang telah bersama kami menyusun dan menyempurnakan isi buku ini. Kepada Direktur Pembinaan Sekolah Menengah Kejuruan (PSMK), kami sampaikan penghargaan dan terima kasih atas dukungan dan bantuannya sehingga penulisan buku ini dapat dilaksanakan dengan baik dan berhasil memenuhi kriteria.

Akhirnya kami persembahkan buku ini kepada para pelaksana di jajaran SMK. Apapun hasil yang telah dicapai merupakan perwujudan kerja keras yang hasilnya bersama-sama dapat kita lihat setelah implementasi dan siswa mencapai keberhasilan studi. Semoga bermanfaat bagi kita sekalian.

Tim Penulis

DAFTAR ISI

BAB 1
DASAR-DASAR PEMROGRAMAN KOMPUTER

1.1. Pendahuluan

Setiap orang yang bekerja biasanya membutuhkan alat bantu untuk menyelesaikan pekerjaannya supaya menjadi lebih mudah. Seorang tukang kayu misalnya membutuhkan palu, gergaji dan pengukur. Ahli mesin membutuhkan kunci pas dan obeng. Seorang teknisi elektronika membutuhkan multimeter, oscilloscope dan solder untuk menyelesaikan pekerjaannya.

Beberapa peralatan bantu tersebut dapat dikategorikan sesuai dengan jenis pekerjaannya, misalnya seorang ahli bedah, maka orang tersebut harus mempunyai peralatan yang didesain secara khusus untuk melakukan operasi. Peralatan tersebut tentunya tidak biasa digunakan oleh orang lain selain ahli bedah.

Ada beberapa peralatan yang digunakan oleh beberapa profesi, misalnya: obeng digunakan oleh ahli mesin, tukang kayu, tukang listrik dan lain sebagainya. Selain obeng, komputer juga merupakan sebuah peralatan yang digunakan oleh banyak profesi, sehingga hal tersebut sangat sulit dikategorikan pada bidang apa.

Selain seperti dijelaskan diatas komputer juga mencakup banyak pekerjaan yang berbeda atau boleh jadi dapat dikatakan menjadi peralatan yang paling serbaguna yang pernah dibuat.

Pemanfaatan komputer oleh seorang akuntan, digunakan untuk

menganalisis keuntungan, untuk membuat laporan keuangan, tetapi pada sebuah pabrik komputer digunakan sebagai kendali mesin-mesin produksi, sedangkan pada seorang mekanik digunakan untuk menganalisis berbagai sistem pada mesin dan permasalahan lainnya.

Mengapa komputer menjadi peralatan yang sangat serbaguna?. Jawabanya sangat sederhana, komputer dapat mengerjakan tugas-tugas yang bervariasi karena komputer dapat diprogram. Komputer merupakan sebuah mesin yang khusus hanya mengikuti instruksi yang diberikan padanya.Karena komputer bersifat *programmable*, sehingga komputer tidak hanya milik satu profesi saja. Komputer dirancang untuk mengerjakan pekerjaan yang sesuai program-program yang diberikannya padanya.

Pekerjaan sebagai programmer merupakan pekerjaan yang sangat penting karena merekalah yang membuat perangkat lunak yang digunakan untuk menginstruksikan komputer sebagai peralatan yang sesuai dengan yang diinginkan. Tanpa programmer, pengguna komputer tidak mempunyai perangkat lunak, dan tanpa perangkat lunak komputer tidak akan bisa mengerjakan apapun.

Dalam pemrograman komputer ada dua kombinasi yang tidak terpisahkan yaitu seni dan ilmu pengetahuan. Dikatakan dalam seni karena setiap aspek dalam program harus dirancang dengan hati-hati. Hal-hal yang perlu diperhatikan dalam mendesain sebuah komputer adalah sebagai berikut: Aliran instruksi secara logic, Procedure matematik, Tampilan yang akan muncul pada layar monitor, Informasi yang ditampilkan oleh user, Program harus "*user friendly*", serta Petunjuk penggunaan maupun bentuk dokumen tertulis lainnya.

Pemprograman berkaitan dengan aspek ilmu pengetahuan berkaitan dengan ilmu teknik karena jarang sekali program dapat berjalan baik pada saat program pertama kali ditulis. Biasanya perlu banyak dilakukan percobaan, pembetulan dari kesalahan maupun dirancang ulang sesuai dengan kebutuhan. Dengan adanya hal tersebut diatas maka dibutuhan seorang programmer yang memahami dua kemampuan yaitu seni dan ilmu pengetahuan.

Seorang programmer harus menguasai bahasa yang dimiliki oleh komputer seperti bahasa C++, Java atau bahasa pemrograman lainnya. Bahasa tersebut merupakan cara komputer supaya bisa memahami apa yang diperintahkan karena komputer tidak paham bahasa Indonesia atau bahasa manusia manusia, sehingga programmerlah yang harus menyesuaikan dengan komputer.

Bahasa komputer mempunyai aturan-aturan yang harus diikuti. Dalam menulis program komputer yang meliputi seni dan ilmu pengetahuan tersebut seperti halnya dalam merancang sebuah mobil, dimana mobil tersebut harus mempunyai tingkat fungsional yang tinggi, efficient, bertenaga maksimal, mudah digunakan, dan amat menyenangkan jika dilihat.

1.2. Perangkat Keras Komputer

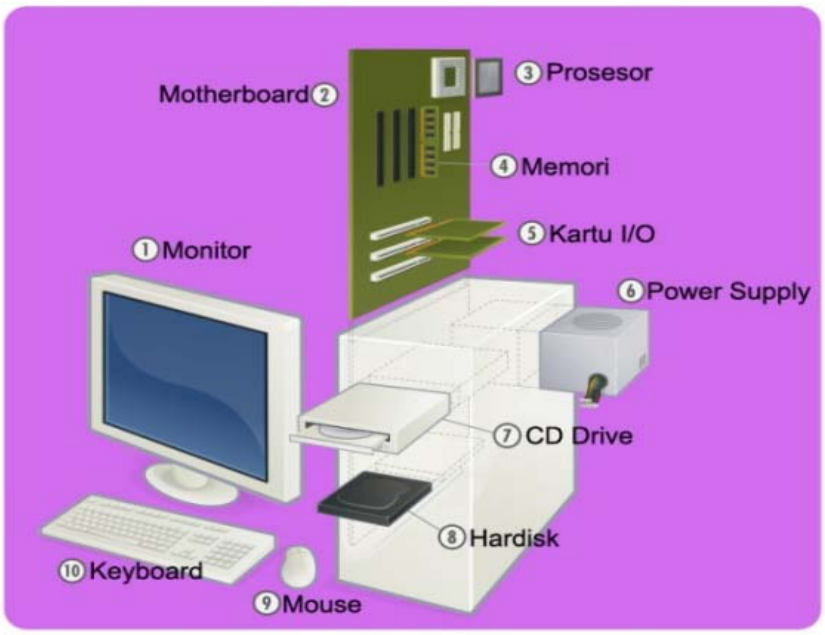
Komputer merupakan sebuah mesin yang bekerja untuk memproses, menyimpan, serta mendapatkan data. Data-data tersebut berupa angka, karakter, titik warna, gelombang bunyi atau suatu kondisi sebuah sistem, seperti pendingin atau CD player. Semua data disimpan dalam bentuk angka-angka.

Angka tersebut dalam bentuk bilangan biner yang diwakili oleh angka 1 dan 0 yang sering disebut bit. Supaya mudah dalam mengingatnya, maka komputer mengelompokan data biner tersebut menjadi *nible, byte* dan *word*. Dengan mengelompokan tersebut, selain mudah diingat, juga akan memudahkan pengguna dalam menuliskan sebuah program berupa kode yang dimengerti oleh mesin, merancang sebuah struktur data dan algoritma yang komplek.

Komputer memanipulasi data dengan melakukan operasi, baik penjumlahan, pengurangan, perkalian maupun pembagian. Hasil manipulasi angka tersebut ditunjukan dalam bentuk gambar pada monitor serta deretan angka-angka pada memori video, dimana masing-masing angka atau sejumlah angka akan mewakili suatu pixel warna.

Untuk memainkan sebuah MP3, komputer akan membaca deretan angka-angka dari disk dan memindahkannya kedalam memori. Selanjutnya komputer menggerakkan angka-angka tersebut untuk dikonversi menjadi data audio yang dimampatkan. Dan yang terakhir adalah data audio yang dimampatkan tersebut akan dikirim ke chip audio.

Semua hal yang dikerjakan oleh komputer, mulai dari web browsing sampai mencetak, melibatkan perpindahan dan pemprosesan angka. Secara elektronis komputer tak lain hanya suatu sistem atau benda yang hanya dirancang untuk menyimpan, dan memindahkan, menggerakkan, serta merubah angka-angka.

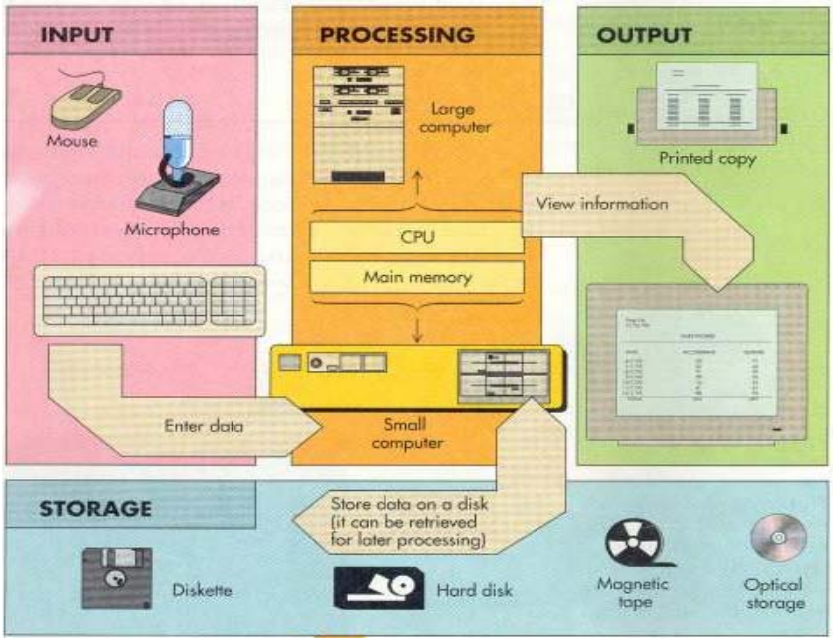


Gambar 1.1. Bagian-bagian komputer

Komputer terdiri dari beberapa komponen, yang secara garis besar dibagi menjadi dua yaitu: berupa perangkat keras dan perangkat lunak. Komponen utama pada perangkat keras, terletak pada pusat komputer adalah *prosesor*, prosesor ini berfungsi mengeksekusi program komputer.

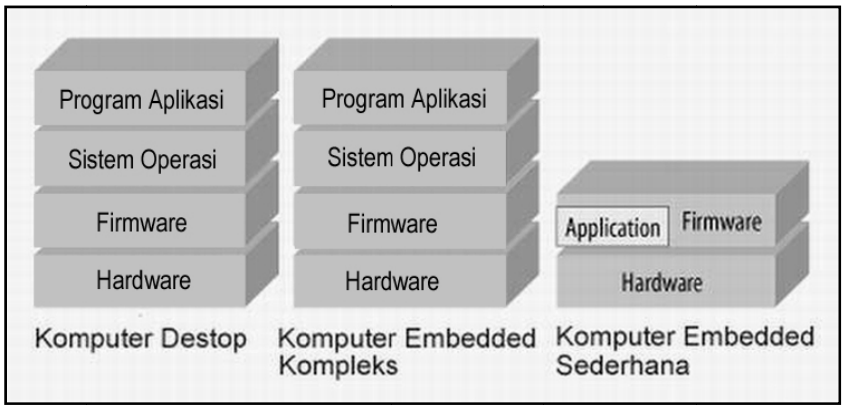
Selain prosesor, komputer juga mempunyai memori. Dalam sebuah komputer biasanya terdapat beberapa memori yang berbeda-beda. Memori ini digunakan untuk menyimpan program ketika prosesor sedang bekerja. Komputer juga mempunyai piranti untuk menyimpanan dan mempertukarkan data dengan dunia luar atau yang sering disebut I/O.

Piranti I/O akan mempertukarkan data, seperti: masukan teks melalui keyboard serta mendisplaykannya pada layar monitor. I/O juga digunakan untuk memindahkan data maupun program ke atau dari suatu disk drive, modem, printer, mouse dan lain-lain.



Gambar. 1.2. Perangkat Keras Sebuah Sistem Komputer

Perangkat lunak mengendalikan fungsi dan operasi sebuah komputer. Ada beberapa lapisan (*layer*) perangkat lunak yang digunakan di dalam komputer. Secara umum lapisan akan saling berhubungan dengan layer diatas atau dibawahnya.



Gambar. 1.3. Lapisan Perangkat Lunak

Pada perangkat lunak tingkatan yang paling rendah, perangkat lunak dijalankan oleh prosesor ketika komputer pertama kali dihidupkan. Perangkat lunak ini melakukan inisialisasi perangkat keras sistem tersebut untuk mengetahui kondisi dan mengatur komputer pada operasi

yang benar. Perangkat lunak ini bersifat permanen dan disimpan dalam memori komputer. Perangkat lunak inilah yang dikenal sebagai *firmware*

Firmware digunakan untuk meletakan program *bootloader*. *Bootloader* adalah sebuah program khusus dan dijalankan oleh prosesor ketika membaca sistem operasi dari disk atau memori nonvolatile yang kemudian menempatkannya di dalam memori. Bootloader biasanya dimiliki komputer desktop dan workstation.

Lapisan perangkat lunak diatas firmware, adalah sistem operasi. Perangkat lunak ini berfungsi mengendalikan operasi komputer, mengorganisir penggunaan memori dan mengendalikan peralatan seperti keyboard, mouse, monitor, disk drive, dan sebagainya. Sistem operasi juga memberikan fsilitas kepada user untuk melakukan antarmuka dengan piranti lain, menjalankan program aplikasi dan mengakses file memori luar seperti Compact Disk (CD).

Sistem operasi, secara umum menyediakan satu set tool untuk program aplikasi, melakukan suatu mekanisme pengaksesan monitor, disk drive, dan seterusnya.

Kenyataan dilapangan sebuah komputer tidak semua menggunakan sistem operasi. Sering juga komputer bersifat sangat sederhana dan langsung menjalankan tugasnya. Pada permasalahan tertentu, seperti router jaringan, perangkat lunaknya terintegrasi dan sangat sederhana proses pengembangan.

Perangkat lunak pada lapisan paling tinggi adalah perangkat lunak aplikasi yang merupakan program yang langsung berhubungan dengan kemampuan sebuah komputer. Kemampuan sebuah komputer sangat tergantung pada aplikasi perangkat lunak sistem.

1.3. Arsitektur Komputer

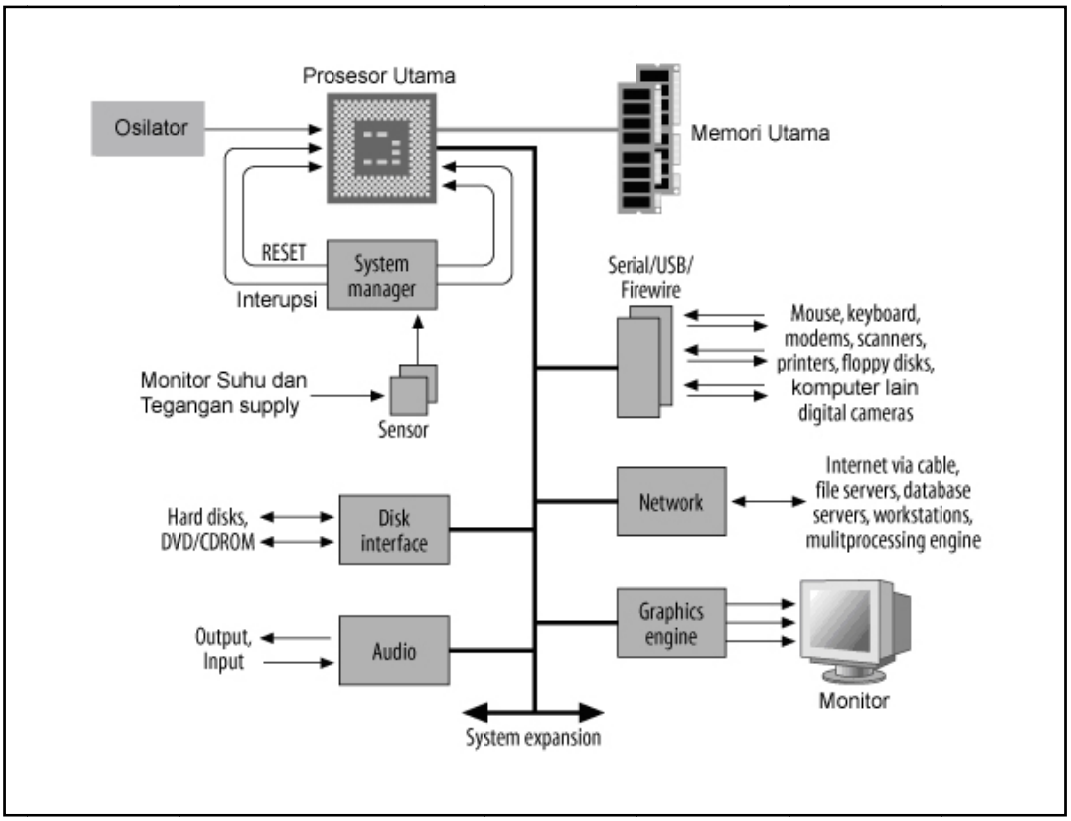
Sebuah prosesor atau yang sering disebut dengan central Procesing Unit (CPU) pada sebuah komputer tidak bisa bekerja sendiri dalam melakukan kerja sebagai pemroses. CPU memerlukan komponen-komponen pendukung seperti memori untuk menyimpan data dan program, serta piranti I/O (Input/Output) yang digunakan untuk memindahkan data antara komputer dan dunia luar.

Selain itu juga komputer memerlukan clock (detak) sebagai penggerak prosesor dalam memproses data.

Mikroprosesor adalah suatu pengolah yang dibentuk oleh sebuah chip tunggal atau sering disebut *integrated circuit*. Mikroprosesor ini sering ditemukan pada sebuah superkomputer, komputer PC, atau sekarang ini hampir semua pengolah data modern adalah mikroprosesor.

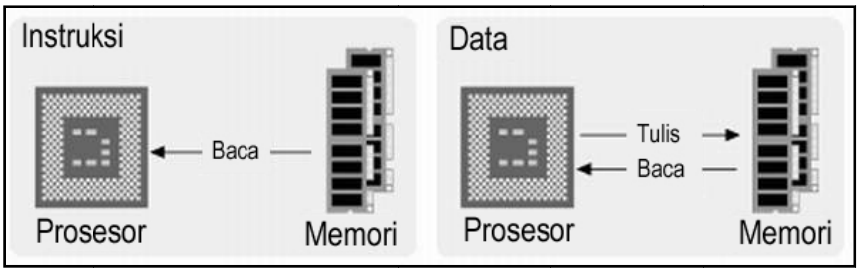
Mikroprosesor yang paling banyak digunakan saat ini adalah: seri Intel Pentium, Freescale/IBM PowerPC, MIPS, ARM, and the Sun SPARC, dan lain-lain.

Blok diagram sebuah sistem komputer dapat ditunjukkan pada Gambar dibawah.



Gambar. 1.4. Diagram blok sistem komputer

Pada gambar diagram blok sebuah sistem komputer diatas, memori berisi instruksi dan bersama-sama prosesor melaksanakan dan menggerakkan data. Memori suatu sistem komputer tidak pernah kosong dan selalu terisi apakah berupa instruksi ataupun berupa data. Instruksi diambil dan dibaca dari memori menuju prosesor, sedangkan data dibaca dari dan ditulis oleh prosesor ke memori, hal ini ditunjukan pada gambar dibawah:



Gambar.1.5. Aliran Data pada sebuah Komputer

Bentuk aliran data arsitektur komputer tersebut diatas dikenal dengan arsitektur Von Neumann, dimana nama tersebut diambil dari penemunya yaitu: Yohanes Von Neumann. Hampir semua komputer modern sekarang ini mengikuti format arsitektur ini.

Pada komputer arsitektur Von Neumann langkah-langkahnya diatur oleh kendali suatu program. Dengan kata lain, komputer mengikuti suatu langkah-langkah program yang memerintahkan operasinya.

1.3.1. Central Processing Unit (CPU)

CPU atau yang sering disebut prosesor merupakan bagian terpenting pada sebuah komputer. Dalam sistem komputer, prosesor menjadi bagian yang menjalankan komputasi dari komputer tersebut.

Prosesor adalah suatu piranti elektronik yang mampu melakukan manipulasi data dengan cara yang disesuaikan oleh suatu urutan instruksi. Instruksi tersebut berfungsi sebagai opcode atau kode mesin. Urutan instruksi ini dapat diubah dan disesuaikan dengan aplikasi, hal ini dikarenakan sifat komputer yang programmable. Urutan instruksi adalah sesuatu yang mendasari sebuah program.

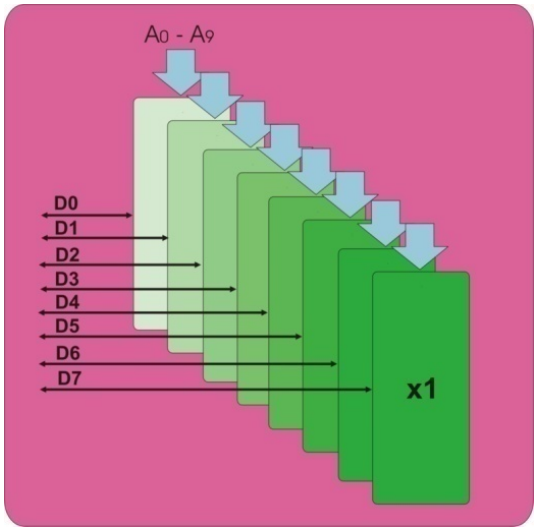
Instruksi pada sebuah komputer adalah berupa angka-angka. Angka yang berbeda, ketika dibaca dan yang dieksekusi oleh suatu prosesor, akan menyebabkan sesuatu hal yang berbeda pula. Instruksi pada sebuah mesin menyesuaikan dengan *machine code* yang sesuai, ini artinya bahwa setiap prosesor mempunyai instruksi masing-masing sesuai industri yang memproduksinya. Suatu instruksi yang berbeda mempunyai arti bahwa mesin yang diprogram juga berbeda.

1.3.2. Memori

Memori digunakan untuk menyimpan perangkat lunak yang berupa data maupun obcode sebuah prosesor. Memori dapat dikategorikan menjadi memori yang dapat menyimpan data secara permanen walaupun listrik yang mengalir pada memori tersebut diputus dan memori ini sering disebut *Nonvolatail memory* (tidak mudah berubah isinya), dan memori yang bersifat sementara atau data yang disimpan dalam memori tersebut akan hilang jika listrik yang mengalir diputus, dan jenis memori ini sering disebut dengan *Volatail memory*.

Kedua jenis memori tersebut mempunyai kelebihan serta kelemahan masing-masing, sehingga penggunaannyapun disesuaikan dengan kebutuhan masing-masing.

Memori diimplementasikan dalam bentuk chip yang didalamnya berisi ribuan komponen elektronika. Memori ini dapat digambarkan dalam blok diagram seperti gambar dibawah:



Gambar. 1.6. Memori 8 bit data x 10 bit alamat

Gambar tersebut diatas adalah memori yang terdiri dari bus alamat yang dikodekan dengan $A_0 – A_9$. Bus alamat ini bersifat satu arah yaitu sebagai masukan sja. Selain bus alamat terdapat juga bus data sebanyak 8 bit yang bersifat dua arah sebagai masukan maupun keluaran.

Bus yang dimiliki memori selain bus alamat dan data adalah bus kendali. Bus alamat digunakan untuk memilih data yang disimpan pada lokasi memori, dimana banyaknya lokasi pada tiap blok memori adalah 2^n, jika n = 10 maka jumlah lokasi memori yang mungkin adalah 2^{10} = 1024 bit.

Setiap bit data tersimpan dalam memori dalam bentuk biner 0 atau 1. Jika banyaknya lokasi dikalikan dengan jumlah banyaknya bit dalam tiap lokasi, untuk 10 alamat bit maka akan memperoleh kapasistas memori 1024x8 bit.

1.3.2.1. Random Access Memory (RAM)

RAM adalah memori yang dapat diakses secara Acak. Nama ini sebenarnya sebenarnya kurang tepat, karena kebanyakan memori komputer dapat diproses secara akses acak. RAM menjadi memori aktif didalam sistem komputer, dimana CPU dapat dengan mudah menulis data pada memori jenis ini.

Data pada memori RAM mudah hilang, jika aliran listriknya terputus. Informasi apapun yang disimpan didalam RAM harus ditahan atau harus ditulis kembali setiap beberapa waktu tertentu, walaupun ada RAM Nonvolatile khusus yang mengintegrasikan battery backup pada sistem.

RAM dapat dikategorikan menjadi dua yaitu: RAM statis yang sering dikenal sebagai SRAM dan RAM dinamik yang sering dikenal sebagai DRAM. RAM statis menggunakan gerbang logika yang berpasangan untuk menjaga masing-masing bit data. SRAM

mempunyai keuntungan mempunyai proses yang lebih cepat dan mudah digunakan, memerlukan rangkaian pendukung eksternal yang sedikit, dan dengan konsumsi daya yang relatif rendah. Kelemahan SRAM adalah pada kapasitas memori SRAM sangat kecil dibandingkan DRAM, serta teknologinya jauh lebih mahal.

Dengan kapasitas SRAM yang rendah, memerlukan lebih banyak chip untuk digunakan pada penerapan yang sama. Pada rancangan PC modern penggunaan SRAM hanya pada mesin yang membutuhkan kinerja yang sangat cepat seperti pada Cache Memory Prosesor.

DRAM menggunakan kapasitor sebagai komponen utamanya untuk mempertahankan data tiap-tiap bit. Deretan kapasitor akan menjaga data hanya dalam waktu yang sangat pendek, selanjutnya data tersebut akan hilang sesuai dengan sifat kapasitor.

DRAM memerlukan penyegaran berlanjut, tiap-tiap penyegaran tersebut sedikitnya seperseribu detik. Penyegaran ini dilakukan secara terus-menerus. Penyegaran tersebut memerlukan komponen pendukung tambahan yaitu komponen eksternal. DRAM mempunyai kelebihan yaitu dapat mempuyai kapasitas yang tinggi.

1.3.2.2. Read Only Memory (ROM)

Read Only Memory (ROM) adalah memori yang hanya dapat dibaca saja. ROM bersifat Nonvolatile memory, karena memori ini dapat mempertahankan muatannya. ROM biasanya lebih lambat dibanding RAM, dan sangat lebih lambat dibanding RAM statis.

Fungsi ROM yang utama di dalam suatu sistem adalah menjaga kode atau data yang diperlukan, seperti untuk inisialisasi pada saat start. Perangkat lunak tersebut biasanya dikenal sebagai firmware. Firmware berisi perangkat lunak inisialisasi komputer dan penempatan sarana Input/Output (I/O) ke dalam suatu status yang dapat dikenal.

Standard ROM dibuat oleh suatu deretan diode yang sangat banyak. Ketika datanya kosong, semua data ROM dalam kondisi berlogika 1, ini artinya data ROM pada 8 bit adalah 0FFH. Ketika mengisikan data kedalam ROM sering dikenal dengan burning ROM. Pengisian data ini dilakukan dengan mengubah data yang berada dalam ROM yang semuanya berupa data 1 dengan menciptakan data nol pada penempatan bit. Suatu piranti yang digunakan untuk memindahkan data disebut ROM downloader atau sering juga disebut dengan ROM Programmer.

ROM biasanya disebut juga dengan *One Time Programmable* (OTP), karena sesuai namanya ROM yang hanya bisa dibaca saja. Pada Industri komputer ROM biasanya digunakan sebagai *firmware*. Mask Programmable ROM juga merupakan ROM yang hanya sekali diprogram, tetapi tidak sama dengan OTP, karena Mask Programmable ROM diprogram oleh industri pembuatnya sebelum dipasarkan. Sedangkan OTP biasanya diprogram oleh pengguna sesuai dengan keinginan dan setelah

diprogram tidak bisa diprogram ulang.

1.3.2.3. Erasable Read Only Memory (EPROM)

OTP ROM mempunyai sifat hanya diprogram sekali, dan hal ini menyebabkan pemborosan karena jika data telah dimasukan dalam OTP ROM dan terdapat kesalahan tidak bisa dihapus lagi, dan harus ganti dengan yang baru serta yang lama harus dibuang. Hal ini jelas merupakan kelemahan dari ROM tersebut. Dengan demikian perlu adanya ROM yang baru yang bisa diprogram ulang, jika data yang dimasukan dalam ROM terdapat kesalahan.

ROM yang muncul setelah OTP ROM adalah jenis ROM yang Erasable Read Only Memory yang sering disebut dengan EPROM. ROM ini sangat cocok untuk pengembangan sistem. Cara menghapus data yang berada dalam EPROM adalah dengan memberikan cahaya ultraungu melalui jendela kecil yang terdapat pada chip dengan waktu tertentu. Setelah berapa saat data yang berada dalam chip tersebut akan terhapus dan menjadi data berlogika tinggi semua.

Setelah data yang berada pada EPROM terhapus memungkinkan untuk dapat memprogram kembali.

EPROM banyak digunakan dalam pengembangan sistem karena kepraktisanya serta lebih hemat dibandingkan dengan OTP ROM. Biasanya OTP ROM dan EPROM mempunyai kapasitas yang sangat kecil dibandingkan dengan kapasitas memori yang ada sekarang ini. Kapasitas ini biasanya hanya beberapa kilobyte saja, tetapi walaupun begitu memori ini tetap digunakan pada berbagai peralatan tertentu, hal ini dikarenakan berbagai pertimbangan yang tidak mungkin digantikan oleh memori jenis lain.

Kelemahan jenis memori EPROM ini adalah bahwa chip harus dipindahkan dari rangkaian ketika akan melakukan penghapusan data yang ada didalamnya. Selain itu dalam melakukan hapus, memerlukan waktu beberapa menit. Dalam melakukan penulisan program ke dalam chip juga harus memindahkannya ke memori EPROM programer. Hal tersebut jelas sangat menyulitkan dan memakan waktu, sehingga dengan munculnya teknologi memori sekarang ini, jenis EPROM mulai ditinggalkan dan jarang digunakan.

1.3.2.4. Electrically Erasable ROM (EEROM)

EEROM adalah memori yang dapat dihapus/tulis secara elektris, atau Electrically Erasable Programmable Read Only Memory dan sering dikenal dengan sebutan sebagai EEPROM. EEPROM dapat dihapus dan ditulis ulang dengan tidak perlu dilepas dari rangkaiannya, sehingga menjadi sangat praktis dan effisien.

Pemrograman dan penghapusan data pada EEPROM dapat dilakukan dengan cepat dibandingkan dengan memori ROM sebelumnya. Hal inilah yang membuat alasan mengapa EEPROM berkembang pesat.

Kapasitas memori EEPROM biasanya hanya beberapa kilobyte saja. Memori ini tidak cocok sebagai firmware, biasanya memori ini digunakan untuk menyimpan informasi atau data sistem, sehingga ketika listrik terputus, datanya tidak akan hilang.

1.3.2.5. Memori Flash

Flash merupakan teknologi ROM yang terbaru, dan sekarang ini paling banyak digunakan. Flash Memori merupakan EEROM yang dapat ditulis dan hapus ulang berkali-kali. Flash memori mempunyai kapasitas yang sangat besar diatas ROM standard.

Flash Chip ini sering juga dikenal sebagai flash ROM atau Flash RAM. Karena ROM tersebut tidak seperti standard ROM atau standard RAM sebelumnya, maka hal tersebut dapat juga disebut dengan flash saja untuk menghindari kekacauan penyebutan antara RAM dan ROM sebelumnya.

Flash memori secara umum diatur menjadi beberapa sektor memori, dan hal ini akan menguntungkan, karena pada tiap-tiap sektor memungkinkan dihapus dan ditulis ulang tanpa mempengaruhi isi sektor lain.

Hal yang khusus pada flash memori adalah sebelum ditulisi, sektor akan dihapus terlebih dahulu, sehingga hal tersebut tidak akan terjadi overwrite seperti pada RAM.

Ada beberapa perbedaan-perbedaan teknologi flash terutama pada penghapusan dan penulisan yang dibutuhkan oleh piranti tersebut.

1.3.3. Input/Output

Dalam sebuah komputer, prosesor dan memori berhubungan dengan berbagai piranti luar yang dihubungkannya. Karena berbagai piranti tersebut merupakan suatu yang ditambahkan dengan prosesor, maka piranti tersebut sering dikenal sebagai piranti peripheral. Piranti tersebut melakukan komunikasi dengan prosesor yang diatur melalui protocol tertentu. Selanjutnya, berbagai piranti tersebut memerlukan pengaturan yang dalam hal ini dilakukan oleh sistem operasi.

Sesuai dengan arah penyalurannya, dalam komputer dikenal sebagai piranti Input (masukan), piranti output (keluaran), dan piranti input output (masukan keluaran). Diantara berbagai jenis piranti tersebut terdapat piranti perekaman informasi berbentuk disk atau disket. Piranti tersebut sering dikenal dengan pheriperal. Biasanya peripheral dibuat oleh berbagai perusahaan untuk berbagai kegunaan.

Pada piranti tertentu, bagian sistem pengelolaan piranti itu dibuat juga oleh perusahaan pembuat piranti bersangkutan. Tentunya pembuatan bagian sistem operasi pengelolaan piranti itu telah disesuaikan dengan sistem operasi yang pada umumnya ada di dalam sistem komputer.

Bagian sistem operasi untuk pengelolaan piranti peripheral itu secara khusus, diatur oleh pengendali piranti secara umum,

diatur oleh piranti lunak pengatur piranti (driver).

Arah komunikasi masukan keluaran bersangkutan dengan alamat. Mereka menunjukan dari alamat mana ke alamat mana, masukkan dan keluaran itu mengarah. Masuk ke suatu alamat dapat berarti keluar dari alamat yang lain, dan demikian pula sebaliknya.

Dalam hal ini prosesor dijadikan sebagai alamat acuan untuk masuk atau keluar. Masukan berarti masuk menuju prosesor atau menuju piranti yang sedang dikelola oleh prosesor. Keluaran artinya keluar dari prosesor atau dari piranti yang sedang dikelola oleh prosesor. Dengan demikian, dapat dinamakan sebagai suatu piranti masukan manakala piranti itu memasukan informasi ke prosesor atau memori kerja. Cara serupa, dapat dinamakan sebagai piranti keluaran manakala piranti itu menerima informasi dari prosesor atau memori kerja.

Selain tahu dimana saja letak peripheral, prosesor juga harus dapat mengendalikan piranti peripheral itu. Pengendalian itu terdiri atas dua bagian. Bagian pertama adalah pengaturan perangkat keras yang berupa penggerak piranti (device controller) serta bagian kedua adalah pengaturan perangkat lunak berupa protocol transfer data (data transfer protocol).

Protocol transfer data dikenal ada lima macam protocol data. Pertama adalah protocol transfer data pengendali, kedua adalah protocol transfer data serta pengendali dengan interupsi, ketiga adalah protocol transfer data dengan akses memori langsung , keempat adalah protocol transfer data dengan penggerak piranti, serta kelima adalah protocol transfer data bebas piranti.

Kerjasama antara penggerak piranti dengan protocol transfer data memungkinkan prosesor mengendalikan piranti peripheral. Biasanya, pengendali piranti telah disiapkan oleh perusahaan pembuat piranti peripheral serta disesuaikan dengan sistem komputer dimana piranti peripheral itu dipasang. Adakalanya, bersama-sama dengan piranti penghubung lainnya, pengendali piranti terpasang pada kartu antar muka (interface card). Dengan memasang kartu antar muka ke sistem komputer, maka telah dapat memasang juga pengendali piranti yang siap diperintah oleh pengendali aplikasi atau oleh pemakai komputer.

Biasanya piranti peripheral terdiri atas bagian mekanik dan bagian elektronika. Kalau bagian mekanik adalah piranti peripheral itu sendiri, maka bagian elektronika yang mengatur kerja piranti mekanik itu, atau biasanya dinamakan penggerak piranti (device controller) atau adapter. Dengan demikian, pada sejumlah piranti yang memiliki penggerak piranti tersebut berbentuk rangkaian elektronika. Bahkan dalam banyak hal, rangkaian elektronika itu disusun dalam suatu papan rangkaian tercetak (printed circuit). Dengan adanya rangkaian ini, maka sistem operasi selalu berurusan dengan penggerak piranti serta tidak berurusan langsung dengan pirantinya.

Salah satu sebab mengapa sistem operasi selalu berhubungan dengan penggerak piranti dan tidak dengan piranti itu sendiri, adalah karena pada umumnya, piranti

perangkat keras merupakan piranti yang cukup kasar. Penggunaan piranti memerlukan kegiatan yang cukup rumit sehingga dengan membebankan kerumitan iti pada penggerak piranti, sistem operasi tidak perlu terjun ke dalam kerumitan itu.

Setiap penggerak piranti memiliki register untuk mencatat data serta melalui bus, penggerak piranti berhubungan dengan prosesor. Dengan register tersebut, penggerak piranti memonitor status piranti, mengendalikan pengendali piranti pada motor, melaksanakan pemeriksaan data serta mengetahui format data dari piranti. Selanjutnya, penggerak diperlukan untuk mengubah perintah prosesor ke dalam pulsa listrik yang sepadan untuk diterapkan kepiranti. Sebaliknya, penggerak piranti juga mengubah informasi tentang status piranti ke dalam bentuk yang dapat dipahami oleh prosesor.

Dengan demikian, penggerak piranti menggerakkan piranti secara elektronika. Pada piranti perekam berbentuk disk atau disket, penggerak piranti mengatur pemutaran disk atau disket itu melalui motor listrik serta mengatur pula gerakan head tulis baca (read write head) pada disk atau disket itu. Pada pencetak, penggerak piranti melaksanakan gerakan head cetak sesuai dengan arah yang ditentukan. Dan demikian seterusnya, penggerak piranti menggerakan piranti peripheral yang bersangkutan dengan penggerak piranti itu.

Setelah piranti itu digerakan oleh penggerak piranti, maka kerja piranti itu selanjutnya perlu diatur melalui suatu pengendali atau subrutin. Salah satu cara pengaturan adalah dengan melalui sebuah protocol transfer data pengendali. Protocol ini dikenal juga sebagai programmed data transfer protocol. Sesuai dengan namanya, pada protokol ini, transfer data diatur oleh pengendali. Pada saat data akan ditransfer dari prosesor kepiranti atau dari piranti ke prosesor, pengendali membuat sehingga semua permohonan interupsi diabaikan. Setelah itu, transfer data dilaksanakan.

Dengan demikian, pada protocol tersebut, tidak dapat mengenal interupsi melalui permintaan. Sekali transfer data dilaksanakan, maka pelaksanaan akan berlangsung sampai selesai, kecuali tentunya kalau muncul interupsi dan jenis interupsi yang tak terabaikan. Protokol transfer data pengendali ini sering memanfaatkan pustaka (library) dan spool (simultameous peripheral operation on line). Pada pustaka, dapat dilakukan pemanggilan rutin masukan keluaran tertentu dan bahkan dapat mengatur format masukan keluaran itu. Pada spool, dapat dilakukan pengaturan piranti yang tak dapat dipakai bersama yakni piranti seperti pencetak atau panel kunci ketik.

Protokol transfer data pengendali dengan interupsi. Protokol ini juga dikenal dengan nama programmed interrupt data transfer protocol. Sesuai dengan namanya, protocol ini masih mengenal interupsi melalui permintaan. Dengan demikian, setiap terjadi interupsi, maka interupsi itu dilayani. Pada saat itu, transfer data terputus, untuk kemudian dilanjutkan lagi setelah interupsi selesai. Selama tiada interupsi, maka transfer data dapat terus

berlangsung, sampai pada saat transfer data itu selesai.

Dengan menerima interupsi, maka proses yang tadinya sudah terhenti, kini memiliki peluang melanjutkan kembali. cara melanjutkan proses yang sudah terhenti itu adalah melalui interupsi. Protokol transfer data dengan akses memori langsung (direct memory access) merupakan suatu proses yang cukup rumit. Mula-mula penggerak piranti membaca data di dalam blok secara berurutan. Setelah itu, penggerak piranti perlu memeriksa apakah data yang dibaca itu tidak mengandung kekeliruan. Jika tidak terdapat kekeliruan, maka prosesor akan membaca semua data dari atau ke memori kerja melalui penampung (buffer). Karena prosesor harus terlihat dalam pembacaan data, maka selama pembacaan dan penulisan itu berlangsung, prosesor tidak dapat mengerjakan pekerjaan lain. Untuk membebaskan prosesor dari aktifitas ini, maka diciptakan penggerak yang dapat mendukung protocol transfer data akses memori langsung (*direct memory access*).

Pada protokol ini, prosesor diinterupsi pada saat transfer data dimulai. Setelah itu, prosesor tidak lagi ikut mencampuri kegiatan transfer data itu. Kemudian, pada saat transfer data selesai, barulah prosesor diinterupsi sekali lagi. Dengan demikian, di antara saat awal dan saat akhir transfer data itu, prosesor dapat melaksanakan pekerjaan lain.

Karena protocol transfer data melalui akses memori ini membebaskan prosesor untuk melaksanakan pekerjaan lain, maka protocol ini lebih unggul dari kedua protocol lainnya.

Protokol transfer data dengan penggerak piranti. Penggerak piranti juga dikenal sebagai devide driver. Penggerak piranti ini berbentuk piranti lunak yang menghubungkan prosesor dengan alat, tentunya melalui penggerak alat. Bahkan register pada penggerak alat dimanfaatkan oleh penggerak alat untuk menyalurkan informasi dari prosesor kealat dan demikian pula sebaliknya.

Dalam keadaan tertentu, satu alat dapat berhubungan dengan beberapa penggerak piranti sejenis. Satu piranti disk atau disket, misalnya dapat berhubungan dengan satu atau lebih penggerak piranti disk atau disket. Pada saat kegiatan, penggerakan piranti berbentuk proses yang mengendalikan kerja piranti peripheral. Di antaranya, proses tersebut menerima permintaan piranti masukan keluaran (ada kalanya berbentuk suatu antrian), memulai kerja masukan keluaran, menata kekeliruan umum pada penyaluran informasi, melaksanakan interupsi, serta mengirim berita selesai kembali ke proses.

Ada yang mengatakan bahwa tugas utama proses pada penggerak piranti mencakup mencegah permintaan dari satu proses, melaksanakan kerja tertentu pada prose situ, serta memberitahukan proses yang meminta itu tentang hasil kerja yang telah terlaksana. Proses yang meminta itu adalah proses umum yang tidak tergantung kepada piranti (tidak khas piranti tertentu). Dengan demikian, penggerak piranti menerima perintah umum serta melaksnakan perintah iti pada piranti peripheral.

Rincian dari proses itu sendiri berbeda dari piranti menuju alat, misal, penggerak disk (disk driver) merupakan bagian satu-satunya pada sistem operasi yang mengetahui berapa register yang dimiliki oleh penggerak disk serta apa gunanya register itu. Pelaksana disk itu adalah satu-satunya yang mengetahui seluk beluk sector, lintas (track), silinder, hulu, gerak tangkai hulu, *factor seling* (interleave), waktu pengaturan hulu, serta segala sesuatu yang dapat membuat disk itu bekerja secara benar.

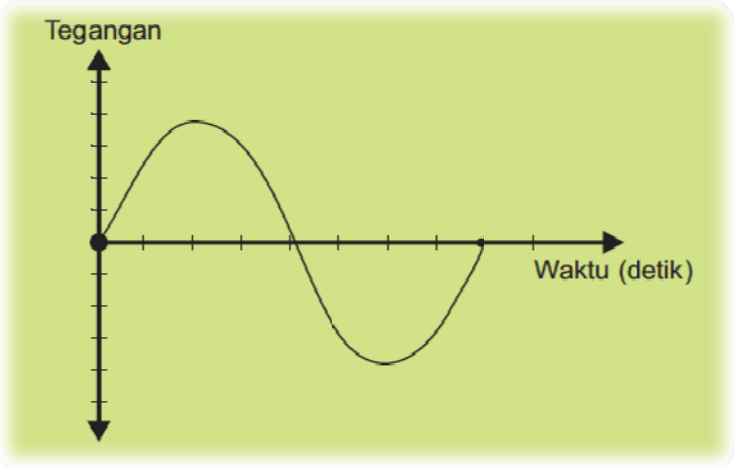
Penggerak piranti bekerja sama secara erat dengan penggerak piranti. Karena itu, ada orang yang menamakan kedua-duanya sebagai penggerak piranti saja (device controller) atau sebagai penggerak piranti (device driver) saja. Di sini, mereka tetap kita pisahkan yakni sebagai penggerak piranti dan sebagai penggerak piranti, sedangkan secara bersama-sama, mereka kita namakan sebagai pengendali piranti.

1.4. Kerja Komputer

Komputer yang merupakan mesin penghitung sekarang ini mesin tersebut banyak dipakai untuk pemroses data. Fungsinya sangat sederhana yaitu digunakan untuk memproses data, kemudian hasil prosesnya diselesaikan secara elektronis didalam Central Processing Unit (CPU) dan komponen lainnya yang menyusun sebuah komputer personal.

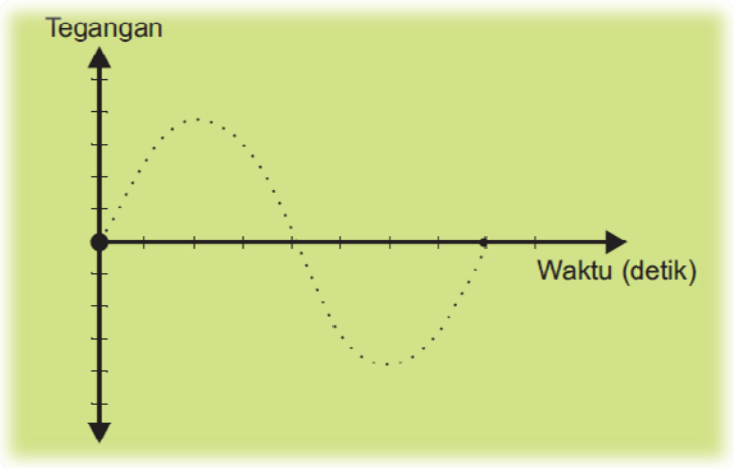
Suatu sinyal yang dikirimkan dari suatu pemancar (transmitter) ke penerima (receiver) untuk berkomunikasi adalah berupa data.

Data-data yang biasa dijumpai sehari-hari memiliki banyak bentuk, antara lain: suara, huruf, angka, dan karakter lain (tulisan tangan atau dicetak), foto, gambar, film dan lain sebagainya. Suatu sistem yang dapat memproses nilai yang kontinyu berbanding terhadap waktu dinamakan sistem analog. Pada sistem analog, nilainya biasa diwakili oleh tegangan, arus dan kecepatan. Berikut ini adalah gambar grafik nilai tegangan analog terhadap waktu.



Gambar 1.7 Grafik Nilai Tegangan Analog Terhadap Waktu

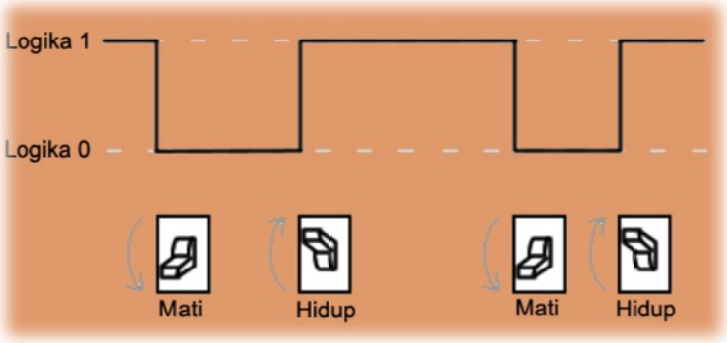
Sistem yang memproses nilai diskrit (langkah demi langkah) dinamakan digital. Pada sistem digital untuk menunjukkan suatu nilai digunakan simbol yang dinamakan digit. Sinyal pada gambar diatas dapat "didigitalkan" dengan menggunakan Analog to Digital Converter (ADC). ADC mengubah sinyal kontinyu menjadi sinyal diskrit dengan menyamplingnya tiap detik (tiap satuan waktu). Perhatikan gambar dibawah ini.



Gambar 1.8. Sinyal Diskrit Dengan Penyamplingan Tiap Detik

Komputer adalah sebuah perangkat elektronik. Data yang dapat diolah adalah data yang direpresentasikan oleh sinyal listrik. Sinyal yang digunakan bisa dianalogikan dengan saklar listrik,

yaitu tombol off (mati) atau on (hidup). Jika saklar pada kondisi off, maka komputer membaca sebagai data 0, jika saklar dalam kondisi hidup, maka komputer membaca sebagai angka 1. Perhatikan gambar berikut:



Gambar 1.9. Saklar Dalam Kondisi Hidup Dan Mati

Sebuah komputer terdiri dari saklar-saklar yang banyak jumlahnya saklar-saklar ini menggunakan komponen elektronik berupa transistor. Jumlah transistor yang digunakan bisa sampai jutaan, sehingga dapat memproses data dari jutaan angka 0 dan 1.

Setiap angka 0 dan 1 biasa disebut Bit. Bit adalah singkatan dari *Binary Digi**t***. Kata Binary diambil dari nama Sistem Bilangan Biner (Binary Number Sistem). Tabel dibawah berikut menunjukkan tentang bit:

Tabel 1.1 Bilangan Biner dengan besar bit data

0	**1 bit**
1	**1 bit**
0110	**4 bit**
10011101	**8 bit**

Sistem bilangan biner disusun dari angka-angka, sama seperti sistem bilangan desimal (sistem bilangan 10) yang sering digunakan saat ini. Tetapi untuk desimal menggunakan angka 0 sampai 9, sistem bilangan biner hanya menggunakan angka 0 dan 1.

Pengolahan data yang paling sering digunakan saat ini adalah pengolah kata (*word processing*).. Ketika melakukan suatu pengolahan kata, komputer bekerja dengan keyboard. Ada 101 tombol yang mewakili karakter alphabet A, B, C, dan seterusnya. Selain itu juga akan

ditemui karakter angka 0 sampai dengan 9, dan karakter-karakter lain yang diperlukan, antara lain: **, . - ; ( ) : _ ? ! " # * % & .**

Seluruh karakter yang ada pada keyboard harus didigitalkan. Karakter-karakter tersebut diwakili oleh angka-angka 0 dan 1. Bit yang digunakan adalah 8 bit biner. 8 bit biner dinamakan Byte.

Pada sistem bilangan biner, banyaknya kombinasi dihitung dengan $2^n \leq m$, dimana n adalah jumlah bit, m adalah kombinasi yang dapat diwakili, sehingga pada 8 bit biner, dapat mewakili 2^8=256 kombinasi maksimal.

Ketika mengetik kata "digital" simbol yang digunakan adalah 6 huruf, saat komputer mengolahnya, 6 huruf tersebut didigitalkan menjadi 6 bytes, yang kemudian disimpan pada RAM komputer saat mengetik, dan kemudian disimpan pada harddisk, jika disimpan. Tabel seperti dibawah menunjukkan perbandingan ukuran unit data.

Tabel 1.2. Perbandingan Ukuran Unit Data

UNIT	DEFINISI	BYTE	BIT	CONTOH APLIKASI
Bit (b)	Binary Digit, 0 dan 1	1	1	On/Off, buka/tutup
Byte (B)	8 bit	1	8	Kode ASCII
Kilobyte (KB)	1.024 byte	1000	8000	Ukuran email biasa = 2 KB 10 halaman dokumen= 10 KB
Megabyte (MB)	1.024 kilobyte 1.048.576 byte	1 juta	8 juta	Floppy disks = 1,44 MB CDROM = 650 MB
Gigabyte (GB)	1.024 megabyte 1.073.741.824 byte	1 milyar	8 milyar	Hard drive = 40 GB
Terrabyte (TB)	1.024 gigabyte	1 trilyun	8 trilyun	Data yang dapat dikirim pada fiber optik selama 1 detik.

1.5. Sistem Bilangan

Sistem bilangan yang paling umum digunakan adalah sistem bilangan desimal, biner, oktal, dan heksadesimal. Sistem bilangan desimal merupakan sistem bilangan yang paling banyak digunakan oleh manusia karena berbagai kemudahannya untuk dipergunakan sehari-hari. Sistem bilangan biner merupakan sistem bilangan yang paling banyak digunakan dalam sistem digital karena sistem bilangan ini secara langsung dapat mewakili logika yang ada, sedangkan sistem bilangan oktal dan heksadesimal biasanya banyak digunakan dalam sistem digital untuk memperpendek penyajian suatu bilangan yang tadinya disajikan dalam sistem bilangan biner, sehingga lebih mudah dipahami atau dihafalkan.

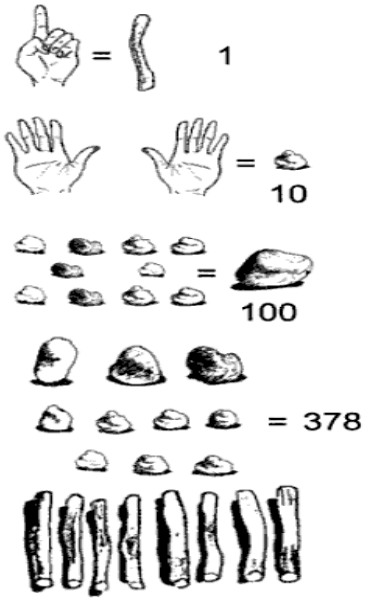
Secara umum bilangan dapat dibagi menjadi beberapa kategori. Ditinjau dari segi koma desimal (point), bilangan dapat dibagi menjadi bilangan bulat (integer number/fixed-point number) dan bilangan pecahan (*floating-point number*). Dilihat dari dari segi tanda, bilangan dapat dibagi menjadi bilangan tak bertanda (unsigned number) dan bilangan bertanda (signed number).

1.5.1. Bilangan Desimal

Sistem bilangan desimal disusun oleh 10 angka atau lambang. Dengan menggunakan lambang-lambang tersebut sebagai digit pada sebuah bilangan, maka akan dapat mengekspresikan suatu kuantitas. Kesepuluh lambang tersebut adalah:

D = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }

Sistem bilangan desimal disebut juga sistem bilangan basis 10 atau radiks 10 karena mempunyai 10 digit. Sistem bilangan ini bersifat alamiah karena pada kenyataannya manusia mempunyai 10 jari. Kata digit itu sendiri diturunkan dari kata bahasa Latin *finger*.



Gambar 1.10. Bilangan desimal

Ciri bilangan yang menggunakan sistem bilangan desimal adalah adanya tambahan subskrip des atau 10 atau tambahan D di akhir suatu bilangan.

$357_{des} = 357_{10} = 357D.$

Namun karena bilangan desimal sudah menjadi bilangan yang digunakan sehari-hari, subskrip tersebut biasanya dihilangkan. Sistem bilangan desimal merupakan sebuah sistem nilai posisi.

Bilangan 357.

Pada bilangan tersebut, digit 3 berarti 3 ratusan, 5 berarti 5 puluhan, dan 7 berarti 7 satuan. Sehingga, 3 mempunyai arti paling besar di antara tiga digit yang ada. Digit ini bertindak sebagai digit paling berarti (Most Significant Digit, MSD). Sedangkan 7 mempunyai arti paling kecil di antara tiga digit yang ada dan disebut digit paling tidak berarti (Least Significant Digit, LSD).

Bilangan 35,27

Bilangan ini mempunyai arti 3 puluhan ditambah 5 satuan ditambah 2 per sepuluhan. Koma desimal memisahkan pangkat positif dari 10 dengan pangkat negatifnya.

$$35{,}2 = 3 \times 10^{1} + 5 \times 10^{0} + 2 \times 10^{-1}$$

Secara umum dapat dikatakan, nilai suatu bilangan desimal merupakan penjumlahan dari perkalian setiap digit dengan nilai posisinya.

1.5.2. Bilangan Biner

Sistem sistem digital hanya mengenal dua logika, yaitu 0 dan 1. Logika 0 biasanya mewakili kondisi mati dan logika 1 mewakili kondisi hidup. Pada sistem bilangan biner, hanya dikenal dua lambang, yaitu 0 dan 1. Karena itu, sistem bilangan biner paling sering digunakan untuk merepresentasikan kuantitas dan mewakili keadaan dalam sistem digital maupun sistem komputer.

Digit bilangan biner disebut binary digit atau bit. Empat bit dinamakan nibble dan delapan bit dinamakan byte. Sejumlah bit yang dapat diproses komputer untuk mewakili suatu karakter (dapat berupa huruf, angka atau lambang khusus) dinamakan word. Sebuah komputer dapat memproses data satu word yang terdiri dari 4 sampai 64 bit. Sebagai contoh, sebuah komputer yang menggunakan mikroprosesor 32 bit dapat menerima, memproses, menyimpan dan mengirim data atau instruksi dalam format 32 bit.

Pada komputer yang digunakan untuk memproses karakter, maka karakter (yang meliputi huruf, angka, tanda baca dan karakter kontrol) tersebut harus diformat dalam bentuk kode alfanumerik. Format kode ASCII (American Standard Code for Information Interchange) menggunakan format data tujuh bit untuk mewakili semua karakter yang ada termasuk tanda baca dan penanda kontrol. Dengan menggunakan format tujuh 7 bit tersebut, maka ASCII dapat menanpung 2^{7} = 128 data. Sistem bilangan biner merupakan sistem bilangan basis dua. Pada sistem bilangan ini hanya dikenal dua lambang, yaitu:

$$B = \{ 0, 1 \}$$

Ciri suatu bilangan yang menggunakan sistem bilangan biner adalah adanya tambahan subskrip bin atau 2 atau tambahan huruf B di akhir suatu bilangan.

$$1010011_{bin} = 1010011_{2} = 1010011B.$$

Bit paling kiri dari suatu bilangan biner bertindak sebagai bit paling berarti (Most Significant Bit, MSB), sedangkan bit paling kanan bertindak sebagai bit paling tidak berarti (Least Significant Bit, LSB).

1.5.3. Bilangan Oktal

Sistem bilangan oktal merupakan sistem bilangan basis delapan. Pada sistem bilangan ini terdapat delapan lambang, yaitu:

$$O = \{ 0, 1, 2, 3, 4, 5, 6, 7 \}$$

Ciri suatu bilangan menggunakan sistem bilangan oktal adalah adanya tambahan subskrip okt atau 8 atau tambahan huruf O di akhir suatu bilangan.

$1161_{okt} = 1161_{8} = 1161O$

1.5.4. Bilangan Heksadesimal

Sistem bilangan heksadesimal merupakan sistem bilangan basis enambelas. Meskipun operasi pada sistem digital dan komputer secara fisik dikerjakan secara biner, namun untuk merepresentasikan data menggunakan format bilangan heksadesimal karena format ini lebih praktis, mudah dibaca dan mempunyai kemungkinan timbul kesalahan lebih kecil.

Penerapan format bilangan heksadesimal banyak digunakan pada penyajian lokasi memori, penyajian isi memori, kode instruksi dan kode yang merepresentasikan alfanumerik dan karakter nonnumerik. Pada sistem bilangan ini terdapat enam belas lambang, yaitu:

H = { 0,1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F }

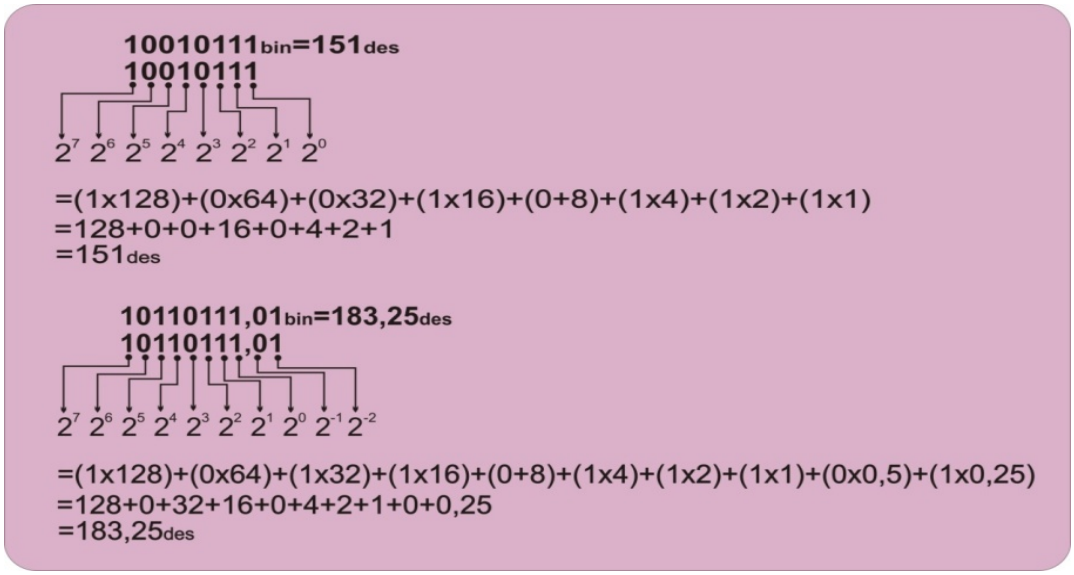
Ciri suatu bilangan menggunakan sistem bilangan heksadesimal adalah adanya tambahan subskrip heks atau 16 atau tambahan huruf H di akhir suatu bilangan.

$271_{heks} = 271_{16} = 271H.$

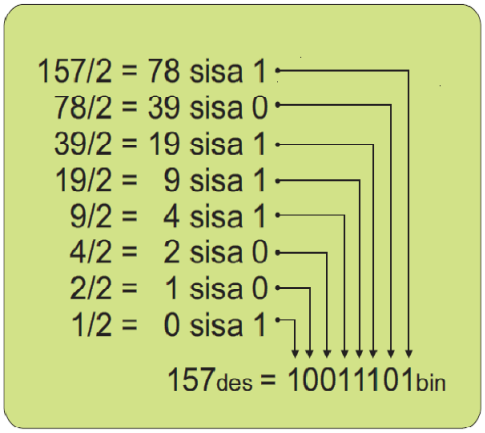
1.5.5. Konversi Bilangan

Sistem bilangan dapat dilakukan konversi menjadi bilangan yang diinginkan. Tujuan konversi ini adalah untuk menjembatani antara manusia dan mesin dalam hal ini komputer supaya dapat berkomunikasi. Manusia lebih mudah memahami bilangan decimal karena memang bilangan decimal merupakan bilangan yang digunakan manusia sehari-hari, sedangkan komputer bekerja dengan bilangan biner.

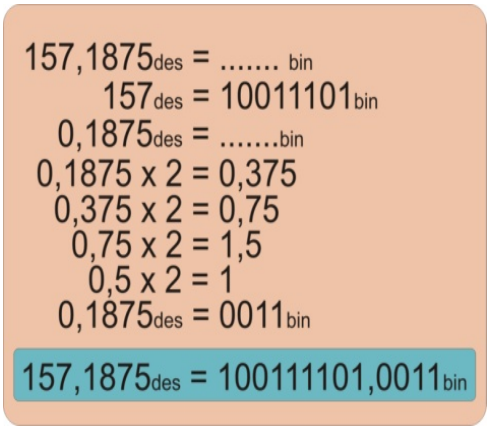
Komputer yang bekerja dengan bilangan biner tentunya harus bisa disingkat baik penulisan maupun pembacaannya dalah hal ini menggunakan bilangan hexadecimal atau oktal. Karena komputer hanya paham terhadap sat bilangan yaitu biner maka manusia dituntut untuk bisa memahami bagaimana konversi antara bilangan-bilangan tersebut. Konversi bilangan biner ke desimal dilakukan dengan menjumlahkan hasil perkalian semua bit biner dengan beratnya.

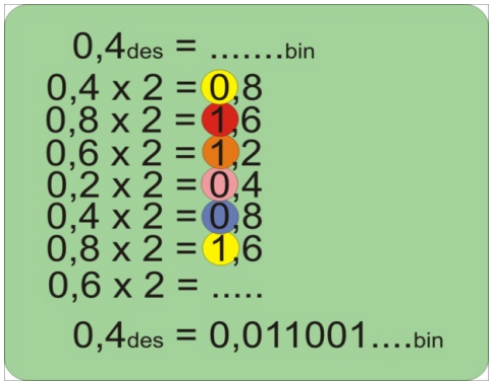


Konversi bilangan desimal bulat ke biner dilakukan dengan membagi bilangan desimal dengan 2 secara berulang-ulang sehingga akan dapat diketahui sisa tiap operasi pembagian. Sisa yang dihasilkan setiap pembagian merupakan bit yang didapat. Dengan cara mengurutkan bit-bit tersebut dari bawah keatas maka dapat diketahui hasil konversi yang telah dilakukan.

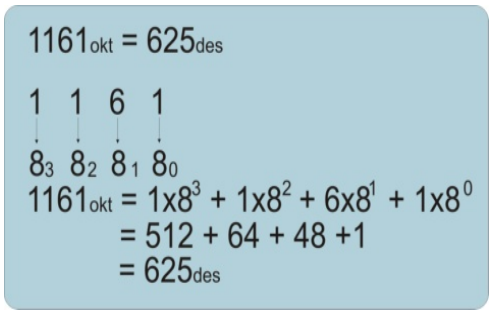


Sebuah bilangan desimal real dapat pula dikonversi ke bilangan real biner. Konversi dilakukan dengan cara memisahkan antara bagian bulat dan bagian pecahannya. Konversi bagian bulat dapat dilakukan seperti pada gambar diatas. Sedangkan konversi bagian pecahan dilakukan dengan mengalikan pecahan tersebut dengan 2, kemudian bagian pecahan dari hasil perkalian tersebut dikalikan dengan 2. Langkah tersebut diulang-ulang sehingga mendapatkan hasil akhir 0. Bagian bulat dari setiap hasil perkalian merupakan bit yang didapat.

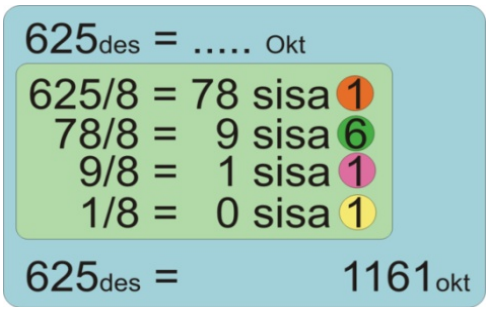




Konversi bilangan oktal ke desimal dilakukan dengan menjumlahkan hasil perkalian semua digit oktal dengan beratnya.

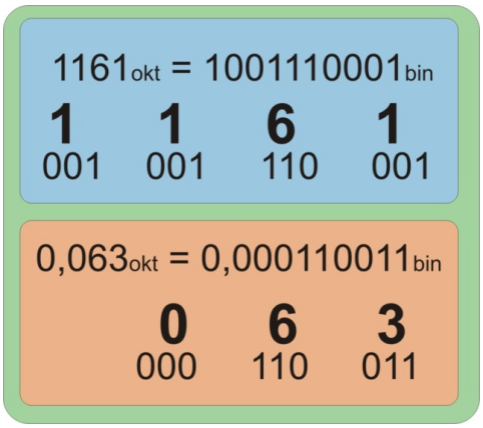


Konversi bilangan bulat desimal ke oktal dilakukan dengan membagi secara berulang-ulang suatu bilangan desimal dengan 8. Sisa setiap pembagian merupakan digit oktal yang didapat.



Konversi bilangan desimal pecahan ke oktal dilakukan dengan cara hampir sama dengan konversi bilangan desimal pecahan ke biner, yaitu dengan mengalikan suatu bilangan desimal pecahan dengan 8. Bagian pecahan dari hasil perkalian ini dikalikan dengan 8. Langkah ini diulang hingga didapat hasil akhir 0. Bagian bulat dari setiap hasil perkalian merupakan digit yang didapat.

Konversi bilangan oktal ke biner lebih mudah dibandingkan dengan konversi bilangan oktal ke desimal. Satu digit oktal dikonversi ke 3 bit biner.



Konversi bilangan biner ke oktal lebih mudah dibandingkan konversi bilangan desimal ke oktal. Untuk bagian bulat, kelompokkan setiap tiga bit biner dari paling kanan, kemudian konversikan setiap kelompok ke satu digit oktal. Dan untuk bagian pecahan, kelompokkan setiap tiga bit biner dari paling kiri, kemudian konversikan setiap kelompok ke satu digit oktal. Proses ini merupakan kebalikan dari proses konversi bilangan oktal ke biner.

$1\ 001\ 110\ 001_{bin} = 1161_{okt}$

$1\ 1\ 6\ 1$

Konversi bilangan heksadesimal ke desimal dilakukan dengan menjumlahkan hasil perkalian semua digit heksadesimal dengan beratnya.

$271_{hex} = 625_{des}$

$\mathbf{2\ \ 7\ \ 1}$

$16^2\ \ 16^1\ \ 16^0$

$271hex = 2x16^2 + 7x16^1 + 1x16^0$

$= 512 + 112 + 1$

$= 625_{des}$

Konversi bilangan desimal bulat ke heksadesimal dilakukan dengan membagi secara berulang-ulang suatu bilangan desimal dengan 16. Sisa setiap pembagian merupakan digit heksa-desimal yang didapat.

$625_{des} = 271_{hex}$

625/16 = 29 sisa 1

39/16 = 2 sisa 1

2/16 = 0 sisa 2

$625_{des} = 271_{hex}$

Konversi bilangan desimal pecahan ke heksadesimal dilakukan dengan cara hampir sama dengan konversi bilangan desimal pecahan ke biner, yaitu dengan mengalikan suatu bilangan desimal pecahan dengan 16. Bagian pecahan dari hasil perkalian ini dikalikan dengan 16. Langkah ini diulang hingga didapat hasil akhir 0. Bagian bulat dari setiap hasil perkalian merupakan digit yang didapat.

$625{,}1875_{des} = \ldots_{bin}$

$0{,}75_{des} = 0{,}C_{heks}$

0,75 X 16 = C

$0{,}1_{des} = 0{,}19 \ldots\ldots_{heks}$

0,10 X 16 = 1,6

0,60 X 16 = 9,6Konversi

Konversi bilangan heksadesimal ke biner lebih mudah dibandingkan konversi bilangan heksadesimal ke desimal. Satu digit heksadesimal dikonversi ke 4 bit.

$271_{hex} = 1001110001_{bin}$	
Sehingga menjadi	
1	**= 0001**
7	**= 0111**
2	**= 0010**

1001110001_{bin}

Konversi bilangan biner ke heksadesimal lebih mudah dibandingkan konversi bilangan desimal ke heksadesimal. Untuk bagian bulat, kelompokkan setiap empat bit biner dari paling kanan, kemudian konversikan setiap kelompok ke satu digit heksadesimal.

Pada bagian bilangan pecahan, kelompokkan setiap empat bit biner dari paling kiri, kemudian konversikan setiap kelompok ke satu digit heksadesimal. Proses ini merupakan kebalikan dari proses konversi bilangan heksadesimal ke biner.

$1001110001_{bin} = 271_{hex}$		
Jika dipecah menjadi		
	0001	**= 1**
	0111	**= 7**
	0010	**= 2**

271_{hex}

1.5.6. Data Karakter

Beberapa aplikasi menggunakan data yang bukan hanya bilangan tetapi juga huruf dari alfabet dan karakter khusus lainnya. Data semacam ini disebut dengan data alfanumerik dan mungkin dapat ditunjukkan dengan kode numerik. Jika bilangan-bilangan dimasukkan dalam data, maka bilangan-bilangan tersebut juga dapat ditunjukkan dengan kode khusus.

Set karakter alfanumerik secara khusus mencakup 26 huruf alfabet (termasuk huruf besar dan huruf kecil), angka dalam digit sepuluh desimal, dan sejumlah simbol seperti +, =, *, $, …, dan !. Dua kode alfabet yang paling umum dipakai adalah ASCII (American Standard Code for Information Interchange) dan EBCDIC (Extended Binary Coded Decimal Interchange Code). ASCII merupakan kode 7-bit dan EBCDIC berupa kode 8 bit. Jika suatu komputer menangani 8-bit (1-byte) kode lebih efisien, versi 8-bit, disebut dengan ASCII-8 juga telah dikembangkan.

Selain itu ada juga beberapa kode spesial didalam penambahan set karakter alfanumerik. Kode simpanan ini digunakan sebagai signal komunikasi dalam aplikasi dimana data transfer terjadi antara komputer yang dihubungkan melalui baris komunikasi. Misalnya, LF (*line feed*) dan CR (*carriage return*) dihubungkan dengan printer, BEL

digunakan untuk mengaktifitaskan bell; ACK (acknowledge), NAK (negative acknowledge), dan DLE (data link escape) berupa signal yang dapat diubah dalam baris komunikasi.

Orang yang sudah cukup lama berkecimpung di dunia komputer, pasti pernah bekerja dengan 'kode ASCII'. Dan bagi yang bekerja dengan mesin-mesin mainframe IBM, pasti pernah menjumpai 'kode EBCDIC'. Di luar ASCII dan EBCDIC, besar kemungkinan anda paling tidak pernah mendengar istilah-istilah lain seperti berikut ini: ISO-8859-1, UCS-2, UTF-8, UTF-16, atau windows-1252. Kode-kode apakah itu? ASCII, EBCDIC, ISO-8859-X, UCS-2, UTF-X, dan windows-x merupakan sebagian dari kumpulan character set (set karakter) yang ada di dunia komputer.

1.5.7. Kode BCD

Sebelum ASCII dan EBCDIC berkembang terlebih dahulu dikembangkan Binary Coded Decimal (BCD). Metode ini awalnya digunakan pada komputer mainframe IBM. Pada grup ini karakter diwakili oleh 64 - (2^6) lambang. Dengan kode ini, setiap huruf/angka diberikan kode yang terdiri dari enam bit, dua untuk zone dan empat untuk angka. Huruf A sampai dengan I diberikan tanda 11 pada tempat zone. Karena A adalah huruf pertama dalam kelompok ini, maka kodenya adalah: 0001, B sebagai huruf kedua dengan kode: 0010, C adalah 0011 dan seterusnya. Dengan perkataan lain, zone bit yang mempunyai formasi 11 harus juga disertakan pada kode lengkap masing-masing pada grup ini.

Grup alfabetik kedua adalah J hingga R, ditetapkan kode awalnya 10, yang juga posisi masing-masing huruf ditentukan oleh angkanya masing-masing. Huruf S hingga Z dibentuk dengan menambahkan angka bit 0010 hingga 1001 berurutan pada kode 01 dimana pada grup ini hanya ada delapan huruf. Angka-angka 0 hingga sembilan diberikan kode 00 di depannya diikuti oleh angka itu sendiri dalam sistim binary. Angka 0 (nol) harus dibedakan dengan tanda kosong (spasi) guna mempermudah cara penggunaan kode.

1.5.8. Kode EBCDIC

Sistim Extended Binary Coded Decimal Interchange Code (EBCDIC) merupakan set karakter yang merupakan ciptaan dari IBM. Salah satu penyebab IBM menggunakan set karakter di luar ASCII sebagai standar pada komputer ciptaan IBM adalah karena EBCDIC lebih mudah dikodekan pada punch card yang pada tahun 1960-an masih jamak digunakan.

Penggunaan EBCDIC pada mainframe IBM masih terbawa hingga saat ini, walaupun punch card sudah tidak digunakan lagi. Seperti halnya ASCII, EBCDIC juga terdiri

dari 128 karakter yang masing-masing berukuran 7-bit. Bila menggunakan ukuran 8-bit maka karakternya menjadi 256 atau 2^8. Hampir semua karakter pada ASCII juga terdapat pada set karakter EBCDIC.

1.5.9. Kode ASCII

ASCII (Sistem American Standard Code for Information Interchange**)** dan EBCDIC merupakan awal dari set karakter lainnya. ASCII merupakan set karakter yang paling umum digunakan hingga sekarang. Set karakter ASCII terdiri dari 128 – (2^7) buah karakter yang masing-masing memiliki lebar 7-bit atau gabungan tujuh angka 0 dan 1, dari 0000000 sampai dengan 1111111. Mengapa 7-bit? Karena komputer pada awalnya memiliki ukuran memori yang sangat terbatas, dan 128 karakter dianggap memadai untuk menampung semua huruf Latin dengan tanda bacanya, dan beberapa karakter kontrol. ASCII telah dibakukan oleh ANSI (American National Standards Institute) menjadi standar ANSI X3.4-1986.

1.6. Pemrograman Komputer

Komputer sebuah mesin yang dirancang untuk mengikuti instruksi. Program komputer merupakan sebuah instruksi yang digunakan oleh komputer untuk memecahkan masalah atau tugas-tugas yang diberikan padanya. Misalnya jika menggunakan komputer untuk menghitung pembayaran pada seorang pegawai maka langkah-langkah yang dilakukan adalah sebagai berikut:

1. Menampilkan pesan pada layar " berapa jam kamu bekerja?"
2. Menunggu pengguna memasukan data jam bekerja dan setelah ditekan ENTER maka komputer akan memasukan data ke dalam memory.
3. Menampilkan pesan pada layar " berapa besarnya gaji tiap jamnya?"
4. Menunggu pengguna memasukan data per jam bekerja dan setelah ditekan ENTER maka komputer akan memasukan data ke dalam memory
5. Komputer melakukan perkalian antara jumlah jam dengan gaji perjam
6. Komputer menampilkan hasil perhitungan gaji pegawai yang harus dibawa pulang.

Sekumpulan instruksi-instruksi tersebut diatas disebut dengan algoritma. Algoritma merupakan kumpulan instruksi yang terdefinisi langkah demi langkah secara baik dengan tujuan untuk menyelesaikan masalah. Catatan langkah-langkah ini diurutkan secara sekensial. Dalam algoritma langkah pertama harus dilakukan lebih dahulu sebelum langkah kedua dan seterusnya. Langkah-langkah komputer tidak bisa terbalik. Pada langkah-langkah

diatas, komputer diinstruksikan untuk menghitung besarnya gaji pegawai, untuk dapat bekerja maka langkah-langkah tersebut harus dikonversi bahasa yang bisa diproses oleh komputer, karena kenyataannya prosesor hanya dapat memproses instruksi yang ditulis dalam bahasa mesin. Prosesor hanya mengetahui penggunaan bilangan sebagai perintah yang harus dilakukan, oleh karena itu algoritma tersebut harus dikodekan menjadi bahasa mesin.

Penulisan bahasa mesin secara langsung akan sangat sulit dan membosankan. Supaya penulisan bahasa pemrograman komputer menjadi lebih mudah maka digunakan bahasa aras tinggi yang bahasanya sudah mendekati bahasa manusia. Dengan bahasa inilah programer dapat menulis instruksi-instruksi yang akan dilakukan oleh komputer. Pemrograman dengan menggunakan C++ misalnya akan menjadi lebih mudah dan C++ inilah yang akan mengkonversi bahasa program yang ditulis menjadi bahasa mesin. Dibawah ini merupakan program sederhana yang ditulis dengan bahasa C++.

Program 1.1

```
#include <conio.h>
#include <iostream>

using namespace std;

int main()
{
        double jam, besar, bayar;
        cout << "Berapa Jam Kamu bekerja? ";
        cin >> jam;
        cout << "Berapa besar gaji tiap jamnya? ";
        cin >> besar;
        bayar  = jam * besar;
        cout << "Kamu mendapatkan Gaji (Rp)" << bayar << endl;
        getch();
        return 0;
}
```

Keluaran program diatas adalah:

```
Berapa Jam Kamu bekerja? 10
Berapa besar gaji tiap jamnya? 15000[enter]
Kamu mendapatkan Gaji (Rp)150000
```

Dengan bahasa pemrograan tersebut diatas, maka instruksi komputer dapat dibuat dengan mudah. Seperti program diatas yang digunakan untuk menghitung gaji pegawai, jika diminta memasukan

data jumlah jam kerja kemudian diisi 10 dan memasukan gaji tiap jamnya 150000 maka komputer akan menghitung dengan mengalikan data tersebut sehingga data keluaran akan muncul 150000.

1.7. Bahasa pemrograman

Bahasa pemrograman adalah notasi yang digunakan untuk menulis program (komputer). Bahasa ini dibagi menjadi tiga tingkatan yaitu bahasa mesin, bahasa tingkat rendah dan bahasa tingkat tinggi.

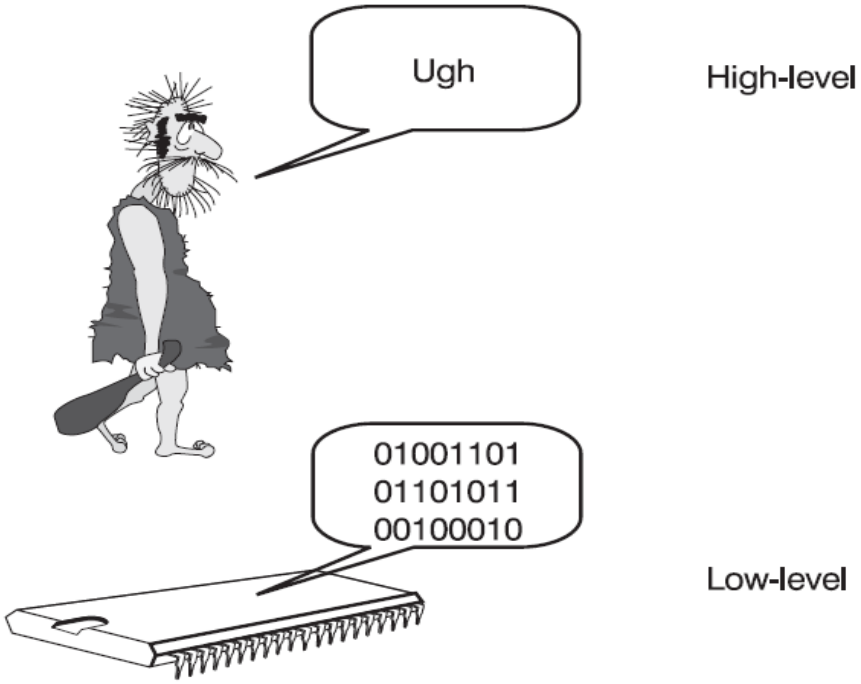
Bahasa mesin (*machine language*) berupa microinstruction atau hardwire. Programnya sangat panjang dan sulit dipahami. Di samping itu sangat tergantung pada arsitektur mesin. Keunggulannya adalah prosesnya sangat cepat dan tidak perlu interpreter atau penterjemah Bahasa tingkat rendah (low level language) berupa macroinstruction (assembly). Seperti halnya bahasa mesin, bahasa tingkat rendah tergantung pada arsitektur mesin. Programnya panjang dan sulit dipahami walaupun prosesnya cepat. Jenis bahasa tingkat ini perlu penterjemah berupa assembler.

Bahasa tingkat tinggi (high level language) menyerupai struktur bahasa manusia sehingga mudah dipahami. Bahasa ini tidak tergantung pada arsitektur mesin tetapi memerlukan penterjemah berupa compiler atau interpreter.

Secara garis besar ada dua kategori bahasa pemrograman yaitu: bahasa pemrograman aras rendah (low level) dan bahasa pemrograman level tinggi (high level). Bahasa pemrograman aras rendah cenderung mendekati level komputer, ini artinya bahwa bahasanya ditulis mendekati atau sama dengan bahasa mesin komputer, hal ini sangat sulit ditulis karena bahasanya jauh dari bahasa manusia yang digunakan sehari-hari.

Bahasa pemrograman yang lebih mudah dipelajari adalah bahasa pemrograman aras tinggi. Disebut aras tinggi karena bahasanya mendekati level bahasa manusia sehingga lebih mudah dipahami. Gambar dibawah ini merupakan gambaran antara bahasa aras tinggi dengan aras rendah.

Gambar 1.11. Bahasa aras tinggi



Gambar 1.12. Bahasa aras rendah

Dengan gambar tersebut diatas, maka dapat dilihat bahwa menulis program dengan bahasa aras tinggi akan lebih mudah dan dapat dipahami dibandingkan dengan bahasa aras rendah karena bahasanya ditulis dengan kode numerik ataupun dengan sintak yang sangat pendek sehingga sulit sekali dipahami. Untuk lebih jelasnya beberapa jenis bahasa pemrograman aras tinggi yang digunakan dapat dilihat pada tabel dibawah ini:

Tabel 1.3. bahasa pemrograman aras tinggi

NAMA	PENJELASAN
BASIC	Beginners All-purpose Symbolic Instruction Code, bahasa pemrograman yang biasa digunakan untuk merancang program sederhana pada programer pemula
FORTRAN	Formula Translator, Bahasa pemrograman yang dirancang untuk menyelesaikan algoritma matematika yang kompleks
COBOL	Common Business-Oriented Language. Bahasa pemrograman yang dirancang pada aplikasi bisnis
Pascal	Pemrograman terstruktur, bersifat umum, dan biasanya bahasa pemrogram ini banyak diajarkan
C	Pemrogram terstruktur, bersifat umum. Bahasa ini dikembangkan oleh bell laboratories. Bahasa C ini dapat digunakan sebagai bahasa aras tinggi dan aras rendah.
C++	Dasar pengembangan C. C++ dapat digunakan sebagai bahasa berorientasi objek, yang tidak ditemukan pada bahasa C. Bahasa ini juga dikembangkan oleh laboratorium Bell
C#	C# atau “C sharp”. Bahasa ini ditemukan oleh microsoft untuk mengembangkan aplikasi pada aplikasi microsoft .NET
Java	Bahasa ini merupakan bahasa berorientasi objek yang dikembangkan oleh Sum Microsistem. Dengan java memungkinkan untuk pengembangan program yang berjalan pada jaringan internet atau pada web browser.
VISUAL BASIC	Bahasa pemrograman microsoft dimana bahasa ini bertujuan untuk pengembangan perangkat lunak yang dapat memudahkan programmer dalam membuat aplikasi berbasis windows.

Dalam memilih bahasa pemrograman harus memperhatikan hal-hal yang dimiliki oleh bahasa tersebut, dan tentunya setiap bahasa pasti mempunyai kekurangan dan kelebihan yang harus kita pertimbangan dalam memilihnya.

Pada bahasa C++ mempunyai kemampuan pada bahasa aras tinggi maupun bahasa aras rendah. Bahasa C++ basisnya adalah pengembangan dari bahasa C, selain itu C++ juga mendukung bahasa pemrograman berorientasi objek. Sebenarnya bahasa ini ari awal dikembangkan untuk menulis program sistem operasi sebuah komputer maupun compiler. Karena bahasa C++ mengembangkan C sebelumnya maka kemampuannya C++ diperbaiki dan ditingkatkan daripada bahasa C.

C++ sangat populer, tidak hanya karena gabungan antara aras tinggi dan aras rendah, tetapi juga karena sifat portabilitas yang dimilikinya, ini artinya C++ dapat ditulis pada satu jenis komputer dan dapat dijalankan pada jenis komputer lain yang berbeda. Hal ini biasanya membutuhkan compiler ulang pada jenis sistem komputer yang akan digunakan, tetapi program yang ditulis tersebut tidak ada perubahan. Salah satu sistem operasi yang ditulis dengan menggunakan C++ adalah X-Window sistem dan sistem operasi yang dimiliki oleh Macintosh.

Ketika progam C++ ditulis, hal tersebut harus disesuaikan dengan komputer dan disimpan dalam bentuk file. Editor text yang digunakan seperti halnya program pengolah kata. Melalui editor ini, statement ditulis oleh seorang programmer yang disebut dengan kode sumber (source code) dan file yang disimpan disebut dengan file sumber (source file). Setelah kode sumber disimpan dalam bentuk file maka proses translasi (terjemahan) menuju bahasa mesin dapat dimulai. Selama fase ini diproses, sebuah program yang dinamakan dengan preprosesor membaca kode sumber.

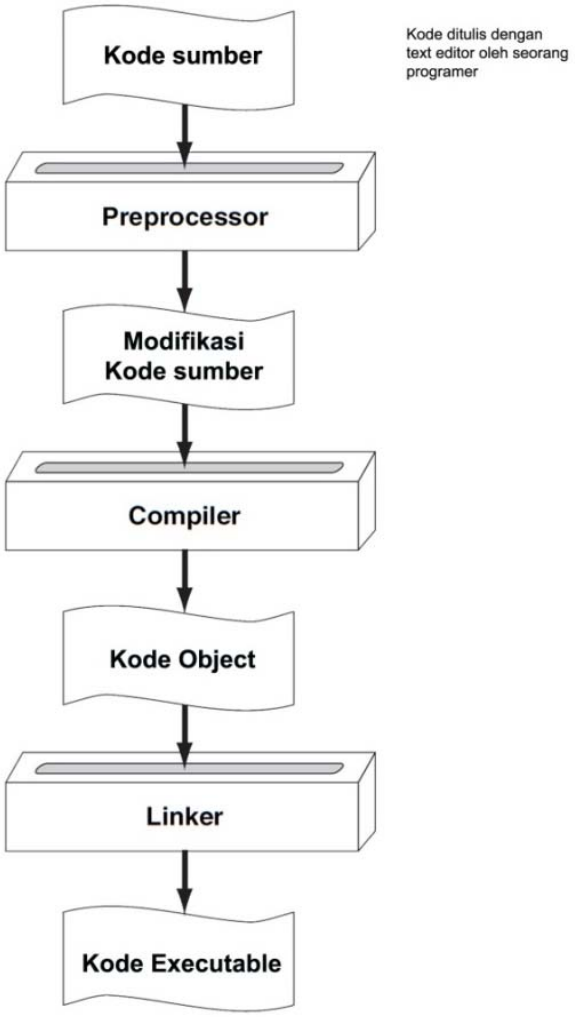
Preprosesor mencari baris khusus yang terdapat simbol. Baris ini terdiri dari instruksi yang menyebabkan preprosesor mengubah kode sumber dalam beberapa langkah atau perubahan yang dikehendaki oleh programmer. Selama terjadi phase lanjut, compiler melangkah melalui preproses kode sumber. Translasi tiap instruksi kode sumber menjadi instruksi bahasa mesin yang sesuai. Proses ini akan dibuka oleh beberapa kesalahan sintak yang mungkin terjadi pada program. Kesalahan sintak adalah penggunaan kata yag salah atau tidak sesuai, atau penggunaan operator, tanda baca, atau elemen bahasa yang lainnya. Jika program telah terbebas dari kesalahan sintak yang terjadi, maka compiler akan menyimpan instruksi bahasa mesin yang disebut dengan kode objeck (objeck code) dalam bentuk *object file*

Meskipun sebuah objeck file terdiri dari instruksi bahasa mesin, tetapi hal tersebut bukan sebuah program yang lengkap. C++ telah menyediakan berbagai kelengkapan dengan library yang kodenya telah ada dan biasanya atau kadang-kadang digunakan untuk mengerjakan tugas-tugas yang sulit. Sebagai contoh saja library digunakan untuk menangani kode perangkat keras khusus seperti menampilkan pada layar, menangani masukan dari keyboard. Selain itu juga menyediakan routin fungsi matematika seperti operasi kwadrat, perakaran bilangan. Kumpulan kode ini disebut dengan *run-time library*.

Hampir semua program menggunakan bagian tersebut. Ketika compiler membangkitkan file object, meskipun hal tersebut tidak termasuk dalam code mesin untuk setiap routin *run-time library* yang mungkin digunakan oleh seorang programmer. Selama phase akhir dari proses translasi, program lain yang disebut dengan linker mengkombinasikan file object dengan routin-routin library yang penting. Lingker akan menghentian langkah setelah file executable telah selesai dibuat. File executable terdiri dari instruksi bahasa mesin atau kode instruksi dan siap dijalankan pada sebuah komputer. Gambar dibawah

ini menunjukan proses translasi dari file sumber menjadi file executable.

Seluruh proses pada gambar tersebut dilakukan oleh preprosesor, compiler dan linker yang hal tersebut dilakukan dengan satu aksi tunggal.



Gambar 1.13. proses translasi dari file sumber menjadi file executable

1.8. Penulisan Bahasa pemrograman

Secara umum, bahasa pemrograman yang berbasiskan prosedur terdiri dari blok atau sub program. Yang memiliki dua bagian utama yaitu: Bagian deklarasi dan Bagian Statement.

1.8.1. Bagian Deklarasi

Bagian deklarasi merupakan bagian program untuk mendefinisikan tipe data suatu variable, konstanta, serta fungsi dan prosedur yang akan digunakan pada program. Selain itu, bagian deklarasi dapat juga digunakan untuk memberi nilai awal suatu variable. Dengan kata lain, deklarasi digunakan untuk memperkenalkan suatu nama kepada Compiler program. Berikut contoh deklarasi:

a. Deklarasi Variable:

Perhatikan deklarasi antara dua buah program yang berbeda. Dibawah ini merupakan deklarasi program yang digunakan oleh bahasa pascal dan bahasa C. perhatikan perbedaan antara dua bahasa tersebut.

Bahasa Pascal

```
Var     I,i2      : Integer;
        s         : String;
```

Bahasa C

```
Int     1,12;
Char    s[100];
```

Untuk mendeklarasikan variable pada Pascal, digunakan reserved word **var**, kemudian diikuti dengan nama variable (identifier) yang ingin digunakan, dan kemudian tipe data dari variable tersebut. Sedangkan pada C, deklarasi diawali dengan tipe data variable baru diikuti dengan nama variable (identifier). Suatu identifier harus diawali oleh karakter bukan angka, tetapi tidak boleh mengandung karakter khusus seperti * , - + / \ = < > . ? & dan sebagainya. Pada bahasa Pascal, identifier tidak bersifat case sensitive, maksudnya, huruf besar ataupun huruf kecil dianggap sama. Sebaliknya pada Bahasa C, identifier bersifat case sensitive, sehingga variable s dan S akan dianggap dua identifier yang berbeda.

b. Deklarasi Konstanta:

Dalam melakukan deklarasi nilai konstanta atau nilai tetap, dilakukan dengan cara menulis const. perhatikan contoh penulisan program dibawah ini:

```
const phi = 3.14;
```

Konstanta yaitu nilai yang tetap. Jadi jika mengacu pada contoh di atas, maka nilai phi tidak dapat diubah-ubah dan akan selalu 3.14

c. Deklarasi Tipe Data.

Perhatikan deklarasi antara kedua program antara C dan pascal seperti terlihat dibawah ini:

Bahasa C

```
struct datasiswa {
        char    nama[30];
        char    alamat[30];
        char    telp[20];
}

enum hari = (senin,selasa,rabu,kamis,jumat,Sabtu,mingu);
```

Bahasa Pascal

```
Type Tdatasiswa = ^dataSiswa
        Datasiswa    = record
        Nama         : String[30];
        Alamat       : String[30];
        Telp         : String[20];
end;

type hari = (senin,selasa,rabu,kamis,jumat,Sabtu,mingu);
```

a) Tipe Data sederhana

Tipe data sederhana merupakan tipe data yang paling kecil, yang hanya melibatkan satu item data, misalnya tipe data integer, string, real, Boolean, dan sebagainya. Kita dapat juga mendefinisikan sendiri tipe data ini. Tipe data yang didefinisikan sendiri tersebut diistilahkan enumerated data type (pada contoh adalah type hari).

b) Tipe Data terstruktur

Tipe data terstruktur merupakan tipe data yang terdiri dari beberapa item data. Bentuk dari tipe data ini dapat berupa array (terdiri dari item-item yang memiliki tipe data yang sama) ataupun record (terdiri dari item-item yang boleh memiliki tipe data yang berbeda). Pada contoh di atas, DataSiswa termasuk tipe data terstruktur.

c) Tipe Data Pointer

Tipe data pointer digunakan untuk menunjuk pada alamat memory suatu data yang lain. Jadi tipe data pointer pada dasarnya tidak menyimpan nilai data secara langsung, melainkan hanya menyimpan alamat dimana data berada. Untuk contoh pada bahasa Pascal, TDataSiswa merupakan tipe data pointer. Pada Bahasa C, untuk mendeklarasikan pointer untuk tipe data DataSiswa pada variable yang bernama TDataSiswa, dapat dituliskan sebagai berikut:

```
DataSiswa *TDataSiswa;
```

d. Deklarasi Procedure/Function:

Untuk membahas masalah procedure pada bahasa Pascal dan funsi pada C Perhatikan potongan program dibawah ini:

Pada bahasa pascal

```
procedure cetak (kal: string);
function Tambah (a,b: integer): Integer;
```

Pada bahasa C

```
void cetak (char 8string);
int Tambah (int a, int b);
```

Jika melihat pada contoh deklarasi pada bahasa C, mungkin timbul pertanyaan apa beda prosedur dengan fungsi? Pada Bahasa C, semua sub program dianggap fungsi, berbeda dengan Pascal yang menyertakan reserved word **procedure** dan **function** untuk membedakan antara keduanya. Sebenarnya, perbedaan utama antara prosedur dan fungsi yaitu: prosedur adalah fungsi yang tidak mengembalikan suatu nilai. Sebaliknya fungsi adalah suatu prosedur yang mengembalikan nilai.

Apabila mengacu pada contoh di atas, maka fungsi tambah akan mengembalikan suatu nilai yang bertipe integer, sedangkan prosedur Cetak tidak mengembalikan nilai apa-apa. Pada Bahasa C, procedure pada dasarnya adalah function yang mengembalikan **void** alias tidak mengembalikan nilai apa-apa**.**

1.8.2. Statement

Bagian statement merupakan bagian program yang berisi perintah yang akan dieksekusi /dijalankan. Pada bahasa Pascal, bagian statement selalu diawali dengan reserved word **begin** dan **end**. Apabila blok statement adalah blok utama program, maka reserved word **end** harus diakhiri dengan tanda titik(.), sebaliknya jika blok statement bukan blok utama program maka reserved word **end** diakhiri dengan tanda titik koma (;). Sebaliknya pada bahasa C, dimulai dari deklarasi variable hingga akhir statement diawali dan diakhiri dengan tanda kurung kurawal { dan }. Berikut adalah contoh potongan kode untuk implementasi menghitung luas lingkaran dengan Bahasa.

Program 1.2

```
#include <stdio.h>

void main()
{
      const phi = 3.24;
      float diameter,radius,luas;
      scanf ("%f", &diameter);
      radius = diameter/2.0;
      luas = phi * radius * radius;
      printf ("%f",Luas);
}
```

Berikut adalah penjelasan baris demi baris dari potongan kode Bahasa C untuk contoh di atas.

1. #include <stdio.h>
   Baris di awal program ini meng*include*kan header library stdio ke dalam program. Seperti halnya Pascal, Bahasa C juga memiliki cukup banyak library standar yang dapat digunakan.
2. void main()
   Baris kedua ini menandakan awal dari blok statement utama. Pada bahasa C, blok program utama merupakan suatu fungsi/sub program yang diberi nama 'main'.
3. { const phi = 3.14;
   Pada awal baris ketiga ini, terdapat tanda kurung kurawal sebagai pembuka blok statement. Kemudian reserved word **const** digunakan untuk mendeklarasikan konstanta phi.
4. float diameter, radius, Luas;
   Baris keempat ini digunakan untuk mendeklarasikan variable diameter, radius, dan luas dengan tipe data float (bilangan pecahan)
5. scanf("%f", &diameter);
   Baris kelima berisi perintah yang berfungsi untuk meminta input bertipe float dari user, dan kemudian nilainya disimpan ke variable diameter.
6. radius = diameter / 2.0;
7. Luas = phi * radius * radius;
   Baris keenam dan ketujuh melakukan operasi matematika untuk menghitung luas lingkaran.
8. printf("%f",Luas);
   Baris ini digunakan untuk mencetak isi variable luas yang bertipe float.
9. }
   Baris ini menandakan akhir dari blok statement.

1.9. Element Bahasa Pemrograman

Ketika kita mempelajari suatu bahasa pemrograman, kita akan menjumpai element-element yang pada dasarnya serupa antara satu bahasa dengan bahasa yang lain. Hal itu dikarenakan element-element tersebut merupakan bagian dari tata bahasa pemrograman yang bersangkutan. Berikut adalah element-element pada bahasa pemrograman: Aturan Leksikal, Tipe data, Expression, Statement, serta Function dan Procedure.

Aturan leksikal yaitu aturan yang digunakan dalam membentuk suatu deklarasi, definisi, maupun statement hingga menjadi satu program yang utuh. Aturan ini meliputi beberapa element antara lain:

a. Token
b. Komentar
c. Identifier
d. Keywords (Reserved Words)
e. Operator

Dibawah ini akan dibahas satu per satu element-element tersebut di atas.

a. Token

Token yaitu element terkecil pada bahasa pemrograman yang memiliki arti penting bagi compiler. Yang termasuk token antara lain: identifier, keywords(reserved words), operator, dan sebagainya. Token yang satu dengan yang lain

dipisahkan dengan satu atau lebih spasi, tab, baris baru, atau komentar.

b. Komentar

Komentar yaitu teks (kumpulan karakter) yang diabaikan oleh Compiler. Komentar sangat berguna untuk memberi catatan mengenai bagian program tertentu sebagai referensi baik bagi programmer itu sendiri maupun orang lain yang membaca kode program tersebut.

Pada bahasa Pascal, teks yang berada di antara kurung kurawal pembuka {dan kurung kurawal tutup } akan dianggap sebagai komentar. Selain itu, dapat pula menggunakan tanda (* sebagai pembuka komentar, dan tanda *) sebagai penutup. Perhatikan contoh program dibawah:

```
begin
{ Cetak hello World Oleh Saya}
        Writeln ('Hello World');
end.
```

Pada bahasa C, teks yang berada di antara tanda /* dan tanda */ akan dianggap sebagai komentar. Dan untuk teks yang ada setelah tanda // juga akan dianggap komentar satu baris. Berikut adalah contoh penggunaan komentar pada bahasa C:

```
void main() {
// Cetak hello World
// Oleh Saya
printf("Hello World");
}
```

c. Identifier

Identifier merupakan kumpulan karakter yang digunakan sebagai penanda untuk nama variable, nama tipe data, fungsi, prosedur, dan sebagainya. Aturan penulisan identifier pada bahasa Pascal dan bahasa C dapat dikatakan serupa. Yaitu: suatu identifier harus diawali oleh karakter non angka sebagai berikut:

```
_a b c d e f g h i j k l m n o p q r s t u v w x y z
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
```

Selanjutnya boleh menggunakan karakter angka (0 1 2 3 4 5 6 7 8 9) maupun karakter non angka tersebut di atas, namun tidak boleh menggunakan karakter khusus seperti + - * / ? ! { } [ ] dan sebagainya. Berikut adalah contoh-contoh identifier yang benar maupun salah.

Tabel 1.4. Identifier benar maupun salah

_nama	Benar
no_Telpon	Benar
bilangan2	Benar
4data	Salah, karena diawali oleh karakter angka: 4data
teks?	Salah, karena mengandung karakter khusus/special: Teks?

Catatan yang perlu diingat, identifier pada bahasa Pascal bersifat case insensitive (huruf besar dan huruf kecil dianggap sama), sedangkan pada bahasa C, identifier bersifat case sensitive (huruf besar dan huruf kecil dibedakan). Sebagai contoh, identifier No_Telpon dan no_telpon pada bahasa Pascal dianggap sama, sedangkan pada bahasa C, dianggap sebagai dua identifier yang berbeda.

d. Keywords (Reserved Words)

Keywords atau Reserved words merupakan kata-kata yang telah ada/didefinisikan oleh bahasa pemrograman yang bersangkutan. Kata-kata tersebut telah memiliki definisi yang sudah tetap dan tidak dapat diubah. Karena telah memiliki definisi tertentu, maka kata-kata ini tidak dapat digunakan sebagai identifier.Pada bahasa Pascal, yang termasuk reserved words antara lain:

and	array	asm	begin	case	const	div
do	downto	else	end	file	for	forward
function	goto	if	In	label	mod	nil
not	of	or	packed	procedure	program	record
repeat	set	string	then	to	type	unit
until	uses	var	while	with		

Pada bahasa C, yang termasuk reserved words antara lain:

break	case	char	const	continue	default	do
double	else	enum	float	for	goto	if
inline	int	long	return	short	signed	sizeof
static	struct	switch	type	def	union	void
unsigned	while					

1.10. Bahasa C++

Bahasa C++ diciptakan oleh Bjarne Stroustrup di AT&T Bell Laboratories awal tahun 1980-an berdasarkan C ANSI (American National Standard Institute). Pertama kali, prototype C++ muncul sebagai C yang dipercanggih dengan fasilitas kelas. Bahasa tersebut disebut C dengan kelas (C wih class). Selama tahun 1983-1984, C dengan kelas disempurnakan dengan menambah fasilitas pembeban lebihan operator dan fungsi yang kemudian melahirkan apa yang disebut C++.

Symbol ++ merupakan operator C untuk operasi penaikan, muncul untuk menunjukkan bahwa bahasa baru ini merupakan versi yang lebih canggih dari C.

Borland International merilis compiler Borland C++ dan Turbo C++. Kedua compiler tersebut sama-sama dapat digunakan untuk mlakukan kompilasi kode C++.

Bedanya, Borland C++ selain dapat digunakan dibawah lingkungan DOS, juga dapat digunakan untuk pemrograman Windows. Selain Borland International, beberapa perusahaan lain juga merilis compiler C++, seperti Topspeed C++ dan Zortech C++. Perhatikan dan bandingkan program dibawah ini:

Program 1.3. Contoh program dalam bahasa C

```
#include <stdio.h>

int main()
{
   double jam, besar, bayar;
   printf ( "Berapa Jam Kamu bekerja? ");
   scanf ( jam);
   printf ( "Berapa besar gaji tiap jamnya? ");
   scantf( besar);
   bayar  = jam * besar;
   printf ("Kamu mendapatkan Gaji (Rp)", bayar << endl;
   return 0;
}
```

Program 1.4. Contoh program dalam bahasa C++

```
#include <iostream>

int main()
{
   double jam, besar, bayar;
   cout << "Berapa Jam Kamu bekerja? ";
   cin >> jam;
   cout << "Berapa besar gaji tiap jamnya? ";
   cin >> besar;
   bayar  = jam * besar;
   cout << "Kamu mendapatkan Gaji (Rp)" << bayar << endl;
   return 0;
}
```

C++ diciptakan untuk mendukung pemrograman berorientasi pada objek (Object Oriented Pragramming/OOP) yang tidak dimiliki C. sementara C merupakan bahasa pemrograman

terbaik dilingkungannya, bahasa ini tidak memiliki kemampuan OOP. Reputasi C tidak diragukan lagi dalam menghasilkan program .EXE berukuran kecil, eksekusi yang cepat, antarmuka (interfacing) yang sederhana dengan bahasa lain dan fleksibilitas pemrograman. Apa yang membuat C tampak sukar dipelajari mungkin karena tiadanya pemeriksaan tipe. Sebagai contoh, dapat mencampur bilangan bulat dengan string untuk menghasilkan karakter. Namun, justru dsitu letak fleksibilitas C, dapat mengolah data C sebebas mengolah data dalam bahasa assembly.

Dibandingkan compiler C++ yang lain, Borland C++ memiliki keunggulan terutama dalam hal kecepatan dan efisiensi kompilasi. Disamping itu, Borland C++ mendukung beberapa sistem operasi yaitu DOS, Windows 16bit (Window 3.0) dan windows 32 bit (Windows NT). Meskipun demikian compiler Borland C++ juga memiliki kelemahan bila dibandingkan compiler C++ yang lain, misalnya : pemrograman dengan Borland C++ terutama yang menyangkut tampilan jauh lebih sulit daripada pemrograman dengan Microsoft Visual C++.

1.11. Struktur Bahasa C++

Program C maupun C++ selalu tersusun dari 4 (empat) bagian utama, yaitu : bagian komentar yang ditandai dengan simbol // dan pasangan /* … */, bagian pengarah compiler yang ditandai dengan symbol #, bagian deklarasi dan bagian definisi

1.11.1. Bagian Komentar

Program yang baik pada umumnya diberi komentar yang akan membantu orang lain maupun pembuat program itu untuk memahami program yang dibuat. Dalam C atau C++ setiap tulisan yang diapit oleh symbol /* … */ atau setiap baris yang dimulai dengan symbol // dianggap komentar.

Bahasa C++ tidak mengizinkan komentar bersarang (nested comment), namun Borland C++ lebih fleksibel dalam hal ini.

Program 1.5

```
#include <iostream>

int main()
{
   double jam, besar, bayar;                    // tipe data double
   cout << "Berapa Jam Kamu bekerja? ";         // tampilkan kata tersebut
   cin >> jam;                                  // masukan jam
   cout << "Berapa besar gaji tiap jamnya? ";   // tampilkan kata tersebut
   cin >> besar;                                // masukan besar gaji tiap jam
   bayar  = jam * besar;                        // kalikan jam dengan gaji tiap jam
   cout << "Kamu mendapatkan Gaji (Rp)" << bayar << endl; // jumlah gaji total
   return 0;
```

```
}
```

Pada Borland C++ dapat menggunakan komentar bersarang asalkan opsi cek Nested comments pada menu Options/Compiler/Source dipilih.

1.11.2. Bagian Pengarah Kompiler

Supaya lebih jelas mengenai bahasa C++ perhatikan program dibawah ini:

Program 1.6

```
# include <iostream.h>

void main ( )
{
      char pesan [ ] = "Hello, C++ programmers!" ;
      cout << pesan ;
      return 0 ;
}
```

Merupakan statement pre-prosesor, disebut juga pengarah compiler karena berfungsi mengatur proses kompilasi. iostream.h merupakan file program yang mengandung deklarasi kelaskelas yang diperlukan oleh objek cout. File-file dengan ekstensi .h yang berisi deklarasi fungsi-fungsi standar C ini, disebut secara umum sebagai file header.

Beberapa pengarah compiler antara lain: # define, # include, # if, # else, # elif, # endif, # ifdef, # ifndef

a. Pengarah kompiler # define

Untuk mendefinisikan suatu pengenal / konstanta yang nantinya akan digantikan oleh praprosesor saat program dikompilasi. Perhatikan contoh Program dibawah ini:

```
# define SIZE 30

int array [SIZE] ;
for (register int i = 0 ; i < SIZE ; i++)
{
          cout << array [ i ] ;
}
```

b. Pengarah Kompiler # Include

Berfungsi membaca file program tertentu dan mengikutsertakan file tersebut dalam proses kompilasi. Nama file yang dimaksud harus diapit symbol ‘ < ‘ dan ‘ > ‘ atau tanda kutip dua (“ … “).

c. Pengarah Kompiler # If, # Else, # Elif, # Endif

Digunakan untuk memilih bagian program yang akan dikompilasi. Kompilasi cari ini disebut kompilasi bersyarat dan program yang baik biasanya memanfaatkan teknik ini.

d. Pengarah Kompiler # Ifdef, # Ifndef

Digunakan juga dalam kompilasi bersyarat. # **Ifdef** dapat dibaca: 'jika didefinisikan' dan # **ifndef** dapat dibaca: 'jika tidak didefinisikan'. Pengarah compiler ini sering digunakan untuk menandai bahwa suatu file sudah diikutsertakan dalam kompilasi.

1.11.3. Deklarasi Dan Definisi

Semua program C pada dasarnya tersusun dari rangkaian pemanggilan fungsi yang bekerja atas sekelompok data. Selain pemanggilan fungsi, program C mengandung komponen lain yang disebut statement.

Statement C ada dua, yaitu : statement yang tidak dapat dieksekusi / non executable (bila dikompilasi tidak menghasilkan kode objek dan biasanya digunakan untuk mengatur alur program), dan statement yang dapat dieksekusi / executable (bila dikompilasi akan menghasilkan kode objek).

Setiap pemanggilan fungsi maupun statement executable dalam C harus diakhiri dengan tanda titik koma (;). Perhatikan Contoh program C++:

Program 1.7

```
#include <conio.h>
#include <iostream>

using namespace std;

int main()
{
      char pesan [ ] = "Hello, Programmer C++ !" ;
      cout << pesan;
      getch();
      return 0 ;
}
```

Keluaran programnya adalah:

```
Hello, Programmer C++ !
```

Dalam contoh program C++ diatas, **return** merupakan contoh statement executable yang menginstruksikan agar suatu fungsi

mengembalikan nilai balik tertentu. Contoh statement non executable adalah: If, else, dan while. Main () merupakan contoh fungsi, sedangkan pesan adalah contoh data. Baik data maupun fungsi harus dideklarasikan. Data perlu dideklarasikan agar compiler tahu berapa byte memori yang harus disediakan untuk data yang bersangkutan, sedangkan fungsi perlu dideklarasikan agar compiler dapat memeriksa ketepatan pemanggilan fungsi yang bersangkutan. Deklarasi fungsi sering disebut pula prototype fungsi.

1.12. Input Dan Output

Di ANSI C, operasi input dan output dilakukan dengan menggunakan fungsi-fungsi yang ada di header file stdio.h. contohnya untuk input dan output ke layer monitor digunakan perintah seperti printf, scanf, putch, dan sebagainya. Instruksi input dan output ke file digunakan perintah seperti fread, fwrite, fputc, dan sebagainya.

Bahasa C++ mempunyai teknik input dan output yang baru, yaitu: menggunakan stream. Header file untuk input dan output stream adalah iostream.h dan beberapa file lain, Bentuk umum operator output adalah sebagai berikut:

```
cout << ekspresi ;
```

Bentuk umum operator Input adalah sebagai berikut:

```
cin >> variable ;
```

seperti strstrea.h, fstream.h, dan constrea.h.

Stream adalah suatu logika device (peralatan logika) yang menghasilkan dan menerima informasi atau sebagai tempat yang digunakan untuk menampung keluaran dan menampung aliran data. Stream adalah nama umum untuk menampung aliran data (contoh: file, keyboard, mouse), maupun untuk keluaran (contoh: layer, printer). Dalam bahasa C++, input berarti membaca dari stream dan output berarti menulis ke stream.

Dalam C++, menggunakan escape sequences untuk merepresentasikan suatu karakter yang tidak terdapat dalam tradisional simbol. Beberapa diantaranya :

```
\ n : linefeed / baris baru
\ b : back space
\ “ : petik ganda
```

Program 1.8. contoh program versi ANSI C :

```
# include <stdio.h>

void main ( )
{
        int x ;
        printf ( "Masukkan sebuah bilangan : \n" ) ;
        scanf ( " %d ", &x ) ;
        printf ( "Bilangan yang dimasukkan adalah %d\n ", x ) ;
```

```
}
```

Program 1.9 contoh program versi C++ :

```
# include <iostream.h>

void main ( )
{
        int x ;
        cout << "Masukkan sebuah bilangan : " << endl ;
        cin >> x ;
        cout << "Bilangan yang dimasukkan adalah " << x << endl ;
}
```

Program 1.10 contoh program input dan output pada C++:

```
# include <iostream.h>

void main ( )
{
        int a ;
        cout << "masukkan suatu bilangan :" ;
        cin >> a ;
        cout << "nilai tersebut ditambah 1 = ' << a+1 ;
        return 0 ;
}
```

1.15. Soal Latihan

Jawablah soal latihan dibawah ini dengan baik dan benar.

1. Sebutkan bagian-bagian perangkat keras komputer
2. Sebutkan lapisan perangkat lunak komputer personal
3. Sebutkan berbagai macam memori yang digunakan pada komputer
4. Apa yang dimaksud dengan piranti Input dan output (I/O).
5. Jelaskan sistem bilangan yang digunakan oleh komputer
6. Jelaskan proses translasi dari file sumber menjadi file executable
7. Jelaskan struktur bahasa C++
8. Sebutkan perbedan bahasa C dan C++

BAB 2
BAHASA DAN ALGORITMA PEMROGRAMAN

2.1. Bahasa pemrograman

Bahasa atau dalam bahasa inggris *language* adalah suatu sistim untuk berkomunikasi. Bahasa tertulis menggunakan simbol (yaitu huruf) untuk membentuk kata. Dalam ilmu komputer,bahasa manusia disebut bahasa alamiah (natural languages), dimana komputer tidak bisa memahaminya, sehingga diperlukan suatu bahasa komputer. Komputer mengerjakan transformasi data berdasarkan kumpulan perintah - program - yang telah dibuat oleh pemrogram. Kumpulan perintah ini harus dimengerti oleh komputer, berstruktur tertentu (syntax) dan bermakna.

Bahasa pemrograman merupakan notasi untuk memberikan secara tepat program komputer. Berbeda dengan bahasa alamiah, mis. Bahasa Indonesia, Inggris dsb. yang merupakan bahasa alamiah(natural language), sintaks dan semantik bahasa pemrograman (komputer) ditentukan secara kaku, sehingga bahasa pemrograman juga disebut sebagai bahasa formal (formal language). Jadi, dalam bahasa pemrograman yang digunakan sebagai alat komunikasi untuk memberikan perintah kepada komputer tidak berlaku kebebasan berekspresi seperti laiknya dalam bahasa alamiah.

Pemrograman dalam pengertian luas meliputi seluruh kegiatan yang tercakup dalam pembuatan program, termasuk analisis kebutuhan (requirement's analysis) dan keseluruhan tahapan dalam perencanaan (planning) ,

perancangan (design) dan pewujudannya (implementation). Dalam pengertian yang lebih sempit, pemrograman merupakan pengkodean (coding atau program writing = penulisan program) dan pengujiannya (testing) berdasarkan rancangan tertentu.

Pemahaman yang lebih sempit ini sering digunakan dalam pembuatan program-program terapan komersial yang membedakan antara system analyst yang bertanggung jawab dalam menganalisa kebutuhan, perencanaan dan perancangan program dengan pemrogram (programmer) yang bertugas membuat kode program dan menguji kebenaran program. Generasi bahasa pemrograman:

- Generasi I: machine language
- Generasi II: assembly language : Asssembler
- Generasi III: high-level programming language: C, PASCAL, dan sebagainya.
- Generasi IV: 4 GL (fourth-generation language): SQL

2.1.1. Bahasa Tingkat Rendah

Merupakan bahasa assembly atau bahasa mesin. Bahasa ini lebih dekat ke mesin (*hardware*), dimana pada high-level programming languages, suatu bahasa yang sudah dekat pada bahasa manusia. Pada bahasa ini akan lebih cepat dipahami oleh programmer karena sifatnya yang dekat dengan bahasa manusia

2.1.2. Bahasa Mesin

Bahasa mesin merupakan representasi tertulis machine code (kode mesin), yaitu kode operasi suatu mesin tertentu. Bahasa ini bersifat khusus untuk mesin tertentu dan "dimengerti" langsung oleh mesin, sehingga pelaksanaan proses sangat cepat. Bahasa mesin kelompok komputer tertentu berlainan dengan bahasa mesin kelompok komputer yang lain.

Abstraksi bahasa ini adalah kumpulan kombinasi kode biner "0" dan "1" yang sangat tidak alamiah bagi kebanyakan orang - kecuali insinyur pembuat mesin komputer. Karena tidak alamiah bagi kebanyakan orang, bahasa mesin juga disebut bahasa tingkat rendah.

2.1.3. Bahasa Assembly

Bahasa rakitan (assembly language) merupakan notasi untuk menyajikan bahasa mesin yang lebih mudah dibaca dan dipahami oleh manusia. Bahasa ini sudah menggunakan simbol alpabet yang bermakna (mnemonic). Contoh “MOV AX 1111”, pindahkan ke register AX nilai 1111.Proses data oleh komputer berdasarkan perintah bahasa rakitan adalah cepat. Meski demikian masih merepotkan-bahkan bagi kebanyakan pemrogram, karena masih harus mengingat-ingat tempat penyimpanan data. Bahasa rakitan juga bersifat khusus untuk mesin tertentu. Contoh: Assembler

2.1.4. Bahasa Tingkat Tinggi

Adalah bahasa pemrograman yang dekat dengan bahasa manusia, kelebihan utama dari bahasa ini adalah mudah untuk di baca, tulis, maupun diperbaharui, sebelum bisa

dijalankan program harus terlebih dahulu di-compile. Contoh Ada, Algol, BASIC, COBOL, C, C++, FORTRAN, LISP, dan Pascal, dsb. Pada generasi bahasa pemrograman terakhir sekarang ini, kedua cara interpretasi dan kompilasi digabungkan dalam satu lingkungan pengembangan terpadu (IDE=integrated development environ-ment).

Cara interpretasi memudahkan dalam pembuatan program secara interaktif dan cara kompilasi menjadikan eksekusi program lebih cepat. Pertama program dikembangkan interaktif, kemudian setelah tidak ada kesalahan keseluruhan program dikompilasi. Contoh bahasa program seperti ini adalah Visual BASIC yang berbasis BASIC dan Delphi yang berbasis PASCAL. Bahasa tingkat tinggi bersifat portable. Program yang dibuat menggunakan bahasa tingkat tinggi pada suatu mesin komputer bersistem operasi tertentu, hampir 100% bisa digunakan pada berbagai mesin dengan aneka sistem operasi. Kalaupun ada perbaikan sifatnya kecil sekali.

2.1.5. Bahasa Generasi 3 & 4

Lebih dekat ke bahasa manusia dibandingkan dengan high-level programming languages. Biasanya dipakai untuk mengakses database. Contoh perintah pada bahasa SQL: FIND ALL RECORDS WHERE NAME IS "JOHN".

2.1.6. Bahasa Pemrograman untuk tujuan tertentu

Dibawah ini merupakan Tabel Bahasa Pemrograman untuk tujuan tertentu. (Mc. Connell, h 46)

Tabel 2.1. Bahasa Pemrograman

JENIS PROGRAM	BAHASA TERBAIK	BAHASA TERBURUK
Data terstruktur	ADA, C /C++, PASCAL	Assembler, BASIC
Proyek cepat	BASIC	PASCAL, ADA, Assembler
Eksekusi cepat	Assembler, C	BASIC,Intrepreter Language
Kalkulasi matematika	FORTRAN	PASCAL
Menggunakan memori dinamis	PASCAL, C	BASIC
Lingkungan memori terbatas	BASIC, Assembler, C	FORTRAN
Program real-time	ADA, Assembler, C	BASIC, FORTRAN
Manipulasi string	BASIC, PASCAL	C
Program mudah dikelola	PASCAL, ADA	C, FORTRAN

Tabel Rasio pernyataan bahasa tingkat tinggi dengan kode bahasa rakitan yang setara. (Mc. Connell, h 46)

Tabel 2.2. Rasio bahasa tingkat tinggi dengan kode bahasa rakitan

BAHASA	RASIO
Assembler	1: 1
ADA	1 : 4.5
Quick / Turbo / Basic	1 : 5
C	1 : 2.5
FORTRAN	1 : 3
PASCAL	1 : 3.5

2.2. Compiler dan Intepreter

Compiler Adalah suatu program yang menterjemahkan bahasa program (source code) ke dalam bahasa objek (object code). Compiler menggabungkan keseluruhan bahasa program dikumpulkan kemudian disusun kembali.

2.2.1. Tahapan Kompilasi:

Dalam melakukan penulisan bahasa pemrograman komputer ada beberapa tahapan-tahapan yang harus dilakukan antara lain:
Pertama source code (program yang ditulis) dibaca ke memory komputer.

- Source code tersebut diubah menjadi object code (bahasa assembly)
- Object code dihubungkan dengan library yang dibutuhkan untuk membentuk file yang bisa di eksekusi.

Compiler memerlukan waktu untuk membuat suatu program yang dapat dieksekusi oleh komputer. Tetapi, program yang diproduksi oleh Compiler bisa berjalan lebih cepat dibandingkan dengan yang diproduksi oleh Interpreter, dan bersifat independen.

2.2.2.Interpreter

Berbeda dengan compiler, Interpreter menganalisis dan mengeksekusi setiap baris dari program tanpa melihat program secara keseluruhan. Keuntungan dari Interpreter adalah dalam eksekusi yang bisa dilakukan dengan segera. Tanpa melalui tahap kompilasi, untuk alasan ini interpreter digunakan pada saat pembuatan program berskala besar.

2.3. Tipe Pemrograman

Dalam Bahasa pemrograman komputer terdapat tipe pemrograman. Tipe pemrograman yang pertama adalah pemrograman terstruktur dan prosedural.

2.3.1. Pemrograman Terstruktur

Pemrograman terstruktur adalah cara pemrosesan data yang terstuktur. Terstruktur dalam: analisa, cara dan penulisan program. Prinsip pemrograman terstruktur adalah mempunyai ciri sebagai berikut:

- Menggunakan rancangan pendekatan dari atas ke bawah (top down design),

- Membagi program ke dalam modul-modul logika yang sejenis,
- Menggunakan sub-program untuk proses-proses sejenis yang sering digunakan,
- Menggunakan pengkodean terstruktur: IF ..... THEN, DO ..... WHILE dan lain-lainnya,
- Menghindari penggunaan perintah GO TO bila tidak diperlukan,
- Gunakan nama-nama bermakna (mnemonic names), dan
- Membuat dokumentasi yang akurat dan berarti.

Dalam perencanaan dan perancangan dari atas ke bawah, kategori dan penyelesaian masalah dimulai dari bagian yang utama kemudian dibagi menjadi bagian yang lebih kecil. Rancangan cara ini memudahkan penulisan, pengujian, koreksi dan dokumentasi program. Tahapan rancangan atas ke bawah dalam pemrograman:

- Tentukan keluaran (output) yang diminta, masukan (input) yang diperlukan dan proses-proses utama yang diperlukan untuk transformasi data.
- Membagi proses utama ke dalam modul-modul fungsional.
- Buat algoritma msing-masing modul, dari modul utama ke sub-sub modul.

Setiap modul dalam proses rancangan atas ke bawah biasanya dibatasi dalam isi maupun batasan-batasan berikut:

- Setiap modul hanya mempunyai satu masukan dan keluaran
- Setiap modul hanya mewakili satu fungsi program.

Rancangan (design) terstruktur:

- Membagi program menjadi subprogram
- Menekankan fungsionalitas.
- Cocok untuk sistem yang banyak mempunyai fungsi independen. Gaya penulisan program terstruktur: Menggunakan indentasi sehingga jelas struktur dan kontrol program. Memudahkan pembacaan, pemahaman, penelusuran kesalahan dan pembuatan koreksi.

2.3.2. Pemrograman procedural dan terstruktur

Bahasa pemrograman prosedural adalah bahasa pemrograman yang mendukung pembuatan program sebagai kumpulan prosedur. Prosedur-prosedur ini dapat saling memanggil dan dipanggil dari manapun dalam program dan dapat menggunakan parameter yang berbeda-beda untuk setiap pemanggilan. Prosedur adalah bagian dari program untuk melakukan operasi-operasi yang sudah ditentukan dengan menggunakan parameter tertentu. Bahasa pemrograman terstruktur adalah pemrograman yang mendukung abstraksi data, pengkodean terstruktur dan kontrol program terstruktur. Kontrol program terstruktur adalah sebagai berikut:

1. Runtun - urut (sequence)
2. Pilihan (selection)
3. Pengulangan (repetition - loop)
4. Batasan Masalah Merencanakan sistim dan spesifikasi program: Siapa yang akan menggunakan program dan untuk apa? dengan cara:
   - Menentukan tujuan dan hasil yang akan dicapai

- Menentukan hal-hal yang diperlukan oleh sistim
- Pengumpulan data

5. Pengembangan Model Pembuatan model dari sistim yang akan kita bangun, model adalah suatu gambaran sederhana dari sistim yang kita buat. Dengan pembuatan model akan terlihat dengan jelas hubungan antara objek-objek dalam sistim yang akan kita bangun. Untuk penyelesaian aritmatik, biasanya model dibuat dalam bentuk rumus matematik. Contoh: untuk membuat program luas_lingkaran kita membuat model matematis c = axb
6. Rancangan algoritma Pembuatan urutan instruksi yang akan ditulis pada program (dijelaskan lebih lanjut)
7. Pemrograman Implementasi algoritma ke dalam program (algoritma sendiri dalam komputer adalah merupakan program).
8. Uji dan Validasi Pengujian terhadap program : seperti kesalahan penulisan (syntax error) , kesalahan saat eksekusi (runtime error) kesalahan logika program (program berjalan tapi menghasilkan output yang salah-fatal error).

Dokumentasi Pembuatan catatan pada program terutama pada modul-modul yang rumit.

2.4. Algoritma

Pemrograman komputer dan algoritma pemrograman adalah dua hal yang tidak dapat dipisahkan karena pembuatan program komputer akan lebih sulit dan lama tanpa mengetahui dengan pasti bagaimana algoritma penyelesaian masalahnya. Sebelum mengetahui lebih lanjut apa yang dimaksud dengan algoritma pemrograman, kita bahas dahulu apa yang dimaksud dengan pemrograman komputer atau program komputer.

Definisi program komputer adalah sederetan perintah-perintah (instruksi) yang harus dikerjakan oleh komputer untuk menyelesaikan masalah. Deretan perintah-perintah tersebut tidak bisa kita tulis secara sembarangan atau semau kita tetapi harus teratur agar komputer dapat memahami dan memprosesnya dengan baik sehingga permasalahan yang ada dapat diselesaikan dengan baik pula.

Untuk itulah diperlukan algoritma karena definisi dari algoritma itu sendiri menurut Microsoft Press Computer and Internet Dictionary (1998) adalah urutan langkah logis tertentu untuk memecahkan suatu masalah. Yang ditekankan adalah urutan langkah logis, yang berarti algoritma harus mengikuti suatu urutan tertentu, tidak boleh melompat-lompat dan disusun secara sistematis. Sedangkan yang dimaksud dengan langkah-langkah logis adalah kita harus dapat mengetahui dengan pasti setiap langkah yang kita buat.

Menurut Sjukani (2005), algoritma adalah alur pemikiran dalam menyelesaikan suatu pekerjaan yang dituangkan secara tertulis. Yang ditekankan pertama adalah alur pikiran, sehingga algoritma seseorang dapat berbeda dari algoritma orang lain. Sedangkan penekanan kedua adalah tertulis, yang artinya dapat berupa kalimat, gambar, atau tabel tertentu. Jadi

dapat disimpulkan bahwa algoritma lebih merupakan alur pemikiran untuk menyelesaikan suatu pekerjaan atau suatu masalah daripada pembuatan program komputer. Algoritma inilah yang kemudian dijadikan landasan (pedoman) untuk membuat program komputer.

Meskipun algoritma tidak dapat dipisahkan dengan pemrograman komputer tetapi jika anda beranggapan bahwa algoritma identik dengan pemrograman komputer, anda salah besar. Hal ini dikarenakan dalam kehidupan sehari-haripun seringkali kita berhadapan dengan masalah-masalah yang kalau kita cermati mengikuti kaidah-kaidah penyelesaian secara algoritma. Misalkan saja cara-cara memasak mie instan, membuat kopi atau teh, memasak makanan yang dinyatakan dalam bentuk resep, dan masih banyak lagi yang semuanya itu dapat kita sebut sebagai algoritma. Pada mie instan misalnya, biasanya pada bungkusnya terdapat urutan langkah-langkah bagaimana cara memasak atau menyajikannya. Bila langkah-langkah tersebut tidak logis, maka dapat dipastikan bahwa kita akan memperoleh hasil yang tidak sesuai dengan yang diharapkan. Kita harus membaca satu demi satu langkah-langkah pembuatannya kemudian mengikutinya agar memperoleh hasil yang baik.

Yang harus diingat disini adalah kita tidak harus mengikuti langkah-langkah yang sudah diberikan, tetapi kita dapat memodifikasinya atau bahkan membuat resep atau cara baru yang lebih baik tetapi menghasilkan hal yang sama (mempunyai tujuan yang sama), yaitu dapat menikmati hasil masakan. Demikian juga dengan pemrograman komputer, kita juga tidak harus mengikuti algoritma yang sudah ada, tetapi kita dapat menambah ataupun mengurangi bahkan membuat algoritma yang baru asalkan permasalahan yang ada dapat terpecahkan dengan baik.

2.5. Ciri Algoritma

Oleh karena algoritma digunakan untuk memecahkan suatu permasalahan maka algoritma tersebut harus menghasilkan suatu jawaban atas permasalahan tersebut. Dengan kata lain algoritma harus memiliki paling tidak satu keluaran. Sedangkan masukan dari algoritma dapat nol (tidak ada) atau banyak masukan (data). Yang dimakdud dengan nol masukan adalah jika algoritma itu hanya untuk menampilkan suatu informasi saja. Misalnya output "Hello World" yang sering kita temukan pada tutorial-tutorial saat kita baru belajar membuat program dari suatu bahasa pemrograman tertentu. Kedua hal diatas, memiliki paling sedikit satu keluaran dan dapat memiliki nol atau banyak masukan, merupakan dua dari beberapa ciri algoritma.

Tugas algoritma dikatakan selesai kalau algoritma tersebut sudah menghasilkan satu atau lebih jawaban atas permasalahan yang ada. Dengan demikian setelah mengerjakan langkah-langkah penyelesaian masalah, maka algoritma tersebut harus berhenti tidak melakukan proses apapun. Berhenti di sini artinya adalah jika diterjemahkan ke dalam bentuk program dan program dijalankan,

maka setelah menghasilkan suatu output, program dapat langsung berhenti atau menunggu instruksi lebih lanjut dari pengguna program seperti mengulang perhitungan lagi, keluar program (menghentikan program), dan lain sebagainya. Dengan demikian ciri ketiga dari algoritma adalah setelah selesai mengerjakan langkah-langkah penyelesaian masalah, algoritma harus berhenti.

Ciri keempat dari algoritma adalah setiap langkah yang dibuat harus dibuat sesederhana mungkin tetapi efektif agar dapat dipahami oleh pemroses (manusia maupun komputer) sehingga dapat diselesaikan dalam waktu yang singkat serta masuk akal. Sedangkan ciri yang terakhir adalah setiap langkah dalam algoritma harus didefinisikan dengan tepat dan jelas sehingga tidak berarti-dua (ambiguitas) sehingga menimbulkan kesalahan dalam penafsiran oleh pemroses

2.6. Penerapan Algoritma

Berikut adalah contoh pemecahan masalah (algoritma) yang diambil dari permasalahan yang mungkin sering kita hadapi dalam kehidupan kita sehari-hari. Diketahui dua buah ember A dan B dimana ember A berisi air dan ember B berisi minyak tanah. Jika diinginkan isi kedua ember itu saling ditukar sehingga ember A berisi minyak tanah dan ember B berisi air, bagaimanakah caranya? Apakah cukup dengan cara (membuat algoritma), tuangkan isi ember A ke ember B dan kemudian tuangkan isi ember B ke ember A? Apakah permasalahan dapat diselesaikan dengan cara (algoritma) tersebut? Jawabannya adalah tidak, karena algoritma tersebut tidak logis dan hasilnyapun tidak sesuai dengan yang diinginkan karena algoritma tersebut akan menghasilkan ember A akan berisi campuran air dan minyak tanah sedangkan ember B akan kosong.

Bagaimanakah cara (algoritma) yang benar dari permasalahan tersebut? Algoritma yang benar adalah pindahkan dahulu isi ember A ke ember lain (misal ember C), kemudian setelah ember A kosong pindah isi ember B ke ember A. Langkah terakhir adalah mengisi ember B dengan minyak tanah yang ada di ember C. Inilah algoritma yang paling logis dan menghasilkan jawaban yang benar atas permasalahan tersebut.

Seperti yang telah dikemukakan sebelumnya, setiap orang mempunyai cara pemecahan sendiri-sendiri sehingga setiap orang dimungkinkan mempunyai algoritma yang berbeda-beda untuk memecahkan suatu permasalahan yang sama. Contohnya adalah permasalahan di atas. Langkah pertama bisa saja yang dipindah bukan isi ember A ke ember C dulu, tetapi isi ember B yang dipindahkan ke dalam ember C terlebih dahulu. Setelah itu baru memindahkan isi ember A ke dalam ember B dan terakhir memindah isi ember C ke ember A. Algoritma ini sedikit berbeda tetapi tetap menghasilkan jawaban yang sama atas persoalan yang ada.

2.7. Notasi Algoritma

Algoritma mempunyai aturan penulisan sendiri yang disebut dengan notasi algoritma. Notasi algoritma ini tidak tergantung dari spesifikasi bahasa pemrograman tertentu dan komputer yang mengeksekusinya. Hal ini dikarenakan notasi algoritma bukanlah notasi bahasa pemrograman. Notasi algoritma merupakan bahasa universal yang dapat diterima oleh semua bahasa pemrograman yang ada. Oleh sebab itu algoritma yang baik harus dapat diterjemahkan ke dalam bentuk source code dari semua bahasa pemrograman yang ada.

Untuk membuat algoritma dari suatu permasalahan, biasanya digunakan salah satu dari tiga buah notasi algoritma yang dikenal, yaitu uraian kalimat deskriptif, flow chart, atau pseudo code. Sebagai contoh permasalahan, jika diinginkan sebuah program komputer yang dapat mengetahui bilangan terbesar dari tiga buah bilangan yang dimasukkan. Bagaimanakah algoritmanya?

2.8. Deskriptif Algoritma

Algoritma dengan uraian kalimat deskriptif adalah notasi algoritma yang paling sederhana karena algoritma ini menggunakan bahasa sehari-hari. Untuk permasalahan yang sederhana penggunaan notasi ini sangatlah mudah, akan tetapi untuk permasalahan yang lebih komplek dan rumit, penggunaan notasi ini akan lebih sulit dan sering kali terjadi ambigu dalam langlah-langkah penyelesaian masalah. Oleh karena itulah untuk kasus-kasus yang lebih komplek, penggunaan notasi ini jarang sekali bahkan tidak digunakan.

Permasalahan di atas, yaitu mencari bilangan terbesar dari tiga buah bilangan yang dimasukkan, tergolong permasalahan yang sederhana, jadi algoritmanya masih mudah dan dapat dijelaskan dengan uraian kalimat deskriptif sebagai berikut:

1. Masukkan sembarang bilangan sebanyak tiga buah.
2. Ambil bilangan pertama dan set maksimum-nya sama dengan bilangan pertama.
3. Ambil bilangan kedua dan bandingkan dengan maksimum.
4. Apabila bilangan kedua lebih besar dari maksimum maka ubah maksimum-nya menjadi sama dengan bilangan kedua.
5. Ambil bilangan ketiga dan bandingan dengan maksimum.
6. Apabila bilangan ketiga lebih besar dari maksimum maka ubah lagi maksimum-nya menjadi sama dengan bilangan ketiga.
7. Variabel maksimum akan berisi bilangan yang terbesar dan tampilkan hasilnya.

Algoritma dengan uraian kalimat deskriptif seperti di atas sudah jarang sekali kita temukan karena kadang kala agak sulit untuk memahaminya. Yang paling banyak kita temukan adalah algoritma (dengan uraian kalimat deskriptif) yang ditulis secara lebih sistematis dan efisien sehingga lebih mudah untuk memahaminya. Algoritma tersebut adalah sebagai berikut:

1. Masukkan a, b, dan c.
2. mak ← a.
3. Jika b > mak, kerjakan langkah ke-4. Jika tidak, kerjakan langkah ke-5.
4. mak ← b.
5. Jika c > mak, kerjakan langkah ke-6. Jika tidak, kerjakan langkah ke-7.
6. mak ← c.
7. Tulis mak.

Dalam notasi algoritma, baik menggunakan notasi algoritma dengan uraian kalimat deskriptif, flow chart maupun pseudo code, kita tidak menggunakan tanda = (sama dengan) tetapi menggunakan simbol anak panah ke arah kiri (←) seperti yang terlihat pada langkah ke-2, 4, dan 6. Sebagai contoh pada langkah ke-2, arti dari notasi tersebut adalah nilai variabel a (yang ada di sebelah kanan anak panah) diberikan kepada variabel mak (yang ada di sebelah kiri anak panah). Dengan demikian jika nilai variabel a adalah 10, maka nilai mak juga 10 atau dalam penulisan secara matematis mak = a.

Penggunaan anak panah ini dikarenakan, seperti yang telah dikemukakan sebelumnya, algoritma tidak diperuntukkan untuk suatu bahasa pemrograman tertentu, tetapi dapat diaplikasikan atau diterjemahkan ke dalam bentuk source code dari semua bahasa pemrograman yang ada. Dalam pascal misalnya, notasi yang digunakan untuk tanda = (sama dengan) adalah := (titik dua dilanjutkan dengan tanda sama dengan) sehingga langkah ke-2 akan diterjemahkan menjadi mak := a. Akan tetapi dalam bahasa C++ maupun Java, tanda = (sama dengan) masih tetap digunakan sehingga penerjemahannya adalah mak = a

2.9. Flow Chart

Notasi algoritma yang paling banyak digunakan adalah flow chart karena bentuknya yang sederhana dan mudah dipahami. Flow chart (diagram alir) adalah penggambaran secara grafik dari langkah-langkah pemecahan masalah yang harus diikuti oleh pemroses. Flow chart terdiri atas sekumpulan simbol, dimana masing-masing simbol menggambarkan suatu kegiatan tertentu. Flow chart diawali dengan penerimaan masukan (input), pemrosesan masukan, dan diakhiri dengan menampilkan hasilnya (output).

Adapun simbol-simbol yang sering digunakan untuk menyusun flow chart (dalam microsoft visio) adalah sebagai berikut :

1) Masukan
   Masukan merupakan kegiatan penerimaan data yang disimbolkan dengan jajaran genjang. Kita dapat menuliskan masukan yang diperlukan pada suatu waktu secara satu per satu maupun secara keseluruhan, akan tetapi untuk alasan efisiensi ruang gambar biasanya masukan

dituliskan bersamaan secara keseluruhan.

Gambar 2.1. Simbol masukan

2) Masukan manual
Untuk masukan secara manual yang dimasukkan melalui keyboard, atau perangkat input lainnya seperti barcode reader, kita dapat menggunakan simbol masukan secara manual. Sama dengan simbol masukan, pada simbol masukan manual ini untuk alasan efisiensi ruang gambar biasanya masukan juga dituliskan bersamaan secara keseluruhan.

Gambar 2.2. Simbol masukan manual

3) Proses
Data yang dimasukan kemudian diproses untuk menghasilkan jawaban atas persoalan yang ingin dipecahkan. Kegiatan memproses data ini disimbolkan dengan persegi panjang. Sama seperti simbol pada masukan, penulisan operasi-operasi pada data dapat dilakukan secara satu per satu maupun secara keseluruhan.

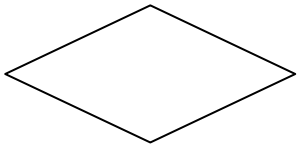
Gambar 2.3. Simbol proses

4) Keluaran
Keluaran adalah hasil dari pemrosesan data dan merupakan jawaban atas permasalahan yang ada. Keluaran ini harus ditampilkan pada layar monitor agar dapat dibaca oleh pengguna program. Sama seperti aturan pada simbol-simbol sebelumnya, penulisan hasil pemrosesan data dapat dilakukan secara satu per satu maupun secara keseluruhan.

Gambar 2.4. Simbol keluaran

5) Percabangan
Yang dimaksud dengan percabangan disini adalah suatu kegiatan untuk mengecek atau memeriksa suatu keadaan apakah memenuhi suatu kondisi tertentu atau tidak. Jadi dalam percabangan ini, kita harus menuliskan kondisi apa yang harus dipenuhi oleh suatu keadaan. Hasil dari pemeriksaan keadaan ini adalah YA atau TIDAK. Jika pemeriksaan keadaan menghasilkan kondisi yang benar, maka jalur yang dipilih adalah jalur yang berlabel YA, sedangkan jika pemeriksaan keadaan menghasilkan kondisi yang salah, maka jalur yang

dipilih adalah jalur yang berlabel TIDAK. Berbeda dengan aturan pada simbol-simbol sebelumnya, penulisan kondisi harus dilakukan secara satu per satu (satu notasi percabangan untuk satu kondisi).

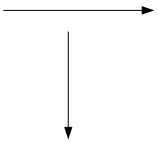


Gambar 2.5. Simbol percabangan

6) Sub rutin
Sub rutin adalah suatu bagian dalam program yang dapat melakukan (atau diberi) tugas tertentu. Jadi sub rutin merupakan "program kecil" yang menjadi bagian dari suatu program yang besar. Sub rutin ada dua macam, yaitu prosedur (procedure) dan fungsi (function). Perbedaan antara keduanya adalah setelah dipanggil prosedur tidak mengembalikan suatu nilai sedangkan fungsi selalu mengembalikan suatu nilai.
Dalam bahasa C++ kedua sub rutin tersebut dijadikan satu yaitu function, sedangkan untuk Java menggunakan class dimana keduanya bisa diatur untuk dapat mengembalikan nilai atau tidak dapat mengembalikan nilai. Sub rutin ini akan didiskusikan pada bab tersendiri.
Sub rutin memiliki suatu flow chart yang berdiri sendiri diluar flow chart utama. Jadi dalam simbol sub rutin, kita cukup menuliskan nama sub rutinnya saja, sama seperti jika kita melakukan pemanggilan terhadap suatu sub rutin pada program utama (main program) yang akan anda pelajari pada bagian atau bab lain pada buku ini. Aturan penulisan simbol sub rutin sama dengan simbol percabangan, yaitu penulisan nama sub rutin dilakukan secara satu per satu.

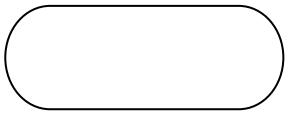
Gambar 2.6. Simbol sub rutin

7) Arah aliran
Arah aliran merupakan jalur yang harus diikuti dan merupakan garis penghubung yang menghubungkan setiap langkah pemecahan masalah yang ada dalam flow chart. Arah aliran ini disimbolkan dengan anak panah



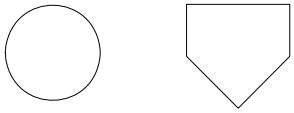
Gambar 2.7. Simbol arah aliran

8) Terminator
Terminator berfungsi untuk menandai titik awal dan titik akhir dari suatu flow chart. Simbol terminator ini diberi label MULAI untuk menandai titik awal dari flow chart dan label SELESAI untuk menandai titik akhir dari flow chart. Jadi dalam sebuah flow chart harus ada dua simbol terminator, yaitu simbol terminator untuk MULAI dan SELESAI.



Gambar 2.8. Simbol terminator

9) Konektor
Konektor berfungsi untuk menghubungkan suatu langkah dengan langkah lain dalam sebuah flow chart dengan keadaan on page atau off page. Yang dimaksud dengan konektor on page adalah konektor yang digunakan untuk menghubungkan suatu langkah dengan langkah lain dalam satu halaman. Sedangkan konektor off page adalah konektor untuk menghubungkan suatu langkah dengan langkah lain dalam halaman yang berbeda. Konektor ini digunakan apabila ruang gambar yang kita gunakan untuk menggambar flow chart tidak cukup luas untuk memuat flow chart secara utuh, jadi perlu dipisahkan atau digambar di halaman yang berbeda.



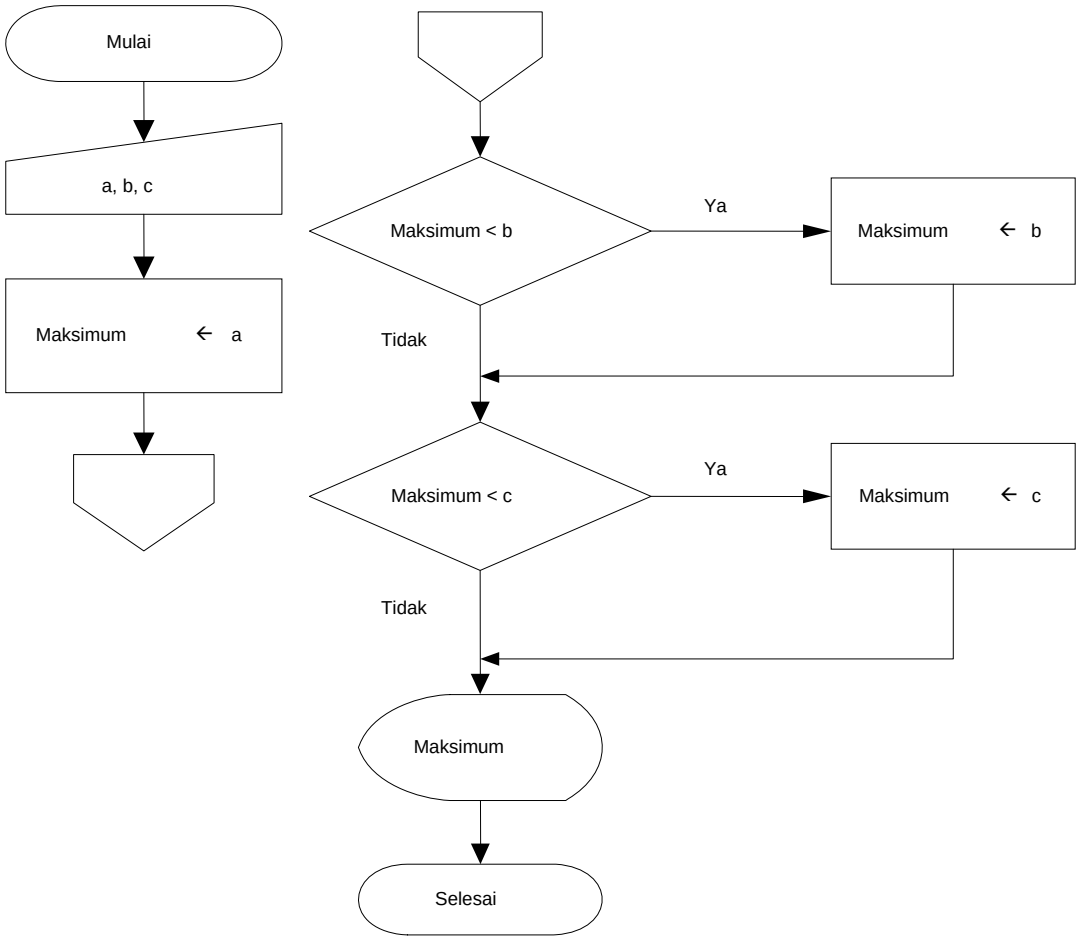
(a) (b)

Gambar 2.9. Simbol konektor on page (a) dan off page (b)

10) Dokumen
Dokumen merupakan tampilan data secara fisik yang dapat dibaca oleh manusia. Data ini biasanya merupakan hasil pemecahan masalah (informasi) yang telah dicetak (print out).

Gambar 2.10. Simbol dokumen

Berikut adalah flow chart untuk permasalahan yang diberikan (mencari bilangan terbesar dari tiga bilangan acak yang dimasukkan)



Gambar 2.11. Flowchart

2.10. Pseudo code

Pseudo code adalah algoritma yang bentuknya (strukturnya) sangat mirip dengan bahasa pemrograman khususnya bahasa pemrograman terstruktur seperti pascal. Kemiripan ini merupakan keuntungan dari pseudo code karena implementasi atau penerjemahan algoritma ke dalam source code suatu bahasa pemrograman sangatlah mudah meskipun penggunaannya tidak sepopuler flow chart.

Dalam penulisannya, pseudo code harus terdiri dari tiga bagian, yaitu :

1. Judul algoritma
   Bagian yang terdiri atas nama algoritma dan penjelasan

(spesifikasi) dari algoritma tersebut. Nama sebaiknya singkat dan menggambarkan apa yang dapat dilakukan oleh algoritma tersebut.

2. Deklarasi
   Bagian untuk mendefinisikan semua nama yang digunakan di dalam program. Nama tersebut dapat berupa nama tetapan, peubah atau variabel, tipe, prosedur, dan fungsi.
3. Deskripsi
   Bagian ini berisi uraian langkah-langkah penyelesaian masalah yang ditulis dengan menggunakan aturan-aturan yang akan dijelaskan selanjutnya.

Algoritma untuk permasalahan di atas yaitu mencari bilangan terbesar dari tiga bilangan acak yang dimasukkan dengan menggunakan pseudo code adalah:

```
Algoritma bilangan_terbesar
{algoritma ini mencari bilangan terbesar dari tiga bilangan yang dimasukkan}
deklarasi
        a,b,c,mak : integer
deskripsi
        read(a,b,c)
        mak ← a
        if (mak<b)
              mak ← b
        else if(mak<c)
              mak ← c
        end if
        write(mak)
```

Dalam pseudo code, garis bawah harus digunakan untuk kata algoritma (yang diikuti oleh judul dari algoritma), kata deklarasi, kata deskripsi, tipe data, read, write, if, else, end if, for, end for, while, end while, do while, dan end do while

2.11. Penerjemahan ke kode sumber

Sebelum membahas mengenai penerjemahan algoritma ke dalam bentuk source code bahasa pemrograman (program komputer) C++ atau bahasa lainnya seperti Java maka sebaiknya kita mengetahui lebih dahulu langkah-langkah yang biasa dilakukan untuk membuat suatu program komputer. Langkah-langkah tersebut adalah :

1. Mendefinisikan permasalahan
   Ini merupakan langkah pertama yang sering dilupakan orang. Menurut hukum Murphy (Henry Ledgard): “Semakin cepat menulis program, akan semakin lama kita dapat menyelesaikannya”. Hal ini terutama dirasakan apabila kita membuat program untuk

permasalahan yang komplek. Oleh karena itu sebelum kita menulis source code suatu program sebaiknya kita menentukan atau mengidentifikasi terlebih dahulu inti dari permasalahannya, kemudian apa saja yang dapat dan harus dipecahkan dengan bantuan komputer, dan yang terakhir adalah apa masukan untuk programnya dan bagaimana nanti keluarannya.

2. Menemukan solusi
   Setelah mendefinisikan masalah, maka langkah berikutnya adalah menentukan solusinya. Jika permasalahan terlalu komplek, maka ada baiknya masalah tersebut dipecah menjadi beberapa modul kecil dapat berupa prosedure, fungsi, atau class sehingga akan lebih mudah untuk diselesaikan. Penggunaan modul ini akan membuat program utamanya menjadi lebih singkat, mudah untuk dilihat dan dianalisis untuk tujuan debuging serta untuk pengembangan dari program.
3. Memilih algoritma
   Pilihlah (atau buatlah) algoritma yang benar-benar sesuai dan efisien untuk permasalahan yang diberikan.
4. Memilih bahasa pemrograman dan menulis source code program
   Bahasa pemrograman yang digunakan hendaknya bahasa yang memang sudah anda kuasai dengan baik. Atau jika masih dalam tahap belajar, pilihlah bahasa pemrograman yang mudah dipelajari dan digunakan serta memiliki tingkat kompatibilitas tinggi dengan perangkat keras dan platform (sistem operasi) lainnya.
5. Menguji program
   Setelah selesai menulis source code (program sudah jadi), ujilah program tersebut dengan segala macam kemungkinan yang ada, termasuk error-handling, sehing-ga program tersebut benar-benar handal dan layak untuk digunakan.
6. Menulis dokumentasi
   Dokumentasi sangatlah penting agar pada suatu saat jika kita ingin mengembangkan program, kita tidak mengalami kesulitan dalam membaca source code yang sudah kita tulis. Cara paling mudah dan sederhana dalam membuat suatu dokumentasi adalah dengan menuliskan komentar-komentar kecil pada suatu baris (atau suatu bagian dari source code) tentang apa maksud dari kode-kode tersebut dan apa kegunaannya, kemudian variabel apa saja yang digunakan dan untuk apa, serta parameter-parameter yang ada pada suatu prosedur dan fungsi.
7. Merawat program
   Program yang sudah jadi sebaiknya juga dilakukan perawatan untuk mencegah munculnya bug yang sebelumnya tidak terdeteksi. Selain itu juga berguna untuk menambah fasilitas-fasilitas baru yang dahulu sewaktu dibuat belum ada.

Pada bab ini kita hanya fokus pada langkah pertama sampai dengan langkah keempat saja. Sedangkan untuk dapat menerjemahkan notasi algoritma yang telah dibuat ke dalam bentuk

source code suatu bahasa pemrograman, seharusnya kita mamahami terlebih dahulu bahasa pemrograman yang akan kita gunakan, seperti aturan tata bahasanya, intruksi-instruksi yang digunakan, tipe data, dan masih banyak lagi. Semua itu akan kita pelajari satu persatu dibagian lain pada buku ini. Oleh karena kita baru akan mempelajari bahasa pemrograman yaitu C++ atau program lainnya seperti java, maka untuk mempermudah pemahaman tentang algoritma akan diberikan contoh-contoh permasalahan sederhana yang sering dijumpai dalam pemrograman, bagaimana algoritmanya dalam bentuk tiga notasi algoritma yang telah diberikan, dan kemudian bagaimana mengimplementasikan atau menerjemakan algoritma tersebut ke dalam bentuk source code bahasa pemrograman C++ atau program lainnya seperti java.

Untuk membantu penerjemahan algoritma kita juga harus memperhatikan jenis-jenis proses yang biasanya kita temukan dalam algoritma. Dalam algoritma ada empat jenis proses yang dikenal, yaitu :

1. Sequence Process, merupakan instruksi yang dikerjakan secara sekuensial (berurutan).
2. Selection Process, adalah instruksi yang dikerjakan jika memenuhi suatu kriteria atau kondisi (keadaan) tertentu.
3. Iteration Process, yaitu instruksi yang dikerjakan selama memenuhi suatu kriteria atau kondisi (keadaan) tertentu.
4. Concurrent Process, beberapa instruksi dikerjakan secara bersama.

Berikut merupakan contoh-contoh permasalahan, algoritma penyelesaian dalam tiga notasi algoritma serta penerjemahannya dalam bahasa pemrograman C++ atau program lainnya seperti java.

1. Masalah penukaran isi ember (contoh sequence process).
   Masalah penukaran isi ember dapat kita aplikasikan untuk masalah berikut ini. Mula-mula A bernilai 19 dan B bernilai 33. Jika nilai kedua variabel A dan B tersebut ditukar, A menjadi 33 dan B menjadi 19, maka algoritmanya adalah :

Menggunakan uraian kalimat deskriptif

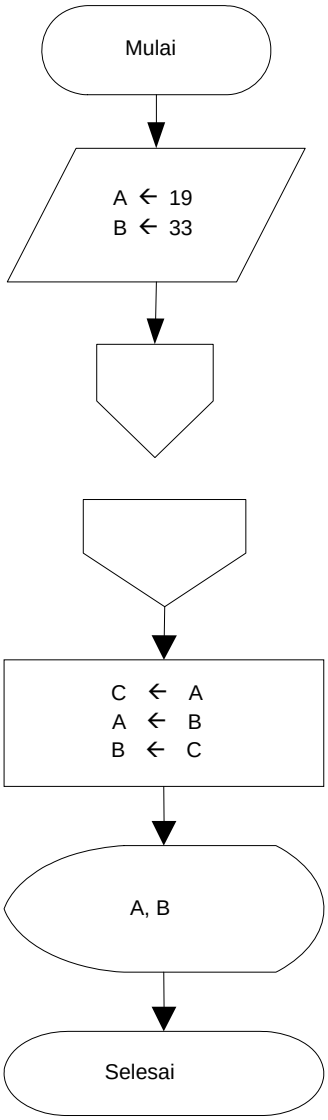
1) Set nilai variabel A ← 19 dan B ← 33
2) Set nilai dari variabel C menjadi sama dengan variabel A.
3) Set nilai variabel A menjadi sama dengan B.
4) Set nilai variabel B menjadi sama dengan C.
5) Tampilkan nilai variabel A dan B.

Algoritma diatas dapat juga ditulis seperti dibawah ini:

1) A ← 19 dan B ← 33
2) C ← A
3) A ← B

4) B ← C
5) Tulis A dan B

Jika algoritma tersebut diatas digambar menggunakan flow chart adalah sebagai berikut:



Gambar 2.12. Gambar flowchart

a) Menggunakan pseudo code

```
Algoritma tukar_data
{algoritma ini digunakan untuk menukarkan dua buah data }
deklarasi
        a,b,c : integer
deskripsi
        a ← 19
        b ← 33
        c ← a
        a ← b
        b ← c
        write(a,b)
```

Dari ketiga notasi algoritma di atas dapat disimpulkan bahwa langkah-langkah pemecahan masalah yang telah dibuat dapat dibagi dalam tiga kelompok, yaitu memberi nilai variabel A dan B, menukarkan nilainya, dan terakhir adalah menampilkannya. Oleh karena itu penerjemahan notasi algoritma ke dalam bentuk source code bahasa pemrograman juga harus mengikuti langkah-langkah yang diberikan dalam ketiga algoritma. Berikut adalah penerjemahan algoritma ke dalam bahasa C++.

```
1.  #include <iostream>
2.  using namespace std;
3.  int main() {
4.      int a = 19, b = 33;
5.      int c = a;
6.      a = b;
7.      b = c;
8.      cout << “a = “ << a << endl;
9.      cout << “b = “ << b;
10.     return 0;
11.}
```

Langkah ke-4 merupakan pemberian nilai variabel A dan B. Langkah selanjutnya, langkah ke-5 sampai dengan langkah ke-7, merupakan proses penukaran data. Dan langkah terakhir, yaitu menampilkan data yang telah ditukar, ada dalam langkah ke-8 dan 9. Langkah-langkah yang lainnya merupakan aturan tata bahasa dalam bahasa C++ yang akan dijelaskan dalam bab-bab selanjutnya.

Sedangkan penerjemahan algoritma-algoritma tersebut dalam bahasa Java adalah sebagai berikut:

```
1.  class tukarData {
2.      public static void main(String[ ] args) {
3.          int a=19, b=33;
4.          int c = a;
5.          a = b;
6.          b = c;
7.          System.out.println("a = " + a);
8.          System.out.println("b = " + b);
9.      }
10. }
```

Langkah ke-3 merupakan pemberian nilai variabel A dan B, langkah ke-4 sampai dengan langkah ke-6 merupakan proses penukaran data, dan langkah ke-7 dan 8 merupakan kode untuk menampilkan data yang telah ditukar. Sedangkan langkah yang lainnya merupakan aturan tata bahasa Java yang akan dijelaskan dalam bab-bab selanjutnya.

2. Masalah mencari bilangan terbesar dari tiga bilangan yang dimasukkan (contoh selection process).

   Algoritma pada permasalahan ini sudah diberikan, silakan lihat kembali sub bab notasi algoritma. Dari ketiga notasi algoritma tersebut dapat kita ketahui bahwa ada dua proses pemeriksaan keadaan (data). Pada algoritma dengan uraian kalimat deskriptif, proses pemeriksaan ini dapat kita identifikasi dengan adanya kata kunci JIKA. Dengan demikian pada saat menerjemahkan langkah ke-3 dan langkah ke-5 dari algoritma tersebut ke dalam bentuk source code bahasa C++ atau program lainnya seperti java, kita harus menggunakan klausa if().

   Sedangkan pada algoritma dengan flow chart, proses pemeriksaan keadaan disimbolkan dengan simbol percabangan dimana kalau jalurnya kita telusuri, baik jalur YA maupun TIDAK, akan terhubung dengan langkah penyelesaian masalah selanjutnya, tidak kembali ke langkah sebelumnya ataupun kembali ke simbol percabangan semula karena jika ini terjadi kita tidak bisa menggunakan klausa if() dalam penerjemahan ke bentuk source code bahasa C++ atau program lainnya seperti java (lihat contoh permasalahan berikutnya, yaitu contoh ke-3 dan ke-4).

   Untuk algoritma dengan pseudo code, penerjemahan akan jauh lebih mudah karena seperti yang telah dikemukakan sebelumnya bahwa pseudo code ini strukturnya sangat mirip

dengan penulisan source code bahasa pemrograman.

Dengan demikian penerjemahan algoritma ke dalam bahasa C++ adalah (perhatikan source code baris ke-12 dan 13):

```
1.  #include <iostream>
2.    using namespace std;
3.    int main() {
4.        int a,b,c,mak;
5.        cout << "Bilangan pertama = ";
6.        cin >> a;
7.        cout << "Bilangan kedua   = ";
8.        cin >> b;
9.        cout << "Bilangan ketiga  = ";
10.           cin >> c;
11.           mak = a;
12.           if (mak < b) mak = b;
13.           else if (mak < c) mak = c;
14.               cout << "Bilangan terbesarnya = " << mak;
15.           return 0;
16.   }
```

Sedangkan penerjemahan dalam bahasa lain misalnya bahasa Java adalah sebagai berikut (perhatikan source code baris ke-14 dan 15)

```
1.  import java.util.Scanner;
2.  import java.io.*;
3.  class bilTerbesar {
4.      public static void main(String[ ] args) {
5.          int a,b,c;
6.          Scanner input = new Scanner(System.in);
7.          System.out.print("Bilangan pertama = ");
8.          a = input.nextInt();
9.          System.out.print("Bilangan kedua   = ");
10.         b = input.nextInt();
11.         System.out.print("Bilangan ketiga  = ");
12.         c = input.nextInt();
13.         int mak = a;
14.         if (mak < b) mak = b;
15.         else if (mak < c) mak = c;
16.         System.out.print("Bilangan terbesarnya  = " + mak);
```

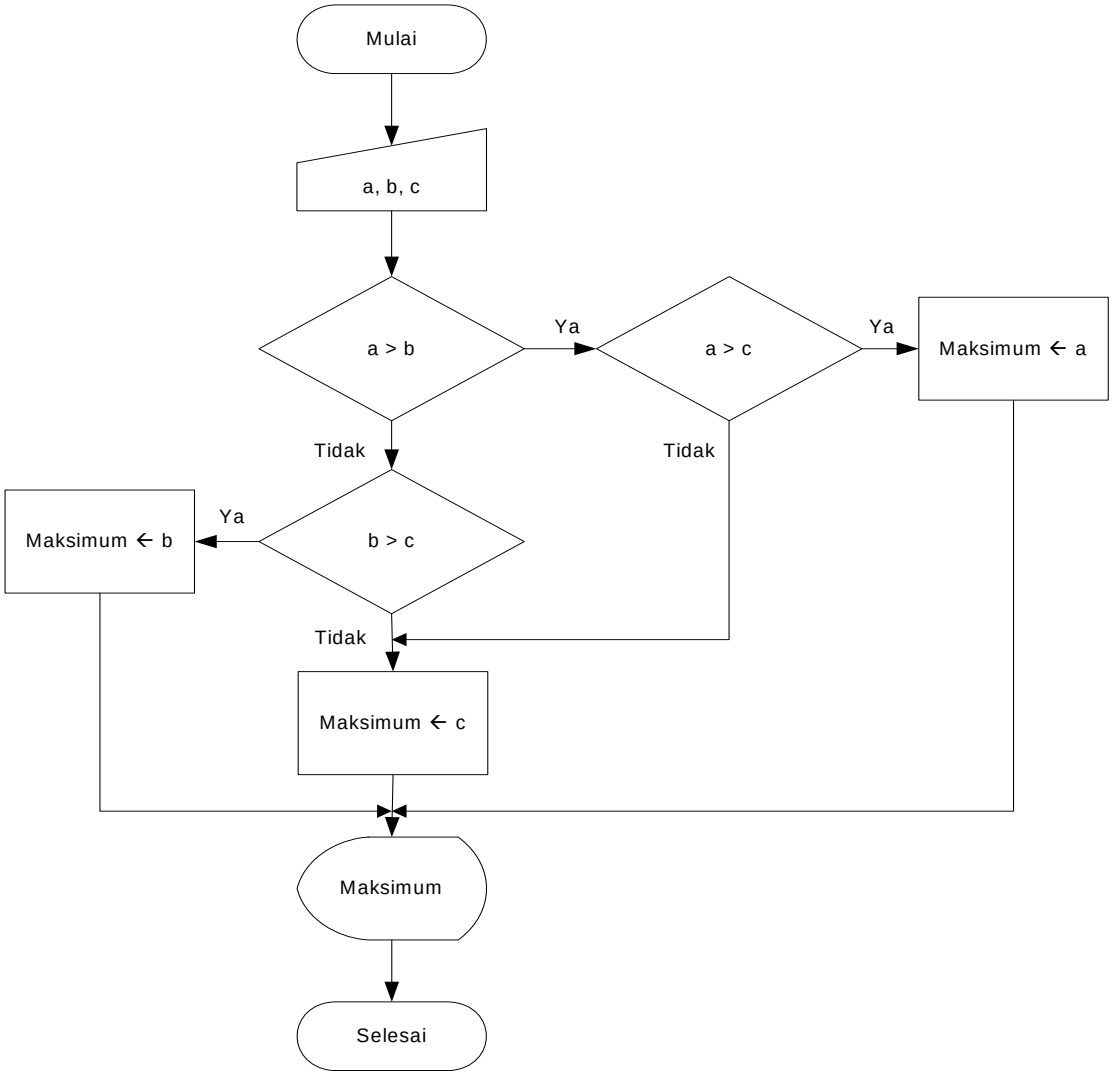
```
17.     }
18. }
```

Pada permasalahan ini, kita juga dapat menggunakan algoritma lain, yaitu :

a) Menggunakan uraian kalimat deskriptif

1) Masukkan a, b, dan c.
2) Jika a > b, maka kerjakan langkah ke-3. Jika tidak, kerjakan langka ke-5.
3) Jika a > c, maka kerjakan langkah ke-4. Jika tidak, kerjakan langka ke-7.
4) mak ← a.
5) Jika b > c, kerjakan langkah ke-6. Jika tidak, kerjakan langkah ke-7.
6) mak ← b.
7) mak ← c.
8) Tulis mak.

b) Menggunakan flow chart



Gambar 2.13. Flowcart

a) Menggunakan pseudo code

Algoritma bilangan_terbesar
{algoritma ini mencari bilangan terbesar dari tiga bilangan yang dimasukkan secara acak}

```
deklarasi
    a,b,c,mak : integer
deskripsi
    read(a,b,c)
    if (a > b)
        if (a > c)
            mak ← a
        else mak ← c
        end if
    else
        if(b > c)
            mak ← b
        else mak ← c
        end if
    end if
write(mak
```

Pada algoritma dengan uraian kalimat deskriptif kita dapat mengidentifikasi bahwa terdapat tiga buah pemeriksaan keadaan, yaitu langkah ke-2, 3, dan 5. Pada langkah ke-2, jika kondisi terpenuhi, langkah selanjutnya, yaitu langkah ke-3, adalah merupakan pemeriksaan keadaan lagi. Demikian pula jika kondisi yang telah ditentukan tidak terpenuhi, langkah berikutnya, yaitu langkah ke-5, adalah merupakan pemeriksaan keadaan kembali. Dari sini dapat disimpulkan bahwa nantinya penggunaan klausa if() yang kedua (langkah ke-3) akan berada di dalam klausa if() yang pertama (langkah ke-2). Sedangkan penggunaan klausa if() yang ketiga (langkah ke-5) juga ada dalam klausa if() yang pertama pada bagian else. Hal ini dinamakan percabangan bersarang, yaitu percabangan yang ada dalam percabangan lainnya.

Keberadaan percabangan bersarang ini juga dapat diidentifikasi dari simbol percabangan yang terhubung dengan simbol percabangan lainnya. Identifikasi ini lebih jelas terlihat pada algoritma dengan pseudo code. Dengan demikian penerjemahan algoritma ke dalam bahasa C++ adalah:

```
1.  #include <iostream>
2.  using namespace std;
3.  int main() {
4.      int a,b,c,mak;
5.      cout << "Bilangan pertama = ";
6.      cin >> a;
```

```
7.      cout << "Bilangan kedua   = ";
8.      cin >> b;
9.      cout << "Bilangan ketiga  = ";
10.     cin >> c;
11.     if (a > b) {
12.         if (a > c)
13.             mak = a;
14.         else mak = c;
15.     }
16.     else {
17.         if (b > c)
18.             mak = b;
19.         else mak = c;
20.     }
21.     cout << "Bilangan terbesarnya = " << mak;
22.     return 0;
23.  }
```

Klausa if() pada source code baris ke-12 sampai dengan baris ke-14 ada di dalam klausa if() baris ke-11. Sedangkan klausa if() baris ke-17 sampai dengan baris ke-19 juga ada di dalam klausa if() (baris ke-11) bagian else (baris ke-16). Sedangkan penerjemahan algoritma ke dalam bahasa Java adalah (perhatikan baris ke-13 sampai dengan baris ke-22) :

```
1.  import java.util.Scanner;
2.  import java.io.*;
3.  class bilBesar {
4.      public static void main(String[ ] args) {
5.          int a,b,c,mak;
6.          Scanner input = new Scanner(System.in);
7.          System.out.print("Bilangan pertama = ");
8.          a = input.nextInt();
9.          System.out.print("Bilangan kedua   = ");
10.         b = input.nextInt();
11.         System.out.print("Bilangan ketiga  = ");
12.         c = input.nextInt();
13.         if (a > b) {
14.                 if (a > c)
15.                     mak = a;
16.                 else mak = c;
17.         }
```

```
18.         else {
19.                 if (b > c)
20.                     mak = b;
21.                 else mak = c;
22.         }
23.         System.out.print("Bilangan terbesarnya  = " + mak);
24.     }
25.  }
```

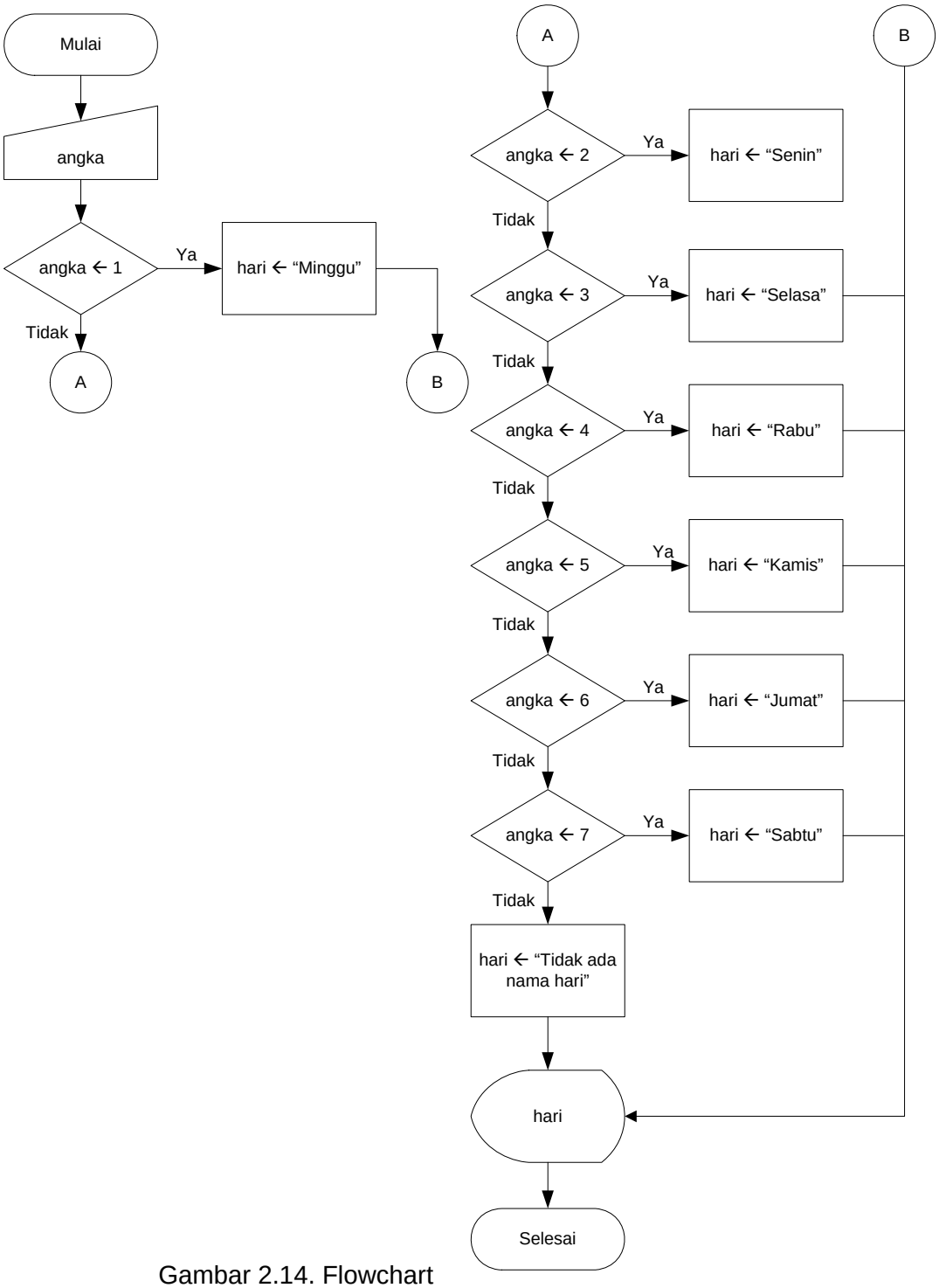
3. Diminta suatu program dimana jika dimasukkan suatu bilangan (1 sampai dengan 7) maka akan menampilkan nama hari, yaitu :
   - Bilangan 1 untuk hari Minggu.
   - Bilangan 2 untuk hari Senin.
   - Bilangan 3 untuk hari Selasa.
   - Bilangan 4 untuk hari Rabu.
   - Bilangan 5 untuk hari Kamis.
   - Bilangan 6 untuk hari Jumat.
   - Bilangan 7 untuk hari Sabtu.

   Algoritmanya adalah :

Menggunakan uraian kalimat deskriptif

1) Masukan bilangan sebuah bilangan bulat (angka).
2) Jika angka ← 1, maka kerjakan langkah ke-3. Jika tidak, kerjakan langkah ke-4.
3) hari ← “Minggu”.
4) Jika angka ← 2, maka kerjakan langkah ke-5. Jika tidak, kerjakan langkah ke-6.
5) hari ← “Senin”.
6) Jika angka ← 3, maka kerjakan langkah ke-7. Jika tidak, kerjakan langkah ke-8.
7) hari ← “Selasa”.
8) Jika angka ← 4, maka kerjakan langkah ke-9. Jika tidak, kerjakan langkah ke-10.
9) hari ← “Rabu”.
10) Jika angka ← 5, maka kerjakan langkah ke-11. Jika tidak, kerjakan langkah ke-12.
11) hari ← “Kamis”.
12) Jika angka ← 6, maka kerjakan langkah ke-13. Jika tidak, kerjakan langkah ke-14.
13) hari ← “Jumat”.
14) Jika angka ← 7, maka kerjakan langkah ke-15. Jika tidak, kerjakan langkah ke-16.
15) hari ← “Sabtu”.
16) hari ← “Tidak ada nama hari untuk angka tersebut”
17) Tampilkan hari.

a) Menggunakan flow chart



Gambar 2.14. Flowchart

b) Menggunakan pseudo code

Algoritma nama_hari

```
{algoritma ini digunakan untuk mengetahui nama hari dari bilangan bulat yang dimasukkan}
deklarasi
    angka   : integer
    hari    : char
deskripsi
    read(angka)
    if(angka ← 1)
        hari ← "Minggu"
    else if(angka ← 2)
        hari ← "Senin"
    else if(angka ← 3)
        hari ← "Selasa"
    else if(angka ← 4)
        hari ← "Rabu"
    else if(angka ← 5)
        hari ← "Kamis"
    else if(angka ← 6)
        hari ← "Jumat"
    else if(angka ← 7)
        hari ← "Sabtu"
    else
        hari ← "Tidak ada nama hari"
    end if
    write(hari)
```

Algoritma diatas dapat juga ditulis seperti dibawah ini:

Algoritma nama_hari

```
{algoritma ini digunakan untuk mengetahui nama hari dari bilangan bulat yang dimasukkan}
deklarasi
    angka   : integer
    hari    : char
deskripsi
    read(angka)
    switch(angka)
        case 1 : hari ← "Minggu"
        case 2 : hari ← "Senin"
```

```
        case 3 : hari ← “Selasa”
        case 4 : hari ← “Rabu”
        case 5 : hari ← “Kamis”
        case 6 : hari ← “Jumat”
        case 7 : hari ← “Sabtu”
        default : hari ← “Tidak ada nama hari”
    end switch
    write(hari)
```

Pada algoritma dengan uraian kalimat deskriptif kita dapat mengidentifikasi bahwa terdapat tujuh buah pemeriksaan keadaan, yaitu langkah ke-2, 4, 6, 8, 10, 12, dan 14. Pemeriksaan yang sebanyak ini sangat tidak efektif kalau kita menggunakan klausa if(). Agar program kita lebih efisien, maka kita menggunakan switch() untuk pemeriksaan keadaan sebanyak ini. Dengan demikian penerjemahan algoritma ke dalam bahasa C++ adalah :

```
1.  #include <iostream>
2.  #include <cstring>
3.  using namespace std;
4.  int main() {
5.      int angka;
6.      string hari;
7.      cout << "Bilangan = ";
8.      cin >> angka;
9.      if (angka==1) hari="Minggu";
10.     else if (angka==2) hari="Senin";
11.     else if (angka==3) hari="Selasa";
12.     else if (angka==4) hari="Rabu";
13.     else if (angka==5) hari="Kamis";
14.     else if (angka==6) hari="Jumat";
15.     else if (angka==7) hari="Sabtu";
16.     else hari="tidak ada";
17.     cout << "Nama hari ke-" << angka << " adalah " << hari;
18.     return 0;
19. }
```

Program diatas dapat juga ditulis seperti dibawah ini:

```
1.  #include <iostream>
2.  #include <cstring>
3.  using namespace std;
4.  int main() {
5.      int angka;
6.      string hari;
7.      cout << "Bilangan = ";
8.      cin >> angka;
9.      switch(angka) {
10.         case 1 : hari = "Minggu";
11.                  break;
12.         case 2 : hari = "Senin";
13.                  break;
14.         case 3 : hari = "Selasa";
15.                  break;
16.         case 4 : hari = "Rabu";
17.                  break;
18.         case 5 : hari = "Kamis";
19.                  break;
20.         case 6 : hari = "Jumat";
21.                  break;
22.         case 7 : hari = "Sabtu";
23.                  break;
24.         default : hari = "tidak ada";
25.     }
26.     cout << "Nama hari ke-" << angka << " adalah = " << hari;
27.     return 0;
28. }
```

Sedangkan dalam contoh dalam bahasa Java adalah :

```
1.  import java.util.Scanner;
2.  import java.io.*;
3.  class bab2_03 {
4.    public static void main(String[ ] args) {
5.      int angka;
6.      String hari;
7.      Scanner input = new Scanner(System.in);
8.      System.out.print("Bilangan = ");
9.      angka = input.nextInt();
10.     if (angka==1) hari="Minggu";
```

```
11.      else if (angka==2) hari="Senin";
12.      else if (angka==3) hari="Selasa";
13.      else if (angka==4) hari="Rabu";
14.      else if (angka==5) hari="Kamis";
15.      else if (angka==6) hari="Jumat";
16.      else if (angka==7) hari="Sabtu";
17.      else hari="tidak ada";
18.      System.out.print("Nama hari ke-" + angka + " adalah = " + hari);
19.      }
20.   }
```

Program diatas dapat juga ditulis seperti dibawah ini:

```
1.   import java.util.Scanner;
2.   import java.io.*;
3.   class bab2_03_2 {
4.     public static void main(String[ ] args) {
5.       int angka;
6.       String hari;
7.       Scanner input = new Scanner(System.in);
8.       System.out.print("Bilangan = ");
9.       angka = input.nextInt();
10.      switch(angka) {
11.        case 1 : hari = "Minggu";
12.                 break;
13.        case 2 : hari = "Senin";
14.                 break;
15.        case 3 : hari = "Selasa";
16.                 break;
17.        case 4 : hari = "Rabu";
18.                 break;
19.        case 5 : hari = "Kamis";
20.                 break;
21.        case 6 : hari = "Jumat";
22.                 break;
23.        case 7 : hari = "Sabtu";
24.                 break;
25.        default : hari = "tidak ada";
26.      }
27.      System.out.print("Nama hari ke-" + angka + " adalah = " + hari);
28.   }
29.  }
```

4. Masalah menghitung rata-rata dari sejumlah data yang dimasukkan secara manual melalui keyboard (contoh iteration process). Algoritmanya adalah:

Menggunakan uraian kalimat deskriptif

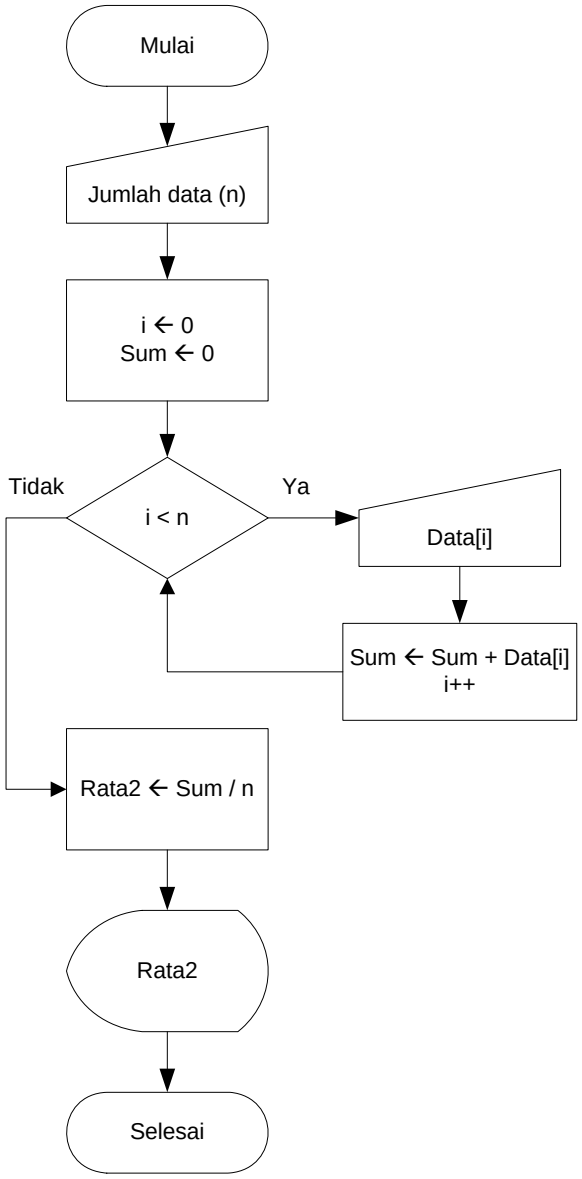
1) Masukan jumlah (n) data yang diinginkan.
2) Selama jumlah data yang dimasukkan kurang dari n, kerjakan langkah ke-3 dan ke-4. Jika tidak kerjakan langkah ke-5.
3) Masukan data.
4) Tambahkan data yang dimasukkan dengan data sebelumnya.
5) Hitung rata-rata data.
6) Tampilkan besar rata-ratanya.

Uraian diatas dapat juga ditulis seperti dibawah ini:

1) Masukkan n
2) i ← 0
3) Sum ← 0
4) Selama i < n, kerjakan langkah ke-5, 6, dan 7.
5) Masukkan data[i]
6) Sum ← Sum + data[i]
7) i++
8) Rata2 ← Sum / n
9) Tulis Rata2

Langkah ke-2 dan ke-3 merupakan proses inisialisasi atau pemberian nilai awal yang diperlukan dalam penulisan source code. Sebaiknya variabel Rata2 juga diinisialisasi agar keluaran dari program lebih valid. Variabel i digunakan sebagai counter (penghitung) untuk proses perulangan yang digunakan untuk memasukkan data satu persatu dan sekaligus menjumlahkan data yang dimasukkan dengan penjumlahan data sebelumnya dimana penjumalahan data sebelumnya disimpan dalam variabel Sum

b) Menggunakan flow chart



Gambar 2.15. Flowchart

Proses perulangan diperlukan karena banyak data yang dimasukkan belum diketahui dengan pasti (lihat langkah pertama yang meminta pengguna untuk memasukkan banyak data). Langkah ke-7 merupakan increment variabel i, yaitu menaikkan nilai variabel i

sebesar satu tingkat. Data pada permasalahan ini disimpan dalam tipe data array (data[i]). Tipe data array ini akan dibahas pada bab tersendiri.

c) Menggunakan pseudo code

Algoritma hitung_rata2

```
{algoritma ini digunakan untuk menghitung rata-rata dari sejumlah data yang
dimasukkan melalui keyboard}
deklarasi
    i, n, sum, data[20]    : integer
    rata2                  : float
deskripsi
    read(n)
    i ← 0
    sum ← 0
    while(i < n)
        read(data[i])
        sum += data[i]
        i++
    end while
    rata2 ← sum / n
    write(rata2)
```

Algoritma diatas dapat juga ditulis seperti dibawah ini:

Algoritma hitung_rata2

```
{algoritma ini digunakan untuk menghitung rata-rata dari sejumlah data yang
dimasukkan melalui keyboard}
deklarasi
    i, n, sum, data[20]    : integer
    rata2                  : float
deskripsi
    read(n)
    sum ← 0
    for(i=0; i<n; i++)
        read(data[i])
        sum += data[i]
    end for
    rata2 ← sum / n
    write(rata2)
```

Langkah ke-2 pada algoritma dengan uraian kalimat deskriptif yang pertama dan langkah ke-4 pada algoritma dengan uraian kalimat deskriptif yang kedua, terdapat kata kunci selama. Kata kunci ini mengidentifikasikan bahwa dalam penerjemahan ke dalam bentuk source code bahasa C++ atau program lainnya seperti java nanti kita akan menggunakan klausa untuk looping (perulangan) yaitu while() dan atau for() karena langkah-langkah tersebut memerintahkan kita untuk melakukan langkah berikutnya, bukan kembali ke langkah sebelumnya, jika kondisi yang telah ditentukan terpenuhi. While() dan for() dapat digunakan secara bergantian jika perulangan yang terjadi dapat diketahui banyaknya (jumlah perulangan yang terjdi), dalam hal ini banyak perulangannya adalah n, yaitu banyak data yang akan dimasukkan. Tetapi jika banyak perulangannya tidak dapat diketahui dengan pasti, maka kita harus menggunakan while(), tidak bisa dengan for(). Pada contoh berikutnya akan diberikan perulangan yang tidak diketahui berapa banyak perulangan yang terjadi.

Langkah-langkah yang masuk dalam proses perulangan ini adalah langkah ke-3 dan 4 pada algoritma dengan uraian kalimat deskriptif yang pertama dan langkah ke-5, 6, dan 7 pada algoritma dengan uraian kalimat deskriptif yang kedua. Pada bab selanjutnya anda akan mengetahui bahwa klausa perulangan ini, dalam bahasa C++ atau program lainnya seperti java, ada tiga macam bentuk yang dapat anda pilih, yaitu for(), while(), atau do while().

Untuk algoritma dengan flow chart, proses perulangan yang menggunakan klausa while() atau for() dapat diidentifikasi dari simbol percabangan dimana kalau jalurnya, dalam hal ini jalur yang berlabel YA, kita telusuri akan terhubung ke simbol berikutnya dan kemudian kembali lagi simbol percabangan semula. Sedangkan untuk algoritma dengan pseudo code, penerjemahan tidak akan mengalami kesulitan karena, sekali lagi, struktur dari pseudo code sangat mirip dengan penulisan source code bahasa pemrograman.

Dengan demikian penerjemahan algoritma ke dalam bahasa C++ adalah (perhatikan baris ke-8 sampai dengan baris ke-14 dimana baris ke-9 dan 10 merupakan baris tambahan agar program lebih user friendly).

```
1.  #include <iostream>
2.  using namespace std;
3.  int main() {
4.      int n, i=0, sum=0,  data[20];
5.      float rata2;
6.      cout << "Jumlah data (maksimum 20 data) = ";
```

```
7.      cin >> n;
8.      while(i < n) {
9.          int j = i + 1;
10.         cout << "Masukkan data ke-" << j << " : ";
11.         cin >> data[i];
12.         sum += data[i];
13.         i++;
14.     }
15.     rata2 = sum / n;
16.     cout << "Rata-ratanya = " << rata2;
17.     return 0;
18.  }
```

Program diatas dapat juga ditulis seperti dibawah ini:

```
1.  #include <iostream>
2.  using namespace std;
3.  int main() {
4.      int n, i, sum=0,  data[20];
5.      float rata2;
6.      cout << "Jumlah data (maksimum 20) = ";
7.      cin >> n;
8.      for(i=0; i < n; i++) {
9.          int j = i + 1;
10.         cout << "Masukkan data ke-" << j << " : ";
11.         cin >> data[i];
12.         sum += data[i];
13.     }
14.     rata2 = sum / n;
15.     cout << "Rata-ratanya = "        << rata2;
16.     return 0;
17.     }
```

Sedangkan jika ditulis dalam bahasa Java adalah sebagai berikut:

```
1.  import java.util.Scanner;
2.  import java.io.*;
3.  class bab2_03 {
4.      public static void main(String[ ] args) {
5.          int n, i=0, sum=0;
6.          int[ ] data = new int[20];
7.          float rata2;
8.          Scanner input = new Scanner(System.in);
```

```
9.          System.out.print("Jumlah data (mak. 20) = ");
10.         n = input.nextInt();
11.         while(i < n) {
12.           int j = i + 1;
13.           System.out.print("Masukkan data ke-" + j + " : ");
14.           data[i] = input.nextInt();
15.           sum += data[i];
16.           i++;
17.         }
18.         rata2 = sum / n;
19.         System.out.print("Rata-ratanya = " + rata2);
20.     }
21.   }
```

Program diatas dapat juga ditulis seperti dibawah ini:

```
1.  import java.util.Scanner;
2.  import java.io.*;
3.  class bab2_03_2 {
4.      public static void main(String[ ] args) {
5.          int n, i, sum=0;
6.          int[ ] data = new int[20];
7.          float rata2;
8.          Scanner input = new Scanner(System.in);
9.          System.out.print("Jumlah data (mak. 20) = ");
10.         n = input.nextInt();
11.         for(i=0;i < n;i++) {
12.           int j = i + 1;
13.           System.out.print("Data ke-" + j + " : ");
14.           data[i] = input.nextInt();
15.           sum += data[i];
16.         }
17.         rata2 = sum / n;
18.         System.out.print("Rata-ratanya = " + rata2);
19.     }
20.   }
```

5. Masalah menentukan Faktor Persekutuan Besar (FPB).
   Untuk memecahkan persoalan FPB di atas, kita dapat menggunakan algoritma Euclid yang langkah-langkahnya adalah sebagai berikut :
   1) Diberikan dua buah bilangan bulat positif, misalnya m dan n dimana syaratnya adalah m > n.

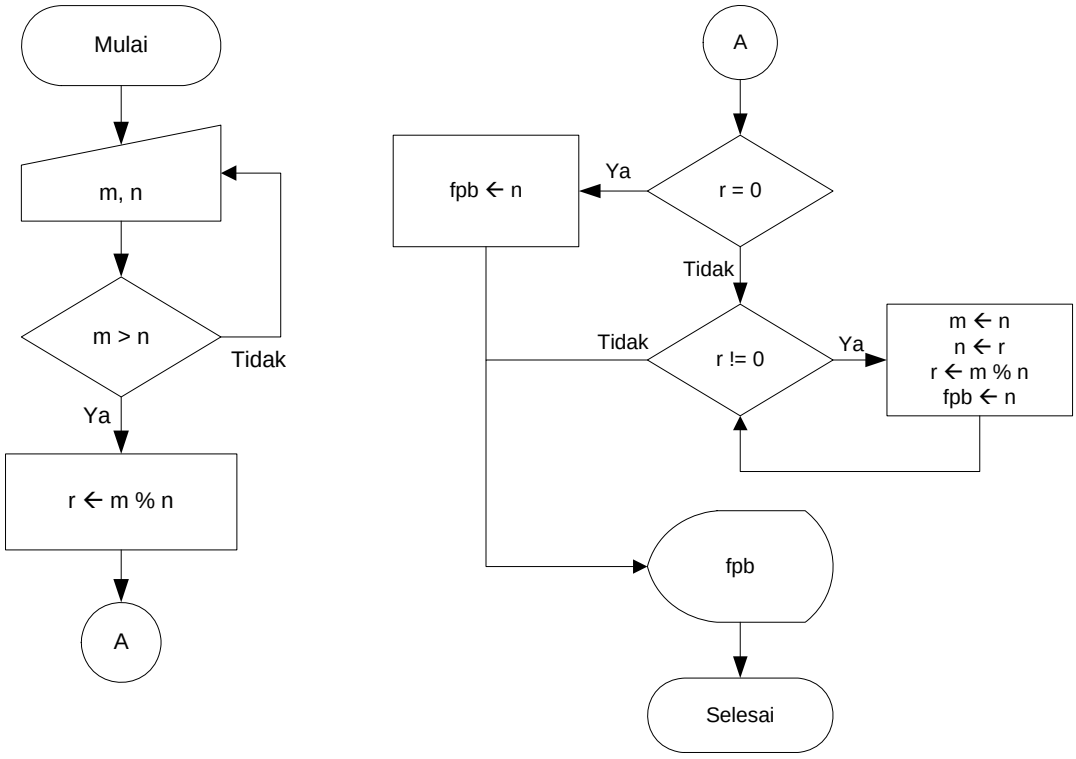
2) Bagilah m dengan n, misalnya hasilnya adalah r.
3) Apabila r = 0, maka hasilnya adalah n itu sendiri (merupakan FPB dari m dan n).
4) Jika r $\neq 0$ maka proses pencarian dilanjutkan yaitu ganti m dengan n dan n dengan r dan ulangi langkah ke-2, demikian selanjutnya.

Misalnya akan dicari FPB dari 36 dan 20, maka penyelesaiannya menurut algoritma Euclid adalah :

1. $\frac{m}{n} = \frac{36}{20}$ → mempunyai sisa r = 16
2. r = 16 → r ≠ 0
3. m = 20 dan n = 16
4. $\frac{m}{n} = \frac{20}{16}$ → mempunyai sisa r = 4
5. r = 4 → r ≠ 0
6. m =16 dan n = 4
7. $\frac{m}{n} = \frac{16}{4}$ → mempunyai sisa r = 0

Jadi FPB(36, 20) = 4

b) Menggunakan flow chart



Gambar 2.16. Flowchart

Algoritma Euclid di atas merupakan algoritma dengan uraian kalimat deskriptif. Algoritma tersebut dapat ditulis kembali sebagai berikut:

a) Menggunakan uraian kalimat deskriptif

1) Masukkan m dan n.
2) Selama m < n ulangi kembali langkah pertama.
3) r ← m % n
4) Jika r = 0 maka fpb← n.Jika tidak kerjakan langkah ke-5.
5) Selama r != 0 kerjakan langkah ke-6 sampai ke-8.
6) m ← n dan n ← r
7) r ← m % n
8) fpb ← n
9) Tulis fpb.

c) Menggunakan pseudo code

```
{algoritma ini digunakan untuk mencari faktor persekutuan besar dari dua buah bilangan yang dimasukkan}
deklarasi
    m, n, r, fpb   : integer
deskripsi
    do read(m,n)
    while(m<n)
    end do while
    r ← m % n
    if (r == 0) fpb ← n
    else
        while(r != 0)
            m ← n
            n ← r
            r ← m % n
            fpb ← n
        end while
    end if
    write(fpb)
```

Langkah ke-2 pada algoritma dengan uraian kalimat deskriptif mempunyai kata kunci SELAMA yang mengidentifikasikan bahwa dalam penerjemahannya ke dalam source code bahasa pemrograman nanti menggunakan klausa untuk perulangan. Akan tetapi kita tidak dapat menggunakan klausa while() seperti pada contoh sebelumnya karena pada kasus ini langkah tersebut memerintahkan untuk kembali ke langkah sebelumnya jika kondisi yang diberikan terpenuhi.

Oleh karena itu klausa yang paling tepat digunakan adalah do while(). Sedangkan langkah ke-5 kita gunakan klausa while() karena langkah ke-5 memerintahkan kita untuk melakukan langkah-langkah berikutnya, bukan kembali ke langkah sebelumnya. Kita tidak bisa menggunakan klausa for() karena, sesuai dengan penjelasan pada contoh sebelumnya, kita tidak bisa mengetahui dengan pasti berapa banyak perulangan yang terjadi.

Dengan demikian dapat kita simpulkan bahwa jika setelah proses pemeriksaan keadaan memerintahkan kita untuk melakukan langkah-langkah berikutnya maka kita menggunakan klausa while(). Sebaliknya jika setelah proses pemeriksaan keadaan memerintahkan kita untuk kembali lagi melakukan langkah-langkah sebelumnya maka kita menggunakan klausa do while().

Untuk algoritma dengan flow chart, proses perulangan yang menggunakan klausa do while() dapat diidentifikasi dari simbol percabangan dimana kalau jalurnya, dalam hal ini jalur yang berlabel TIDAK, kita telusuri akan terhubung ke simbol sebelumnya dan kemudian kembali lagi simbol percabangan semula.

Sedangkan untuk algoritma dengan pseudo code, penerjemahan tidak akan mengalami kesulitan karena, sekali lagi, struktur dari pseudo code sangat mirip dengan penulisan source code bahasa pemrograman. Dengan demikian penerjemahan algoritma ke dalam bahasa C++ adalah (perhatikan baris ke-5 sampai dengan baris ke-10 dan baris ke-14 sampai dengan baris ke-19)

```
1.  #include <iostream>
2.  using namespace std;
3.  int main() {
4.      int m,n,r,fpb;
5.      do {
6.          cout << "Masukkan bilangan pertama = ";
7.          cin >> m;
8.          cout << "Masukkan bilangan kedua   = ";
9.          cin >> n;
10.     } while (m < n);
11.     r = m % n;
12.     if (r==0) fpb = n;
13.     else {
14.         while(r!=0) {
15.             m = n;
16.             n = r;
17.             r = m % n;
```

```
18.             fpb = n;
19.         }
20.     }
21.     cout << "FPB-nya = " << fpb;
22.     return 0;
23. }
```

Sedangkan jika ditulis dalam bahasa Java adalah sebagai berikut:

```
1.  import java.util.Scanner;
2.  import java.io.*;
3.  class bab2_04 {
4.      public static void main(String[ ] args) {
5.          int m,n,r,fpb;
6.          Scanner input = new Scanner(System.in);
7.          do {
8.            System.out.print("Masukkan bilangan pertama = ");
9.            m = input.nextInt();
10.           System.out.print("Masukkan bilangan kedua   = ");
11.           n = input.nextInt();
12.         } while(m < n);
13.         r = m % n;
14.         fpb = 0;
15.         if (r == 0) fpb = n;
16.         else {
17.           while(r != 0) {
18.               m = n;
19.               n = r;
20.               r = m % n;
21.               fpb = n;
22.           }
23.         }
24.         System.out.print("Bilangan terbesarnya  = " + fpb);
25.     }
26. }
```

2.12. Soal Latihan

Jawablah soal latihan dibawah ini dengan baik dan benar.

1. Apa yang dimaksud dengan bahasa pemrograman
2. Apa yang dimaksud dengan compiler dan intepreter
3. Apa yang dimaksud dengan bahasa pemrograman prosedural dan terstruktur
4. Apa yang dimaksud dengan algoritma pemrograman

5. Sebutkan ciri-ciri algoritma pemrograman
6. Buatlah algoritma sederhana untuk mencari bilangan ganjil dan genap
7. Dari soal no 6 diatas buatlah diagram alirnya
8. Dari soal no 7 diatas tulislah kode semu (psudocode) diagram alir tersebut
9. Dari soal no 8 diatas tulislah kode sumbernya

BAB 3
TIPE DATA DAN OPERATOR

3.1. Pengertian Data

Data merupakan bentuk jamak dari bahasa Latin dengan kata *datum*, yang berarti fakta atau sesuatu yang diberikan. Data adalah kelompok simbol-simbol yang teratur dan mewakili kuantitas, tindakan, benda dan sebagainya. Dalam istilah umum data mewakili angka, karakter dan simbol-simbol lain yang berfungsi sebagai masukan untuk proses komputer. Data bisa berujut suatu keadaan, gambar, suara, huruf, angka, matematika, bahasa ataupun simbol-simbol lainnya yang bisa kita gunakan sebagai bahan untuk melihat lingkungan, obyek, kejadian ataupun suatu konsep.

Data belum mempunyai arti apabila tidak diolah. Data yang telah diolah menjadi sebuah bentuk yang berarti disebut informasi . Data yang mewakili simbol-simbol bukan merupakan informasi kecuali dalam pengertian tertentu. Pada komputer data disimpan dalam memori sebelum dan sesudah pemrosesan oleh mikroprosesor.

Jenis data dalam setiap bahasa pemrograman belum tentu sama, namun sebagian besar biasanya terbagi menjadi tiga, yaitu :

1. Data Numerik atau bilangan, yaitu jenis data yang digunakan dalam proses aritmatika atau proses matematis lainnya.
2. Data String, yaitu jenis data yang dapat terdiri dari berbagai macam karakter. Digunakan untuk proses yang non matematis.
3. Data Logika, yaitu data yang hanya terdiri dari dua satuan, yaitu benar (true) dan salah (false). Digunakan dalam suatu proses logika yang terdiri dari persamaan boolean.

3.2. Indentifier

Identifier adalah nama yang didefinisikan oleh programmer dan digunakan untuk mewakili sebuah elemen pada program. Nama variabel merupakan salah satu contoh dari identifier. Programmer dapat memilih sendiri nama sebuah variabel pada C++, selama tidak menggunakan salah satu dari kata kunci (*keyword)* yang dimiliki oleh C++. *Keywords* atau kata kunci merupakan "inti" pada bahasa dan memiliki tujuan tertentu. Tabel dibawah ini menunjukkan daftar lengkap kata kunci C++. Dan yang perlu diperhatikan pada bahasa C adalah bahwa semua harus huruf kecil.

asm	auto	break	bool	case
catch	char	class	const	const_cast
continue	default	delete	do	double
dynamic_cast	else	enum	explicit	extern
false	float	for	friend	goto
if	inline	int	long	mutable
namespace	new	operator	private	protected
public	register	reinterpret_cast	return	short
signed	sizeof	static	static_cast	struct
switch	template	this	throw	true
try	typedef	typeid	typename	union
unsigned	using	virtual	void	volatile
wchar_t	while			

Indentifier atau pengenal adalah nama yang diberikan untuk nama variable, nama konstanta, nama fungsi, nama objek, nama method, nama class, dan obyek yang lain yang didefinisikan oleh pemrogram.

Dalam menulis program harus selalu memilih nama variabel yang memberikan indikasi mengenai hal yang berhubungan dengan yang digunakan variable tersebut. Jika dimungkinkan mendeklarasikan sebuah variabel dengan nama seperti berikut ini:

```
int x;
```

maka x tidak termasuk sesuatu jenis tertentu,sehingga hal tersebut tidak memberikan petunjuk pada tujuan variabel. Perhatikan contoh berikut dibawah yang lebih baik.

```
int daftarUrutan;
```

Nama daftarUrutan memberikan sesbuat pada pembaca program sehingga akan ide dari variabel yang digunakan. Cara coding ini akan membantu dalam menghasilkan dokumentasi program sendiri, yang berarti anda akan mendapatkan pemahaman tentang apa yang dilakukan program hanya dengan membaca kode yang digunakannya.

Karena dunia program biasanya ada ribuan baris, hal ini penting agar bisa diingat oleh programmer itu sendiri dan sebagai dokumentasi yang baik. Selain itu yang perlu disadari adalah adanya campuran huruf besar dan huruf dalam variabel nama daftarUrutan. Walaupun semua C++, kata kunci harus ditulis dalam huruf kecil, tetapi dalam penulisan tersebut dapat menggunakan huruf besar sebagai variabel nama.

Dalam bahasa pemrograman C maupun pada bahasa java identifier sifatnya case sensitive, artinya huruf besar dan huruf kecil dianggap berbeda artinya, walaupun dipernolehkan. Identifier terdiri dari :

- Karakter alphabet, yaitu : huruf 'A' sampai 'Z' dan huruf 'a' sampai 'z'
- Underscore (_) dan tanda dollar ($)
- Digit decimal, yaitu: bilangan antara '0' sampai '9'

Dalam menulis program ada hal-hal yang harus diperhatikan sebagai ketentuan dalam pemberian nama identifier. Ketentuan-ketentuan dalam penulisan identifier antara lain:

- Terdiri dari gabungan huruf dan angka dengan karakter pertama harus berupa huruf atau underscore atau tanda dollar ($).
- Tidak boleh mengandung spasi dan symbol-simbol khusus, kecuali garis bawah (underscore) atau tanda dollar ($) khusus untuk java. Yang termasuk symbol khusus yang tidak diperbolehkan antara lain: $, ?, %, #, !, &, *, (,), -, +, =, dan sebagainya.
- Panjangnya bebas, tetapi hanya 32 karakter pertama yang terpakai, tetapi sebaiknya sependek mungkin minimal satu karakter.
- Tidak boleh sama dengan kata kunci (keyword) dan kata tercadang (reserved word**)** yang ada dalam bahasa pemrograman.

Untuk lebih jelasnya mengenai identifier dapat dilihat pada contoh identifier yang benar dan yang salah

Tabel 3.1. Penulisan Identifier

PENULISAN	KETERANGAN IDENTIFIER
Nomor Nama Nilai_siswadata1 Rekam_1 _temp $harga_jual	Penulisan Benar
1data Nomor-siswa,nama siswa if 2_jalan 2$ main	Penulisan Salah

3.3. Konstanta

Suatu data yang sifatnya tetap, dan digunakan dalam pemrograman diistilahkan dengan konstanta. Konstanta adalah sebuah nama tempat penyimpanan sementara di dalam memori yang nilainya tetap atau tidak dapat diubah. Konstanta harus didefinisikan terlebih dahulu pada awal program. Konstanta dapat bernilai integer, pecahan, karakter atau string. Perhatikan contoh sebuah konstanta berikut ini:

```
50
3.14
'A'
"Bahasa Pemrograman"
```

Dalam melakukan pemrograman ada dua tipe konstanta yaitu konstanta bilangan dan konstanta teks atau string.

3.3.1. Konstanta Bilangan

Konstanta bilangan atau sering juga disebut dengan konstanta numeric yaitu suatu konstanta yang nilai tetapnya berupa bilangan. Konstanta bilangan atau numerik dibagi dalam dua kelompok, yaitu :

a. Konstanta bilangan bulat (integer)

Konstanta bilangan bulat merupakan sebuah konstanta yang nilai tetapnya berupa bilangan bulat. Konstanta bilangan bulat dapat disajikan dalam bentuk decimal, oktal ataupun heksadesimal. Dalam sistem decimal, bilangan yang digunakan berkisar antara 0 sampai 9. Penulisan konstanta langsung angkanya, misalnya: 10, 9, 5 dan sebagainya.

Dalam sistem octal, bilangan yang digunakan berkisar antara 0 sampai dengan 7. Penulisan konstanta diawali dengan 0 misalnya: 010, 046, 055 dan sebagainya. Dalam sistem heksadesimal, bilangan yang digunakan berkisar antara 0 sampai dengan F. Bilangan yang dapat dipakai berupa salah satu diantara 16 simbol adalah berikut:

```
0 1 2 3 4 5 6 7 8 9 A B C D E F
```

Bisa juga ditulis sebagai berikut:

```
0 1 2 3 4 5 6 7 8 9 a b c d e f.
```

Cara penulisan konstanta bilangan ini diawali dengan 0x (nol dan x), misalnya: 0xAF, 0x7F, dan sebagainya.

b. konstanta bilangan real atau pecahan

Konstanta dengan nilai tetapnya berupa bilangan pecahan. Konstanta bilangan real atau pecahan dibagi dua jenis, yaitu: Konstanta data bilangan desimal berpresisi tunggal (floating point), dimana bilangan ini memiliki derajat ketelitian sampai 7 digit dan dapat dinyatakan dalam dua bentuk tampilan, yaitu :

- Bentuk desimal, contoh : 21.333
- Bentuk eksponensial, dituliskan dengan notasi *scientific*. Dengan bentuk umum: bulat pecahan{E|D} {[+] | [-]} pangkat misalnya: 0.21333E+2 dimana hal penulisan ini mempunyai arti: 0.2133 x 10^2

Sedangkan konstanta data bilangan desimal berpresisi ganda (double precision) serupa dengan floating point, hanya derajat ketelitian

dan range jangkauan yang dimiliki lebih tinggi. Derajat ketelitian untuk floating point 7 digit, sedangkan double precision 16 digit.

3.3.2. Konstanta Teks/String.

Konstanta teks atau string adalah suatu konstanta yang nilai tetapnya berupa teks. Konstanta teks dibedakan dalam dua jenis, yaitu:

a. Konstanta data karakter

Konstanta data karakter terdiri dari sebuah karakter saja dan ditandai dengan dua tanda kutip tunggal (‘.’) sebagai pembatasnya. misalnya: ‘A’, ‘b’, ‘&’, ‘!’. Selain itu juga ada beberapa diantara konstanta data karakter yang ditulis dengan diawali tanda \ (penempatan tanda \ setelah tanda petik tunggal pembuka).

Karakter ini dinamakan rangkaian escape (escape sequence), sehingga disebut dengan konstanta karakter rangkaian escape. Beberapa karakter rangkaian escape, antara lain :

Tabel 3.2. karakter rangkaian ESCAPE

KARAKTER	KETERANGAN
\a	Untuk bunyi bell (alert)
\b	Mundur satu spasi (backspace)
\f	Ganti halaman (form feed)
\n	Ganti baris baru (new line)
\r	Ke kolom pertama, baris yang sama
\v	Tabulasi vertical
\0	Nilai kosong (null)
\’	Karakter petik tunggal
\”	Karakter petik ganda
\\	Karakter garis miring

Untuk menyatakan sembarang karakter ASCII, notasi yang dapat dipergunakan, antara lain: \DDD dengan DDD = 1 sampai dengan 3 digit octal atau \xHH dengan HH = 1 sampai dengan 2 digit heksadesimal, misalnya: ‘\”’ atau ‘\42’ atau ‘\x22’. Selain karakter biasa, juga terdapat karakter khusus yang didefinisikan dengan cara mengawalinya menggunakan tanda \ seperti pada dijelaskan pada tabel dibawah ini:

Tabel 3.3. karakter khusus

KODE	NAMA	NILAI UNICODE
\t	Tab	\u0009
\b	Backspace	\u0008
\n	Linefeed	\u000a
\r	Carriage return	\u000d
\’	Single quote	\u0027
*	Double quote	\u0022
\\	Backslash	\u005c

b. Konstanta data string.

Konstanta data string merupakan rangkaian dari beberapa karakter dan ditandai dengan dua tanda kutip ganda (“ ”) sebagai pembatasnya. Perhatikan contoh penulisan dibawah ini:

```
“Helo”, “Lagi Belajar”
“Bahasa Pemrograman”, “Di Sekolahmu ya?”
```

Dalam penulisan program, sebuah konstanta perlu dideklarasikan, dimana deklarasi konstanta merupakan proses untuk menyebutkan karakteristik konstanta seperti nama, tipe data, jangkauan, masa hidup, dan nilai awal. Dalam bahasa C pendeklarasikan konstanta ada 2 cara, yaitu: dengan menggunakan pengarah compiler atau preprocessor `#define`, dan dengan keyword `const`. penulisan sintaks dari deklarasi konstanta dalam bahasa C adalah sebagai berikut:

```
#define <namakonstanta> <nilai>
```

Atau dapat juga ditulis sebagai berikut:

```
const <tipedata> <namakonstanta> = <nilai>;
```

Untuk lebih jelasnya pehatikan contoh program dibawah ini:

```
#define nomor 100
#define phi 3.14
#define huruf ‘K’
#define nama “Hello”
```

Atau bisa juga ditulis seperti program dibawah ini:

```
const int nomor = 100;
const float phi = 3.14;
const char huruf = ‘K’;
const String nama=“Hello”;
```

Program 3.1. Penggunaan konstanta dalam program

```
#include <conio.h>
#include <iostream>
#define nomor 100
```

```
#define phi 3.14
#define huruf 'K'
#define nama "Hello"

using namespace std;

main()
{
	cout << "Nilai konstanta nomor: "<< nomor;
	cout << "\nNilai konstanta phi  : "<< phi;
	cout << "\nNilai konstanta huruf: "<< huruf;
	cout << "\nNilai konstanta nama : "<< nama;
	getch();
	return 0 ;
}
```

Keluaran programnya adalah sebagai berikut:

```
Nilai konstanta nomor: 100
Nilai konstanta phi  : 3.14
Nilai konstanta huruf: K
Nilai konstanta nama : Hello
```

Pada program dibawah ini hasil setelah decompile akan sama dengan program diatas. Program dibawah merupakan salah satu contoh penulisan yang berbeda tetapi akan menghasilkan hasil yang sama dengan program sebelumnya.

Perhatikan pada penggunaan konstanta dalam aplikasi program dibawah ini:

Program 3.2

```
#include <conio.h>
#include <iostream>
const int nomor = 100;
const float phi = 3.14;
const char huruf = 'K';
const char nama[] = "Hello";

using namespace std;

main()
{
		cout << "Nilai konstanta nomor: "<< nomor;
		cout << "\nNilai konstanta phi  : "<< phi;
		cout << "\nNilai konstanta huruf: "<< huruf;
		cout << "\nNilai konstanta nama : "<< nama;
```

```
        getch();
        return 0 ;
}
```

Keluaran program diatas adalah sebagai berikut:

```
Nilai konstanta nomor: 100
Nilai konstanta phi  : 3.14
Nilai konstanta huruf: K
Nilai konstanta nama : Hello
```

3.4. Variabel

Variabel atau pengubah adalah suatu lambang dari sebuah lokasi yang berada dimemori utama komputer yang dapat berisi suatu nilai. Nilai yang berada di lokasi memori tersebut dapat berubah selama program dieksekusi. Guna variable adalah sebagai tempat sementara untuk menyimpan data yang sedang diolah. Pemberian nilai kedalam suatu variabel mempunyai bentuk penulisan yang berbeda-beda pada setiap bahasa pemrograman. Meskipun mempunyai arti yang sama dalam pemrogramannya.

Variabel dibedakan atas variabel numerik dan variabel string. Variabel numerik adalah variabel yang mengandung nilai numerik atau angka sedangkan variabel string adalah variabel yang berisi nilai huruf /alpha-numerik.

Suatu variabel dapat mewakili:

- Nilai Konstanta

```
double pecahan = 13.45;
int nilai = 85;
```

- Nilai dari pengubah lain

```
char abjad = "T" ;
char huruf = abjad;
```

- Nilai yang diperoleh dari kombinasi beberapa pengubah atau nilai konstanta dengan melalui operator.

```
double pi = 3.141593;
double r;
double l_lingkaran = pi * r * r;
```

3.4.1. Tipe Variabel

Berdasarkan pada jenis data yang disimpan pada variabel yaitu: Variabel numeric dan variable teks. Pada variable numeric merupakan suatu variabel yang dipakai untuk menampung data berupa angka.

Variabel numerik digolongkan atas: bilangan bulat atau integer dan bilangan real atau pecahan, dimana bilangan real terdiri dari dua yaitu:

- bilangan decimal berpresisi tunggal (floating point)
- bilangan decimal berpresisi ganda (double precision).

Sedangkan pada variable teks merupakan suatu variabel yang dipergunakan untuk menampung data berupa huruf atau teks. Variabel teks dibedakan menjadi dua yaitu:

- Karakter (untuk karakter tunggal)
- String (untuk rangkaian karakter)

3.4.2. Jenis Variabel

Berdasarkan pada lokasi atau wilayah dan waktu pengaksesannya, variabel dapat dibedakan atas 2 jenis, yaitu: Variabel Lokal dan Variabel Global. Variabel local merupakan Variabel yang dideklarasikan pada fungsi termasuk fungsi utama (main), variabel dialokasikan selama fungsi pendeklarasi dijalankan dan Variabel didealokasikan jika fungsi pendeklarasi tidak dijalankan.

Sedangkan Variabel Global adalah Variabel yang dideklarasikan sebelum definisi fungsi termasuk funsi utama, Variabel dialokasikan selama program dijalankan dan Variabel didealokasikan setelah program selesai dijalankan

3.4.3. Deklarasi Variabel

Deklarasi variable adalah proses untuk menyebutkan karakteristik variable seperti nama, tipe data, jangkauan, masa hidup, dan nilai awal. Penulisan Sintaks untuk mendeklarasikan sebuah variable adalah sebagai berikut:

```
<tipedata> <namavariabel> [= nilaiawal];
<tipedata> <namavariabel 1> [= nilaiawal], …, <namavariabel n> [= nilaiawal];
```

Dalam penulisan sintak tersebut yang perlu diperhatikan adalah bahwa semua yang ada dalam tanda <…> harus ada dan semua yang ada dalam tanda […] bersifat tambahan boleh ada atau boleh tidak tergantung keperluan. Perhatikan contoh penulisan dibawah ini:

```
int bilangan;
char karakter;
float bildesimal;
```

Setelah mendeklarasikan sebuah variabel dengan tipe data, selanjutnya memberikan nilai variabel tersebut dengan tanda = (sama dengan)

```
bilangan = 20;
karakter = 'k';
bildesimal = 22.2;
```

atau dapat juga mendeklarasikan dan memberikan nilai dalam satu baris.

```
int bilangan = 20;
char karakter = 'k';
char huruf, huruf1 = 'C', huruf2 = 'O';
float bildesimal = 22.2;
```

3.4.4. Kelas Variabel

Selain digolongkan berdasarkan tipe datanya, sebuah variabel juga diklasifikasikan berdasarkan kelas penyimpanan, yaitu: Variable local atau otomatis, variable eksternal, variable statis dan variable register

Variable local adalah merupakan variabel yang dideklarasikan dalam sebuah fungsi dan mempunyai sifat-sifat sebagai berikut:

- Secara otomatis diciptakan ketika fungsi dipanggil dan akan lenyap ketika eksekusi terhadap fungsi berakhir
- Hanya dikenal oleh fungsi tempat variable dideklarasikan
- Tidak ada inisialisasi secara otomatis (saat variable diciptakan nilainya tak menentu)

- Deklarasinya dengan kata kunci auto.

```
auto int x;
```

Perhatikan penulisan variable local berikut ini:

Untuk lebih jelasnya perhatikan program dengan menggunakan variable local berikut ini:

Program 3.3

```
#include <conio.h>
#include <iostream>

using namespace std;

void tampil(void);

main()
{
        auto int i = 11;
        tampil();
        cout <<"\ni dalam main() = "<< i;
        getch();
        return 0 ;
}

void tampil(void)
{
auto int i = 10;
        cout <<"\ni dalam tampil() = "<< i;
}
```

Keluaran program diatas adalah sebagai berikut:

```
i dalam tampil() = 10
i dalam main() = 11
```

selain variable local seperti dijelaskan diatas, juga terdapat kelas variable eksternal dimana variable ini mendeklarasikan diluar sebuah fungsi, dan mempunyai sifat berikut:

- Dapat diakses oleh semua fungsi
- Kalau tidak diberi nilai, secara otomatis diinisialisasi dengan nilai sama dengan nol
- Deklarasinya dengan kata kunci extern didalam fungsi yang menggunakan variable jenis ini.

Perhatikan penulisan variable eksternal berikut ini:

```
extern int x;
```

untuk lebih jelasnya perhatikan program dengan menggunakan variable eksternal berikut ini:

Program 3.4

```
#include <conio.h>
#include <iostream>

using namespace std;

/* variabel j disini merupakan variabel eksternal bagi
fungsi kali dan fungsi main, sehingga variabel j dapat
diakses oleh semua fungsi yang ada yaitu kali dan main */

int j = 1;
void kali(void);

main()
{
        cout <<"Nilai awal j     = "<< j<<"\n";
        j = j * 5;
        cout <<"Nilai j kemudian = "<< j<<"\n";
        kali();
        cout <<"Nilai j kini     = "<< j<<"\n";
        kali();
        cout <<"Nilai j sekarang = "<< j<<"\n";
  getch();
        return 0 ;
}

void kali(void)
{
   j*=10;
}
```

Keluaran program diatas adalah sebagai berikut:

```
Nilai awal j       = 1
Nilai j kemudian = 5
Nilai j kini       = 50
Nilai j sekarang  = 500
```

Variable statis merupakan sebuah variable internal yang didefinisikan didalam fungsi maupun variable eksternal, dan variable ini mempunyai sifat-sifat sebagai berikut:

- Kalau bersifat internal, maka hanya dikenal oleh fungsi tempat variable dideklarasikan
- Kalau bersifat eksternal, maka dapat dipergunakan oleh semua fungsi yang terletak pada file yang sama tempat variable dideklarasikan
- Tidak hilang sekeluarnya dari fungsi
- Inisialisasi hanya dilakukan sekali pada saat fungsi dipanggil pertama kali, jika tidak ada inisialisasi oleh pemrogram secara otomatis akan diberi nilai awal nol.
- Deklarasinya dengan kata kunci static.

Perhatikan penulisan variable statis berikut ini:

```
static int x;
```

Untuk lebih jelasnya perhatikan program dengan menggunakan variable statis berikut ini:

Program 3.5

```
#include <conio.h>
#include <iostream>

using namespace std;

void tambah(void);

main()
{
    int k = 100;
    tambah();
    tambah();
    cout<<"Nilai k dalam main() = "<< k<<"\n";
    getch();
          return 0 ;
}

void tambah(void)
{
   static int k;       //variabel statis
   k++;
   cout <<"Nilai k dalam tambah() = "<< k<<"\n";
}
```

Keluaran program diatas adalah sebagai berikut:

```
Nilai k dalam tambah ()          = 1
Nilai k dalam tambah ()          = 2
Nilai k dalam main ()            = 100
```

Variable register adalah variable yang nilainya disimpan dalam register mikroprosesor bukan didalam memori RAM, variable ini mempunyai sifat-sifat sebagai berikut:

- Mempunyai kecepatan akses lebih tinggi
- Hanya dapat diterapkan pada variable local atau parameter formal yang bertipe char atau int
- Biasa diterapkan untuk pengendalian loop.
- Deklarasinya dengan kata kunci register.

Perhatikan penulisan variable register berikut ini:

```
register int x;
```

untuk lebih jelasnya perhatikan program dengan menggunakan variable register berikut ini:

Program 3.6

```
#include <conio.h>
#include <iostream>

using namespace std;

main()
{
        register int n, jml;//variabel register
        int m = 242;
        for(n=1; n<=m; n++)
        jml += n;
        cout <<" 1 + 2 + 3 + ... +  = " << m<< jml;
        getch();
        return 0 ;
}
```

Keluaran program diatas adalah sebagai berikut:

```
1 + 2 + 3 + … + 242 =32621
```

3.5. Tipe Data

Tipe data merupakan tempat untuk menentukan pemberian nilai terhadap suatu variabel yang diberikan oleh user. Selain itu tipe data juga dapat diartikan sebagai batasan terhadap fungsi tanda pengenal terhadap semua nilai yang diterima. Sebagai gambaran dari pengertian tersebut adalah ketika kita menempatkan tanda pengenal harga hanya mengenal angka, maka ketika kita memberikan nilai berupa string maka secara otomatis data tersebut akan ditolak karena nilai tersebut tidak dikenali oleh tipe data yang diberikan.

Tipe data dalam variable menentukan tipe data yang bisa disimpan didalamnya, format data

yang disimpan, dan berapa banyak memori yang dialokasikan untuk menyimpan data.

Tipe data tersebut diklasifikasikan berdasarkan bagaimana keadaan data disimpan dalam memori, dan jenis operasi yang dapat dilakukan. Tipe data merupakan bagian program yang paling penting karena tipe data mempengaruhi setiap instruksi yang akan dilaksanakan oleh komputer. Misalnya saja 3 dibagi 2 bisa saja menghasilkan hasil yang berbeda tergantung tipe datanya. Jika 3 dan 2 bertipe integer, maka program akan menghasilkan nilai 1, namun jika keduanya bertipe float maka akan menghasilkan nilai 1.5000000. Pemilihan tipe data yang tepat akan membuat proses operasi data menjadi lebih efisien dan efektif. Fungsi Tipe data antara lain: Merepresentasikan nilai dari suatu variabel maupun konstanta, Penyimpanan data di memori, dan Menentukan nilai yang dapat diisikan ke dalam sebuah variable.

Sedangkan jenis tipe data ada dua macam: Tipe dasar atau primitive dan Tipe bentukan atau referensi. Dalam bahasa C terdapat lima tipe data dasar, yaitu: int, char, float, double, dan void. Sedangkan dalam bahasa pemrograman lain seperti java misalnya memiliki tipe data yang dapat dikategorikan menjadi dua kelompok, yaitu tipe data primitif dan referensi. Ada delapan macam tipe data primitif dalam pemrograman Java, yaitu : char, byte, short, int, long, float, double, dan Boolean. Tipe data referensi digunakan untuk mereferensikan objek atau class tertentu, seperti String.

Tabel 3.4. Tipe Data (16 bit) Dalam C

TIPE	UKURAN	RANGE
unsigned char	8 bit	0 - 255
char	8 bit	-128 -127
enum	16 bit	-32768 - 32767
unsigned int	16 bit	0 - 65535
short int	16 bit	-32768 - 32767
int	16 bit	-32768 - 32767
unsigned long	32 bit	0 - 4294967295
long	32 bit	-2147483648 - 2147483647
float	32 bit	3.4E-38 - 3.4E+38
double	64 bit	1.7E-308 - 1.7E+308
long double	80 bit	3.4E-4932 - 1.1E+4932
void	0 bit	
near (pointer)	16 bit	
far (pointer)	32 bit	

Tabel 3.5. Tipe Data (32 bit) Dalam C

Tipe	Ukuran	Range
unsigned char	8 bit	0 - 255
char	8 bit	-128 -127
short int	16 bit	-32768 - 32767
unsigned int	32 bit	0 - 4294967295
int	32 bit	-2147483648 - 2147483647
unsigned long	32 bit	0 - 4294967295
enum	32 bit	-2147483648 - 2147483647
long	32 bit	-2147483648 - 2147483647
float	32 bit	3.4E-38 - 3.4E+38
double	64 bit	1.7E-308 - 1.7E+308
long double	80 bit	3.4E-4932 - 1.1E+4932
void	0 bit	
near (pointer)	16 bit	
far (pointer)	32 bit	

Tabel 3.6. Format Tipe Data Dalam C

TIPE	FORMAT	KETERANGAN
char	%c	Karakter/string
int	%i, %d	Integer/bilangan bulat
float	%f	Float/bilangan pecahan presisi tunggal
double	%lf	Bilangan pecahan presisi ganda

3.5.1. Tipe data Integer

Program Komputer merupakan kumpulan potongan data dan memanipulasi data tersebut dalam berbagai cara. Ada berbagai jenis data dalam bidang informasi, misalnya: bilangan, terdapat bilangan bulat dan bilangan pecahan, ada bilangan positif dan negative, dan ada pula bilangan besar serta bilangan kecil dan lain-lain, atau bahkan bilangan yang tidak memiliki nama.

Kemudian ada juga informasi tekstual, misalnya Nama dan alamat, maka data seperti ini akan disimpan sebagai karakter. Bila programmer menulis sebuah program, maka harus menentukan jenis informasi yang digunakannya. Jika menulis sebuah program untuk menentukan jauhnya berapa mil dari bumi ke bintang, maka diperlukan variabel yang dapat menampung jumlah yang sangat besar. Jika Anda sedang merancang perangkat lunak untuk merekam sebuah dimensi mikroskopis, maka hanya perlu untuk

menyimpan data yang sangat kecil dan angka sesuai. Sebagai tambahan, jika Anda menulis sebuah program yang harus melakukan ribuan perhitungan intensif, maka akan menginginkan variabel yang dapat diproses dengan cepat. Tipe Data variabel inilah yang akan menentukan semua factor tersebut.

Bahasa C++ menyediakan berbagai tipe data, yang secara garis besar hanya dibagi menjadi dua yaitu: data numerik dan karakter. Numerik merupakan tipe data yang dibagi menjadi dua kategori yaitu: integer dan floating-point. Integers adalah bilangan yang meliputi seluruh nomor seperti 12, 157, -34, 2 daln lain sebagainya. Floating-point merupakan angka yang ada angka decimal di belakang koma, misalnya 23,7, 189,0231, 0,987 dan lain sebagainya.

Sebelum kita membicarakan tipe data karakter, mari kita hati-hati memeriksa variasi dari data numerik. Pertimbangan utama untuk memilih tipe data numerik adalah:

- Angka Terbesar dan terkecil yang dapat disimpan dalam variable.
- Berapa banyak memori yang digunakan oleh variabel.
- Apakah variabel menangani penyimpanan (baik positif maupun negatif) atau bilangan unsigned (hanya positif).
- Bilangan decimal merupakan variabel yang presisi

Ukuran variabel adalah jumlah byte memori yang digunakan. Biasanya, semakin besar range variable, maka jumlah yang digunakan akan lebih besar pula.

3.5.2. Tipe Data Karakter

Karakter adalah sembarang huruf, angka, atau tanda baca tunggal. Tipe data karakter merupakan kumpulan bermacam-macam karakter yang terdiri dari alfabet. Diman karakter antara lain: Alfabet bilangan decimal: 0, 1, 2, …, 9, Alfabet huruf latin besar : A, B, C, …, Z, Alfabet huruf latin kecil : a, b, c, ... , z dan Tanda baca tunggal : !, @, ~, ?, ;, ‘, &, dan sebagainya.

Tipe data karakter ini hanya terdiri dari 1 karakter dan Bentuk tipe data dari karakter yaitu: char. Perhatikan penulisan contoh deklarasi char, dimana char huruf yang ditampilkan adalah = ‘A’ ;

Program 3.7

```
include <iostream>

using namespace std;

main()
{
        char huruf_1 = 'C', huruf_2 = '+';
        cout <<"Tipe Data Char pada " << huruf_1 << huruf_2<< huruf_2;
        getch();
        return 0 ;
}
```

Keluaran program diatas adalah sebagai berikut:

```
Tipe Data Char pada C++
```

3.5.3. Tipe Data String

String adalah deretan karakter yang diakhiri dengan sebuah karakter kosong. Konstanta bertipe ditulis diantara tanda petik dua (“ ….”). Dalam bahasa C string merupakan larik atau array dari tipe data char, sedangkan dalam bahasa java string merupakan tipe data referensi atau sebuah objek. Contoh dibawah merupakan deklarasi string dalam C:

```
char tek [ ] = " C++ " ;
char kata[ ] = {'C', '+', '+'};
```

untuk Contoh penulisan deklarasi string dalam C adalah sebagai berikut:

Program 3.8

```
#include <conio.h>
#include <iostream>

using namespace std;

main()
{
      char huruf[] = "Tipe Data String pada  C++";
      cout << huruf;
      getch();
      return 0 ;
}
```

Keluaran program diatas adalah sebagai berikut:

```
Tipe Data String pada C++
```

3.5.4. Tipe Data Bilangan Bulat

Tipe data ini digunakan untuk data-data angka yang tidak mengandung angka di belakang koma (int) atau digunakan untuk menyatakan bilangan bulat. Perubahan tanda bilangan pada bilangan bulat dapat diset dalam dua tipe, yaitu: bilangan bulat bertanda (signed integer), yang merupakan bilangan bulat yang memiliki range dari bilangan negatip sampai positip dan bilangan bulat tak bertanda (unsigned integer), yang merupakan bilangan bulat yang hanya memiliki range nilai positip saja.

Tipe data yang termasuk ke dalam bilangan bulat adalah: yang pertama, char atau signed char dan unsigned char atau byte dalam java dan pascal. Rentang nilai signed char mulai -128 sampai 127. Kedua,

rentang nilai unsigned char mulai 0 sampai 255, short int atau signed short int dan unsigned short int. Rentang nilai signed short int mulai -32.768 sampai 32.767. Rentang nilai unsigned short int mulai 0 sampai 65.535. Ketiga adalah int atau signed int dan unsigned int. Rentang nilai signed int mulai -32.768 sampai 32.767.

Rentang nilai unsigned int mulai 0 sampai 65.535, dan keempat adalah long int atau signed long int dan unsigned long int. Rentang nilai signed long int mulai -21474836478 sampai 2147483647. Rentang nilai unsigned long int mulai 0 sampai 4294967295. Rentang di atas untuk tipe data bilangan bulat dalam 16 bit.

Contoh pendeklarasian bilangan int adalah sebagai berikut:

```
int nilai;
int total ;
int harga = 30000;
```

Dapat juga ditulis sebagai berikut:

```
int nilai, total ;
int harga;
harga = 30000;
```

3.5.5. Tipe Data Bilangan Real atau Pecahan

Tipe ini merepresentasikan data-data bilangan yang mengandung angka di belakang koma atau menyatakan bilangan pecahan, maupun eksponensial. Tipe data yang termasuk ke dalam kategori ini adalah: float dan double. Contoh deklarasi float dan double adalah sebagai berikut:

```
float nilai;
double beta;
```

Semua bilangan pecahan atau desimal dalam Java tanpa diakhiri huruf **f** akan dianggap sebagai double. Sedangkan bilangan yang ingin dikategorikan sebagai float harus diakhiri dengan huruf **F**. Misalnya : 4.22F atau 2.314f. Sedangkan untuk bilangan double, bisa menambah dengan huruf D, karena secara default bilangan dengan koma atau pecahan atau desimal akan dianggap sebagai double. Pehatikan contoh program berikut ini:

Program 3.9

```
#include <conio.h>
#include <iostream>

using namespace std;

main()
{
```

```
    short int ssintmin = -32768, ssintmak = 32767;
    unsigned short int usintmak = 65535;
    int intmin = -32768, intmak = 32767;
    unsigned int uintmak = 65535;
    long int slintmin = -2147483648, slintmak = 2147483647;
    unsigned long int ulintmak = 4294967295;
    cout <<"\nRange signed short int      :"<< ssintmin<< ssintmak;
    cout <<"\nRange unsigned short int  :"<< usintmak;
    cout <<"\nRange signed int              : "<< intmin<<  intmak;
    cout <<"\nRange unsigned int          :"<< uintmak;
    cout <<"\nRange signed long int       :"<< slintmin<< slintmak;
    cout <<"\nRange unsigned long int      :"<< ulintmak;
    getch();
    return 0 ;
}
```

Keluaran program diatas adalah sebagai berikut:

```
Range signed short int        :  -32768        s/d         32767
Range unsigned short int      :         0        s/d         65535
Range signed int              :  -32768        s/d         32767
Range unsigned int            :         0        s/d        65535
Range signed long int         : -2147483648 s/d  2147483647
Range signed long int         :         0        s/d  4294967295
```

Program 3.10 Perhatikan juga contoh program dibawah ini:

```
#include <conio.h>
#include <iostream>

using namespace std;

main()
{
        float floatmin = 3.4E-38, floatmak = 3.4E+38;
        double doublemin = 1.7E-308, doublemak = 1.7E+308;
        long double ldoublemin = 3.4E-4932 , ldoublemak = 1.1E+4932;
        cout <<"Range float      : \n"<< floatmin<< floatmak;
        cout <<"Range double     : \n"<< doublemin<< doublemak;
        cout <<"Range long double : \n"<< ldoublemin<<          ldoublemak;
        getch();
        return 0 ;
}
```

Keluaran program adalah sebagai berikut:

```
Range float       : 0.000000 s/d 3399999952144364250000000000000.0
Range double      : 0.000000E+00 s/d 1.700000E=300
Range long doble  : 0 s/d 1.1E+4932
```

3.5.6. Tipe Data Logika

Tipe data logika digunakan untuk merepresentasikan data-data yang mengandung dua buah nilai logika boolean, yaitu: nilai 1 dan 0 atau sering juga disebut sebagai nilai true dan false (benar dan salah). Dalam bahasa C++ tidak ada data tipe boolean. Perhatikan contoh deklarasi Boolean berikut ini:

```
Boolean kondisi;
Boolean kondisi_awal = true;
```

Pehatikan contoh program berikut ini:

Program 3.11

```
include <conio.h>
#include <iostream>

using namespace std;

main()
{
        enum boolean {FALSE, TRUE};
        enum boolean kondisi;
        kondisi = TRUE;
        if (kondisi)
        cout <<"Kondisi : "<< kondisi;
        else
        {
           kondisi = FALSE;
           cout <<"Kondisi : "<< kondisi;
        }
  getch();
        return 0 ;
}
```

Keluaran program adalah sebagai berikut:

```
Kondisi : 1
```

3.5.7. Enumerasi / ENUM

Enumerasi adalah serangkaian symbol berurutan yang menspesifikasikan konstanta bertipe integer. Dalam C++ tidak terdapat tipe Bolean, sehingga untuk merepresentasikan TRUE dengan nilai integer bukan nol (1, 2, dst), sedangkan FALSE dengan nilai nol

(0). Perhatikan contoh deklarasi enum:

```
enum boolean { false, true } ; atau
enum boolean { salah = 0, benar = 1 } ;
```

3.5.8. Tipe Data Void

Void menyatakan tipe kosong dan digunakan untuk: pertaman untuk mendeklarasikan fungsi yang tidak mengembalikan nilai apapun. Serta fungsi kedua adalah untuk mendeklarasikan fungsi yang tidak menerima parameter apapun. Pada tipe data ini jika diawali dengan operator *, menyatakan penunjuk terhadap sembarang tipe data. Perhatikan contoh deklarasi void seperti berikut ini:

```
void cctrputs (char*,int );
```

atau ditulis:

```
main (void) ;
```

atau dapat juga ditulis

```
void* action ;
int ivalue = 100 ;
action = &ivalue ;
```

3.5.9. Tipe Data Pointer

Pointer adalah variable yang berisi nilai alamat suatu lokasi memori tertentu. Deklarasi penunjuk dilakukan dengan menspesifikasikan *, sebelum nama varibel/konstanta. Contoh deklarasi pointer adalah sebagai berikut:

```
char *p;
```

untuk lebih jelasnya mengenai tipe data pointer akan dijelaskan dalam bab pointer

3.5.10. Tipe Data Larik / array

Array adalah sekelompok data bertipe sama yang menduduki lokasi memori yang berurutan. Jumlah elemen array dinyatakan dengan cara mengapit jumlah yang di maksud dengan tanda ‘ […] ‘. Cara penulisan sintaks tipe data larik adalah sebagai berikut :

```
<tipedata> <namavariabelArray> <[
jumlahelemen ]> ;
```

Misalnya penulisan tipe char dengan array adalah sebagai berikut:

```
char kata[5];
```

Untuk menyatakan array berdimensi lebih dari 1 (satu), maka ditambahkan tanda ‘[…]’ sebanyak dimensi yang diinginkan. Misalnya jika ingin mendeklarasikan array 2 dimensi adalah sebagai berikut:

```
int matrix [2][3] ;
```

Untuk lebih jelasnya mengenai tipe data array akan dijelaskan dalam bab selanjutnya yang membahas array.

3.5.11. Tipe Data Struct, Union

Tipe data ini digunakan untuk mendeklarasikan sekelompok data yang memiliki tipe yang berlainan. **struct**: elemennya berada dilokasi memori yang berbeda, dan **union**: elemennya ada dilokasi memori yang sama. Perhatikan potongan program dibawah ini:

```
union namaunion
{
      Tipeanggota1 namaAnggota1 ;
      Tipeanggota2 namaAnggota2 ;
      ……
};
```

Atau dapat juga ditulis seperti dibawah ini:

```
union
{
      Tipeanggota1 namaAnggota1 ;
      Tipeanggota2 namaAnggota2 ;
      ……
}
namaunion;

struct  namaStruktur
{
      Tipeanggota1 namaAnggota1 ;
      Tipeanggota2 namaAnggota2 ;
      ……
};
```

Atau dapat juga ditulis

```
struct
{
      Tipeanggota1 namaAnggota1 ;
      Tipeanggota2 namaAnggota2 ;
      ……
}
namaStruktur ;
```

3.5.12. Tipe Data typedef

Tipe data typedef digunakan untuk menamakan suatu tipe data dengan pengenal yang lebih berarti atau mudah diingat. Sintaks penulisan program tipe data ini adalah sebagai berikut:

```
typedef tipedata namabaru;
```

Perhatikan contoh penulisan program dibawah ini :

```
typedef unsigned char byte;
```

3.6. Operator Bahasa C++

Bahasa C++ menyediakan beberapa operator untuk memanipulasi data. Secara umum, terdapat tiga jenis operator: unary, binary dan ternary. Istilah tersebut mencerminkan jumlah operands operator yang dibutuhkan.

Operator unary hanya memerlukan satu operand. Misalnya, mempertimbangkan ekspresi berikut: -5. Dalam contoh diatas perlu dipahami bahwa mewakili angka lima bernilai negatif. Konstanta 5 adalah diawali dengan tanda minus. Tanda minus, bila digunakan dengan cara seperti ini, yang disebut penyangkalan operator. Karena hanya memerlukan satu operand, hal tersebu merupakan operator unary. Operator binary bekerja dengan dua operand.

Tugas operator ini biasanya pada operasi aritmetik yang hal tersebut sudah sangat umum dalam bahasa pemrograman. Tabel dibawah menunjukkan operator arithmetic pada bahasa C++.

Tabel 3.6. Operator pada bahasa C++

OPERATOR	ARTI	TIPE	CONTOH
*	perkalian	binary	tax = cost * rate;
/	Pembagian	Binary	salePrice = original / 2;
%	Modulus	Binary	remainder = value % 3;
+	Penjumlahan	Binary	total = cost + tax;
-	Pengurangan	binary	cost = total - tax;

Kebanyakan sebagai seorang programmer tidak akan asing dan selalu bekerja dengan operator. Operator Penambahan digunakan untuk menjumlahkan dua operand. Pada pernyataan berikut ini merupakan fungsi variabel **jumlah** yang diberi dengan nilai 12 adalah:

```
jumlah = 4 + 8;
```

operator pengurangan merupakan operasi dimana nilai penguranga berasal dari operand sebelah kiri dikurangi atau disubtracted operand sebelah kanan. Pernyataan dibawah ini akan memperoleh nilai 98 untuk suhu:

```
suhu = 112 - 14;
```

Pada sebuah operator perkalian akan mengalikan dua buah operand. Perhatian pernyataan berikut ini,dimana markup akan diisi nilai 3:

```
markup = 12 * 0,25;
```

pada operator pembagian hasil diperoleh dengan cara memlakukn bagi operand disebelah kiri dibagi dengan operand sebelah kanan. Dalam pernyataan berikutnya, variable poin akan memperoleh nilai 5:

```
poin = 100 / 20;
```

Dalam operasi pembagian integer ternyata tidak selalu memasukan hasil dari operasi pembagian tetapi yang diambil adalah sisa dri hasil bagi tersebut. Untuk lebih jelasnya perhatikan pernyataan berikut ini, dimana hasilnya adalah 2 atau nilai sisa sama dengan 2:

```
sisa = 17% 3;
```

Program 3.12 Perhatikan contoh program dibawah ini:

```
#include <conio.h>
#include <iostream>

using namespace std;
```

```
int main()
{
        double regHours = 40.0,
        otHours = 10,
        regPayRate = 18.25,
        otPayRate = 27.78,
        regWages,
        otWages,
        totalWages;
        regWages = regPayRate * regHours;
        otWages = otPayRate * otHours;
        totalWages = regWages + otWages;
        cout << "Upah untuk minggu ini adalah Rp. " << totalWages << endl;
        getch();
       return 0 ;
}
```

Keluaran Program diatas adalah

```
Upah untuk minggu ini adalah Rp. 1007.8
```

3.7. Operator Unary

Operator Unary merupakan operator yang hanya memiliki atau melibatkan sebuah operand saja. Terdapat beberapa operator unary, seperti terlihat pada tabel berikut ini:

Tabel 3.7. Operator Unary

OPERATOR	ARTI	LETAK	CONTOH	EQUIVALEN
-	Unary minus	Sebelum operator	A + -B * C	A + (-B) * C
++	Peningkatan dengan Penambahan nilai 1	Sebelum dan sesudah	A++	A = A + 1
--	Penurunan dengan Pengurang-an nilai 1	Sebelum dan sesudah	A--	A = A – 1
sizeof	Ukuran dari operand dalam byte	Sebelum	sizeof(I)	-
!	Unary NOT	Sebelum	!A	-
~	Bitwise NOT	Sebelum	~A	-
&	Menghasilkan alamat memori operand	Sebelum	&A	-
*	Menghasil-kan nilai dari pointer	Sebelum	*A	-

Dalam operator unary ada yang perlu diperhatikan ketika menulis program yaitu: bahwa operator peningkatan ++ dan penurunan -- jika diletakkan sebelum atau sesudah operand terdapat perbedaan.

3.7.1. Operator Increment dan Decrement

Suatu variable yang nilainya selalu bertambah satu, seperti pada sebuah variable pencacah naik (up counter) dengan rumus: hitung = hitung + 1 dapat ditulis dengan hitung += 1 dan secara singkat ditulis hitung++ atau ++hitung. Notasi ++ disebut operator penambah (increment) dan pada sebuah variable pencacah turun (down counter) dengan rumus: hitung = hitung - 1 dapat ditulis dengan hitung -= 1 dan secara singkat ditulis hitung-- atau --hitung. Notasi -- disebut operator pengurang (decrement).

Program dibawah merupakan operasi hitung perulangan sebanyak loop kali yang dimulai dari 0 naik ke satu, dua dan seterusnya. Operasi dilakukan dengan melakukan penambahan satu keatas (increment). Perhatikan program dibawah ini:

Program 3.13

```
#include <conio.h>
#include <iostream>

using namespace std;

main()
{
        int hitung = 0, loop;
        loop = ++hitung;
        cout <<"Loop = %d, Hitung = "<< loop<< hitung;
        loop = hitung++;
        cout <<"Loop = %d, Hitung = "<< loop<< hitung;
        getch();
        return 0 ;
}
```

Keluaran program diatas adalah sebagai berikut:

```
Loop    = 1, Hitung      = 1
Loop    = 1, Hitung      = 2
```

Program dibawah merupakan operasi hitung perulangan sebanyak loop kali yang dimulai dari 10 turun ke sembilan, delapan dan seterusnya. Setiap operasi looping dilakukan dengan melakukan pengurangan satu kebawah. Perhatikan contoh program dibawah ini:

Program 3.14

```
#include <conio.h>
#include <iostream>

using namespace std;

main()
{
        int hitung = 10, loop;
        loop = --hitung;
        cout <<"Loop = , Hitung = "<< loop<< hitung;
        loop = hitung--;
        cout <<"Loop = , Hitung = "<< loop<< hitung;
        getch();
        return 0 ;
}
```

Keluaran program diatas adalah sebagai berikut:

```
Loop    = 9, Hitung      = 9
Loop    = 9, Hitung      = 8
```

3.7.2. Operator sizeof

Operator akan menghasilkan ukuran dari suatu variable atau tipe pada saat dikompilasi. Ukuran ini digunakan untuk mengetahui tipe data apakah dan berapa ukuran data yang ada didalam sebuah variable tersebut. Perhatikan potongan program dibawah ini:

```
sizeof(tipe data);
sizeof(char);
sizeof(int);
```

program dibawah ini akan melakukan pengukuran terhadap variable tipe data dan kemudian hasilnya akan dimunculkan setelah dilakukan kompilasi. Perhatikan program dibawah ini:

Program 3.15

```
#include <conio.h>
#include <iostream>

using namespace std;

main()
{
    cout <<"Size of char = "<< sizeof(char);
```

```
    cout <<"Size of short int = \n"<< sizeof(short int);
    cout <<"Size of int = \n"<< sizeof(int);
    cout <<"Size of long int = \n"<< sizeof(long int);
    cout <<"Size of float = \n"<< sizeof(float);
    cout <<"Size of double = \n"<< sizeof(double);
    cout <<"Size of long double = \n"<< sizeof(long double);
    getch();
    return 0 ;
}
```

Keluaran program diatas adalah sebagai berikut:

```
Size of char          =  1 byte
Size of sort int      =  2 byte
Size of int           =  2 byte
Size of long int      =  4 byte
Size of float         =  4 byte
Size of double        =  8 byte
Size of long double   = 10 byte
```

Contoh program yang menggunakan operator unary * (pointer) dan & (alamat memori). Penjelasan dari program tersebut dapat diuraikan dalam bentuk program secara langsung sehingga lebih mudah dipahami.

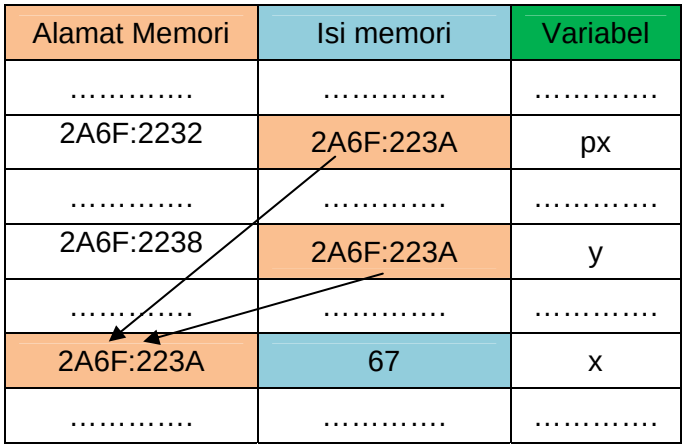
Program 3.16

```
#include <conio.h>
#include <iostream>

using namespace std;

main()
{
        int x = 67, y;          /* var pointer menunjuk ke data yg bertipe int */
        int *px;                // px diisi dengan alamat dari var x
        px = &x;                // y diisi dengan nilai yang ditunjuk oleh px
        y = *px;
        cout <<"\nAlamat x = "<< &x;
        cout <<"\nIsi px   = "<< px;
        cout <<"\nIsi x    = "<< x;
        cout <<"\nNilai yang ditunjuk oleh px = "<< *px;
        cout <<"\nNilai y  = "<< y;
        getch();
        return 0 ;
}
```

Keluaran program diatas adalah sebagai berikut:

```
Alamat x                    = 2A6F : 223A
Isi px                      = 2A6F : 223A
Isi x                       = 67
Nilai yang ditunjuk oleh px = 67
Nilai y                     = 67
```

Adapun penjelasan mengenai pointer program diatas adalah sebagai berikut :



Alamat Memori	Isi memori	Variabel
……	……	……
2A6F:2232	2A6F:223A	px
……	……	……
2A6F:2238	2A6F:223A	y
……	……	……
2A6F:223A	67	x
……	……	……

Perhatikan program Operator Unary & dan * serta gambar diatas. Variabel x dan y merupakan suatu lambang dari sebuah daerah di memori utama komputer. Artinya x dan y masing-masing sebenarnya adalah suatu alamat memori. Ketika variabel tersebut dideklarasikan sebagaimana dibawah ini:

```
int x = 67, y;
```

maka variabel x sama dengan alamat memori misalnya 2A6F:223A dialokasikan dan diisi data 67, variabel y sama dengan alamat memori misalnya 2A6F:2238 dialokasikan. Jadi x berisi 67 atau alamat memori 2A6F:223A berisi data 67.

Variabel tersebut bertipe integer maka akan menempati lokasi memori sebesar 16 bit. Alamat-alamat memori tersebut dipergunakan selama fungsi main dijalankan. Deklarasi:

```
int *px;
```

berarti variabel px sama dengan alamat memori misalnya 2A6F:2232 dialokasikan dengan ukuran memori sebesar 16 bit (ukuran tipe integer). Perintah penugasan:

```
px = &x;
```

berarti variabel px (alamat memori 2A6F:2232) diisi data 2A6F:223A (alamat dari variabel x).

Jadi isi px = 2A6F:223A, Isi px ini merupakan alamat dari variabel x, perhatikan gambar diatas agar lebih jelas. Perintah penugasan selanjutnya adalah:

```
y = *px;
```

berarti variabel y (alamat memori 2A6F:2238) diisi data alamat memori variable px (2A6F:2232). Alamat tersebut berisi data 67. Suatu alamat memori yang berisi data alamat memori disebut pointer.

3.8. Operator Binary

Operator binary adalah operator yang melibatkan atau dikenakan pada dua buah operand. Dibawah ini merupakan tergolong dalam operator binary adalah adalah sebagaimana dijelaskan dibawah ini:

3.8.1. Operator Aritmatika

Digunakan untuk mengoperasikan data-data numerik, seperti perkalian, pembagian, sisa hasil bagi, penjumlahan, dan pengurangan. Dalam proses aritmatika tersebut, pengerjaan operasi tergantung dari tingkat valensi operator-operator yang terlibat. Perkalian memiliki valensi tertinggi, kemudian dilanjutkan dengan sisa pembagian, pembagian, sedangkan penjumlahan dan pengurangan mempunyai valensi yang terendah. jenis operator aritmatika, yaitu :

Tabel 3.8. Operator aritmetika

OPERATOR	ARTI	CONTOH
*	Kali	a * b a dikalikan dengan b 3 * 2 = 6
%	Modulo atau sisa pembagian bulat	a%b sisa hasil pembagian bulat a dibagi dengan b 3 % 2 = 1
/	Bagi	a / b a dibagi dengan b 3 / 2 = 1 untuk tipe data integer 3 / 2 = 1.5 ntuk tipe data float
+	Plus atau tambah	a + b a ditambah dengan b 3 + 2 = 5
-	Minus atau kurang	a – b a dikurangi b 3 – 2 = 1

Program dibawah ini merupakan operasi-operasi aritmetika dan bertujuan untuk melakukan operasi pembagian, modulo, serta operasi kombinasi aritmatika pada bilangan bulat. Hasil operasi tersebut kemudian ditampilkan dilayar monitor. Untuk lebih jelasnya perhatikan program dibawah ini:

Program 3.17

```
#include <conio.h>
#include <iostream>
```

```
using namespace std;

main()
{
 /*operasi aritmatika
   dengan bilangan bulat */
   int v, w, x, y, z;
   v = 100;
   w = 3;
   x = v / w;
   y = v % w;
   z = v * w - w + v % v / w;
   cout <<"Operasi Aritmatika pada Bilangan Bulat\n";
   cout <<"X = \n"<< v<< w;
   cout <<"  = \n"<< x;
   cout <<"Y =   "<< v<< w;
   cout <<"  =   "<< y;
   cout <<"Z = \n"<< v<< w<< w<< v<< v<< w;
   cout <<"  = \n"<< z;
   getch();
   return 0 ;
}
```

Keluaran program diatas adalah sebagai berikut:

```
Operasi Aritmetika pada bilangan Bulat
X       = 100/3
        = 33
Y       = 100 MOD 3
        = 1
Z       = 100 * 3 – 3 + 100 mod 100/3
        = 297
```

Operator % (modulo) hanya berlaku pada tipe data integer, Perhatikan program operasi aritmatika dengan bilangan real dibawah ini:

Program 3.18

```
#include <conio.h>
#include <iostream>

using namespace std;

main()
{
      float a, b, c, d, e;
```

```
        a = 100.0;
        b = 3.0;
        c = a / b;
        d = 100 % 3;
        e = a * b - b + 100 % 100 / b;
        cout <<"\nOperasi Aritmatika pada Bilangan Real\n\n";
        cout <<"C = \n"<< a<< b;
        cout <<"  = \n"<< c;
        cout <<"D = \n"<< a<< b;
        cout <<"  = \n"<< d;
        cout <<"E = \n"<< a<< b<< b<< a<< a<< b;
        cout <<"  = \n"<< e;
        getch();
        return 0 ;
}
```

Keluaran program diatas adalah sebagai berikut:

```
Operasi Aritmetika pada bilangan Real
C       = 100.000000 / 3.000000
        = 33.333332
D       = 100.000000 mod 3.000000
        = 1.000000
E       = 100.00000 * 3.00000 – 3.00000 + 100.000000 / 3.000000
        = 297.000000
```

Dari contoh program diatas dapat dilihat dengan jelas bahwa Operator / (pembagian) dapat berfungsi sebagai pembagian bulat (div) atau pembagian real. Hal ini tergantung pada tipe data yang dipergunakan. Tingkat pengerjaan operasi dari operator aritmatika adalah: * (perkalian), % (modulo), / (pembagian), + (penjumlahan), dan – (pengurangan).

Perhatikan contoh mengenai hal tersebut diatas dapat dilihat pada program dibawah ini:

```
z = 100 * 3 - 3 + 100 % 100 / 3
```

Proses penyelesaian dalam program adalah sama halnya dengan ketika menyelesaikan persamaan aritmatika secara ditulis sebagai berikut ini:

```
z = ((100 * 3) – (3 + ((100 % 100) / 3)))
  = (   300   – (3 + ((100 % 100) / 3)))
  = (   300   – (3 + (      0      / 3 )))
  = (   300   – (3 +              0    ))
  = (   300   –  3                     )
  =  297
```

3.8.2. Operator Relasional

Operator relasi digunakan untuk membandingkan hubungan antara dua buah operand (sebuah nilai atau variable) atau digunakan untuk mewakili sebuah nilai logika (nilai boolean), dari suatu persamaan atau nilai. Jenis-jenis operator relasi,sepeti terlihat pada tabel dibawah ini:

Tabel 3.9. Operator relasi

OPERATOR	ARTI	CONTOH
<	Kurang dari	x < y Apakah x kurang dari y
<=	Kurang dari sama dengan	x <= y Apakah x kurang dari sama dengan y
>	Lebih dari	x > y Apakah x lebih dari y
>=	Lebih dari sama dengan	x >= y Apakah x lebih dari sama dengan y
==	Sama dengan	x == y Apakah x sama dengan
!=	Tidah sama dengan	x != y Apakah x tidak sama dengan y

3.8.3. Operator logika

Operator logika digunakan untuk membandingkan logika hasil dari operator-operator relasi atau digunakan untuk mengoperasikan operand (konstanta, variabel, atau suatu ekspresi) secara logis. Operator logika ada tiga macam yaitu operator AND, OR dan operator NOT. Untuk lebih jelasnya perhatikan dalam tabel berikut:

Tabel 3.10. Operator logika

OPERATOR	ARTI	CONTOH	
&&	AND	a b a && b 0 0 0 0 1 0 1 0 0 1 1 1	(2<1) && (3<1) = 0 0 && 0 = 0 (2<4) && (3<1) = 0 1 && 0 = 0
||	OR	a b a || b 0 0 0 0 1 1 1 0 1 1 1 1	(2<1) || (4<1) = 0 0 || 0 = 0 (2<4) || (3<1) = 1 1 || 0 = 0
!	NOT	a !a !(2>3) = 1 0 1 !0 = 1 1 0	

Program Operasi logika pada Operator Binary Logika

Program 3.19

```
include <conio.h>
#include <iostream>

using namespace std;

main()
{
      int x, x1, x2, y, y1, y2, z, z1, z2, a, b, c;
      a = 125;
      b = 100;
      c = 25;
      x1 = a < b;
      x2 = a > b;
      y1 = c <= b;
      y2 = c >= b;
      z1 = a == c;
       z2 = a != c;
      x = x1 && x2;
      y = y1 || y2;
      z = !z1;
      cout <<"\nA  = "<< a<< b<< c ;
      cout <<"\nX1 = " ;
      cout <<"\n   = "<< a<< b ;
      cout <<"\n   = "<< x1 ;
      cout <<"\nX2 = " ;
      cout <<"\n   = "<< a<< b ;
      cout <<"\n   = "<< x2 ;
      cout <<"\nY1 = " ;
      cout <<"\n   = "<< c<< b ;
      cout <<"\n   = "<< y1 ;
      cout <<"\nY2 = " ;
      cout <<"\n   = "<< c<< b ;
      cout <<"\n   = "<< y2 ;
      cout <<"\nZ1 = " ;
      cout <<"\n   = "<< a<< c ;
      cout <<"\n   = "<< z1 ;
      cout <<"\nZ2 = " ;
      cout <<"\n   = "<< a<< b ;
      cout <<"\n   = "<< z2 ;
      cout <<"\nX  = " ;
      cout <<" \n  = "<< x1<< x2 ;
```

```
    cout <<" \n  = "<< x ;
    cout <<"Y\n  = " ;
    cout <<"\n  = "<< y1<< y2 ;
    cout <<"\n   = "<< y ;
    cout <<"Z\n  = " ;
    cout <<"\n   = "<< z1 ;
    cout <<"\n   = "<< z ;
    getch();
    return 0 ;
}
```

Keluaran program diatas adalah sebagai berikut:

```
A     = 125
B     = 100
C     = 25
X1    = a < b
      = 125 < 100
      = 0
X2    = a > b
      = 125 > 100
      = 1
V1    = C <= B
      = 25 <= 100
      = 1
V2    = C >= B
      = 25 <= 100
      = 0
Z1    = A != C
      = 125 != 25
      = 0
Z2    = A == B
      = 125 == 100
      = 1
X     = X1 && X2
      = 0 && 1
      = 0
Y     = Y1 || Y2
      = 1 || 0
      = 1
Z     = !Z1
      = !0
      = 1
```

3.8.4. Operator Bitwise atau manipulasi bit

Operator bitwise digunakan untuk memanipulasi bit-bit dari nilai data yang ada di memori. Semua operator bitwise hanya bisa dikenakan pada operand bertipe integer atau karakter. Operator bitwise sebagaimana terlihat dalam tabel dibawah ini:

Tabel 3.11. Operator bitwise

OPERATOR	ARTI	CONTOH
<<	Pergeseran bit ke kiri	9 << 2
>>	Pergeseran bit ke kanan	9 >> 2
&	Bitwise AND	9 & 2
^	Bitwise XOR (Exclusive OR)	9 ^ 2
|	Bitwise OR	9 | 2
~	Bitwise NOT	~ 9

Perhatikan contoh operasi bitwise XOR antara data 9D dengan 2D maka akan menghasilkan data 11D

```
 9D   =  00001001B di XOR dengan
 2D   =  00000010B
11D   =  00001011B <= hasil XOR
```

Atau contoh lain misalnya antara data 9D dengan 7D maka akan dihasilkan data 14D seperti dibawah ini:

```
 9D   =  00001001B di XOR dengan
 7D   =  00000111B
14D   =  00001110B <= hasil XOR
```

Tabel 3.12. Kebenaran XOR

A	B	A XOR B
0	0	0
1	0	1
0	1	1
1	1	0

Pada tabel XOR tersebut diatas adalah: data keluaran akan bernilai 0 jika data masukannya bernilai sama baik 0 semua atau 1 semua. Dalam operasi aritmetika sebuah data 9D misalnya akan digeser kekanan atau

kekiri untukan operasi baik OR, AND, OR maupun NOT. Perhatikan contoh pergeseran data 9D yang digeser kekiri maka nilai akan berubah.

Perubahan tersebut akan mewakili sebuah operasi perkalian, pembagian atau operasi yang lain. Perhatikan contoh pergeseran data 9D dibawah ini:

```
9D = 00001001B
```

digeser kekiri:

```
1x      => 00010010B = 18D
2x      => 00100100B = 36D
```

Kalau digeser kekanan:

```
1x      =>  00000100B = 4D
2x      =>  00000010B = 2D

9D      =   00001001B di AND dengan
2D      =   00000010B
0D      =   00000000B <= hasil AND

 9D     =   00001001B di OR dengan
 2D     =   00000010B
11D     =   00001011B <= hasil OR
 9D     =   00001001B di NOT menjadi:
~9D     =   11110110B = -10D model komplemen 2
```

Selain hasil tersebut diatas, dapat dibuktikan dengan cara melakukan komplemen kedua. Komplemen satu yaitu dengan melakukan konversi data 0 menjadi 1 atau sebaliknya, sedangkan komplemen kedua yaitu dengan melakukan penambahan data 01 pada hasil komplemen pertama. Untuk lebih jelasnya perhatikan contoh dibawah ini:

```
 10D    =   00001010B
-10D    =   11110101B model komplemen 1
        =   11110110B model komplemen 2
```

Untuk menyatakan bilangan negatip dalam bilangan biner dipakai model komplemen. Model komplemen 2 diperoleh dengan cara model komplemen 1 ditambah 1.

Program 3.20. Operasi bit pada Operator Bitwise

```
#include <conio.h>
#include <iostream>
```

```
using namespace std;

main()
{
      int u, v, w, x, y, z;
      u = 9 << 2;
      v = 9 >> 2;
      w = 9 & 2;
      x = 9 ^ 2;
      y = 9 | 2;
      z = ~9;
      cout <<"U  = 9 << 2\n";
      cout <<"   = %d\n", u;
      cout <<"V  = 9 >> 2\n";
      cout <<"   = %d\n", v;
      cout <<"W  = 9 & 2\n";
      cout <<"   = %d\n", w;
      cout <<"X  = 9 ^ 2\n";
      cout <<"   = %d\n", x;
      cout <<"Y  = 9 | 2\n";
      cout <<"   = %d\n", y;
      cout <<"Z  = ~9\n";
      cout <<"   = %d\n", z;
      getch();
      return 0 ;
}
```

Keluaran program diatas adalah sebagai berikut:

```
U     = 9<< 2
      = 36
V     = 9>>2
      = 2
W     = 9 & 2
      = 0
X     = 9 ^ 2
      = 11
Y     = 9 | 2
      = 11
Z     = ~9
      = -10
```

Contoh program diatas menunjukkan bahwa hasil operasi dari operator relasi atau operator logika hanya mempunyai satu nilai dari dua nilai kebenaran, yaitu :

- 0 = false = salah atau
- 1 = true = benar

3.8.5. Operator Penugasan dan Operator Kombinasi

Operator penugasan (*Assignment operator*) atau pemberi nilai berupa tanda sama dengan ("="). Perhatikan contoh dibawah ini:

```
nilai = 80;  Artinya : variable "nilai" diisi dengan 80
A = x * y; Artinya : variable "A" diisi dengan hasil perkalian antara x dan y.
```

Operator kombinasi merupakan gabungan antara dua operator tertentu, yaitu antara operator aritmatika atau operator bitwise dengan operator penugasan yang digunakan untuk memendekkan penulisan operasi penugasan, seperti berikut:

Tabel 3.13. Operator penugasan

OPERATOR	ARTI	CONTOH	ARTINYA
*=	Memberi nilai setelah nilai semula dikalikan	x *= 5;	x = x * 5;
/=	Memberi nilai sisa bagi setelah nilai semula dibagi	x /= 5;	x = x / 5;
%=	Memberi nilai sisa bagi dari pembagian nilai semula	x %= 5;	x = x % 5;
+=	Memberi nilai setelah nilai semula ditambahkan	x += 5;	x = x + 5;
- =	Memberi nilai setelah nilai semula dikurangkan	x -= 5;	x = x - 5;
<<=	Memberi nilai dari pergeseran bit ke kiri	x <<= 5;	x = x << 5;
>>=	Memberi nilai dari pergeseran bit ke kanan	x >>= 5;	x = x >> 5;
&=	Memberi nilai hasil bitwise AND	x &= 5;	x = x & 5;
|=	Memberi nilai hasil bitwise OR	x |= 5;	x = x | 5;
^=	Memberi nilai hasil bitwise XOR	x ^= 5;	x = x ^ 5;

3.9. Operator Ternary

Operator ternary adalah operator yang melibatkan tiga buah operand. Yang tergolong operator ini adalah: Operator kondisi dengan symbol ?: Bentuk ungkapan operator kondisi adalah :

```
<kondisi> ? <ungkapan_1> :
<ungkapan_2>
```

Maksud dari ungkapan kondisi adalah: Jika kondisi bernilai benar, maka nilai ungkapan kondisi berupa nilai ungkapan_1 dan jika kondisi bernilai salah, maka nilai ungkapan kondisi berupa nilai ungkapan_2. Perhatikan contoh dibawah ini:

```
maksim = nilai_awal > nilai_akhir ?
nilai_awal : nilai_akhir;
```

maksud dari contoh diatas dapat di ungkapkan dalam bentuk kalimat seperti ini:

```
jika nilai_awal = 80 dan nilai_akhir = 75, maka maksim = 80 (sama dengan nilai_awal)
jika nilai_awal = 75 dan nilai_akhir = 80, maka maksim = 80 (sama dengan nilai_akhir)
jika nilai_awal = 75 dan nilai_akhir = 75, maka maksim = 75 (sama dengan nilai_akhir)
```

Program 3.21. Operasi kondisi pada Operator Kondisi

```
#include <conio.h>
#include <iostream>

using namespace std;

main()
{
        int a, b, c, d;
        a = 80;
        b = 75;
        c = a > b ? a : b;
        cout <<"Mencari nilai yang lebih tinggi\n\n";
        cout <<"a = \n"<< a<< b;
        cout <<"c = \n";
        cout <<"  = \n"<< a<< b<< a<< b;
        cout <<"  = \n\n"<< c;

        a = 75;
        b = 80;
        c = a > b ? a : b;
        cout <<"Mencari nilai yang lebih tinggi\n\n";
        cout <<"a = \n"<< a<< b;
        cout <<"c = \n";
        cout <<"  = \n"<< a<< b<< a<< b;
        cout <<"  = \n"<< c;
        getch();
        return 0 ;
}
```

Keluaran program diatas adalah sebagai berikut:

```
Mencari Nilai yang lebih tinggi
a     = 80
b     = 75
```

```
c     = a > b ? a : b
      = 80 > 75 ? 80 : 75
      = 80

Mencari Nilai yang lebih tinggi
a     = 75
b     = 80
c     = a > b ? : b
      = 75 > 80 ? 75 : 80
      = 80
```

3.10. Ungkapan (Ekspresi)

Ungkapan dapat berupa konstanta (untai/numerik), variabel dan nilai tunggal yang diperoleh dengan mengkombinasikan operand dan operator, seperti 5+4. Ungkapan-ungkapan dibagi menjadi empat kategori :

a. Ungkapan numerik

```
2 + 5          3 * 4     2 + 7 / 5
```

b. Ungkapan string

```
"ABCD" + "EFGH"
nim + nama
```

Satu-satunya operator yang berlaku pada ungkapan string hanyalah tanda +, yang berfungsi untuk menggabungkan dua untai.

c. Ungkapan relasi/hubungan
Tipe untai dapat juga menggunakan operator relasi seperti halnya dengan tipe numerik. Misalnya diketahui bahwa:

```
'A' lebih kecil dari 'B'
```

d. Ungkapan logika

```
!A
((A>5) && (B=4))
((Nama = "Agus") || (nama =
"Doni"))
```

e. Program sederhana menggunakan Tipe data

Program 3.22. contoh penggunaan tipe data

```
#include <conio.h>
#include <iostream>

using namespace std;

typedef unsigned char byte;

main()
{
        long int data1 = 546767226531;
        int data2 = 2235641;
```

```
        short  int data3 = 714;
        byte data4 = 34;
        float ata6 = 1.733; // tipe data pecahan
        double ata5 = 4.967; // tipe data pecahan
        char data7 = 'C';
        enum boolean {false, true};
        enum boolean kondisi;
        kondisi = true;
        char data8[6];
        data8[] = kondisi = = 1 ? "true":"false";
        printf("Nilai Long :  %ld\n"<< data1;
        cout <<"Nilai Int : %d\n "<< data2;
        cout <<"Nilai Short : %hd\n"<< data3;
        cout <<"Nilai Byte : %d\n"<< data4;
        cout <<"Nilai Double : %lf\n"<< data5;
        cout <<"Nilai Float : %f\n"<< data6;
        cout <<"Nilai Char : %c\n"<< data7;
        cout <<"Nilai Boolean : %s\n"<< data8;
        getch(;
        return 0 ;
}
```

3.11. Soal Latihan

Jawablah soal latihan dibawah ini dengan baik dan benar.

1. Apa yang dimaksud dengan data
2. Sebutkan jenis-jenis data yang digunakan pada bahasa pemrograman
3. Apa yang dimaksud dengan konstanta
4. Buatlah sebuah program sederhana menggunakan variabel konstanta
5. Apa yang dimaksud dengan operator unary, binary dan ternary
6. Apa yang dimaksud dengan variabel
7. Sebutkan tipe variabel yang digunakan dalm bahasa c++
8. Buatlah program sederhana yang menggunakan operator unary, binary dan ternary

BAB 4
STRUKTUR PERULANGAN

4.1. Perulangan

Perulangan atau iterasi atau yang biasa disebut dengan “looping” adalah proses melakukan tindakan yang sama secara berulang-ulang atau berkali-kali sampai batas yang telah ditentukan. Perulangan digunakan untuk menjalankan satu atau beberapa pernyataan sebanyak beberapa kali. Dengan kata lain, perulangan dipakai untuk menjalankan beberapa pernyataan dengan hanya menuliskan pernyataan tersebut satu kali. Hal ini banyak sekali dijumpai dalam pemrograman.

Perulangan proses dalam bahasa pemrograman ditangani dengan suatu mekanisme yang disebut loop. Dengan memakai loop, suatu proses yang berulang misalnya menampilkan angka 1 sampai 1000 atau tulisan yang sama sebanyak sepuluh kali di layar dapat diimplementasikan dengan kode program yang pendek.

Pada pemrograman proses perulangan dibagi menjadi 2 jenis, yaitu: Perulangan yang telah diketahui jumlah perulangannya sebelum perulangan tersebut di lakukan. Jenis perulangan ini dilakukan dengan penyataan for. Dan kedua adalah perulangan yang belum di ketahui jumlah perulangannya sebelum perulangan tersebut dilakukan. Perulangan jenis ini terdiri

dari dua kategori, yaitu: kondisi perulangan diperiksa diawal perulangan. Jenis perulangan ini dilakukan dengan penyataan while. Kondisi perulangan diperiksa diakhir perulangan. Jenis perulangan ini dilakukan dengan penyataan do-while. Struktur perulangan secara umum terdiri dari dua bagian:

- Kondisi perulangan, yaitu ekspresi boolean yang harus dipenuhi untuk melaksanakan perulangan.
- Badan (body) perulangan, yaitu bagian algoritma yang diulang.

4.2. Operator Increment dan Decrement

Sebelum jauh membahas perulangan, akan dipelajari dahulu mengenai operasi increment. Operator Increment digunakan untuk menaikan atau bisa juga untuk meningkatkan nilai dengan satu, sedangkan decrement digunakan untuk mengurangi nilai turun dengan satu. Kedua pernyataan berikut merupakan untuk menaikan variabel num dengan satu:

```
num = num + 1;
num += 1;
```

num di kurangi dengan satu dapat dilihat dari pernyataan berikut ini:

```
num = num - 1;
num -= 1;
```

dalam bahasa C++ menyediakan satu set operator unary sederhana yang dirancang hanya untuk menambah dan mengurangi sebuah variabel dengan 1. Operator increment adalah mengunakan kode ++. Sedangkan operator decrement adalah --. Pernyataan yang menggunakan operator ++ untuk menaikan variabel num adalah sebagai berikut:

```
num++;
```

sedangkan pernyataan decrement yang digunakan untuk mengurangi variabel num adalah sebagai berikut:

```
num - - ;
```

dalam bahasa pemrograman baik sebelum maupun sesudah bab ini dibahas sering menggunakan operasi kenaikan dan pengurangan yang menggunakan dalam mode postfix, mode postfik artinya operator diletakkan setelah variabel. Operator juga bekerja dalam mode prefik, dimana operator ditempatkan sebelum nama variabel:

```
+ + num;
- - num;
```

Kedua operator mode postfix dan prefix diatas akan menambahkan 1 atau mengurangi dengan 1 pada setiap operand. Program dibawah menunjukkan operator increment dan decrement.

Program 4.1

```
#include <iostream>

using namespace std;

int main()
{
	int bigVal = 10, smallVal = 1;
	cout << "vabesar adalah " << bigVal << " dan valkecil adalah " << smallVal << endl;
	smallVal++;
	bigVal--;
	cout << "vabesar adalah " << bigVal<< " dan valkecil adalah " << smallVal << endl;
	++smallVal;
	--bigVal;
	cout << "vabesar adalah " << bigVal<< " dan valkecil adalah " << smallVal << endl;
	return 0;
}
```

Keluaran programnya adalah :

```
vabesar adalah 10 dan valkecil adalah 1
vabesar adalah 9 dan valkecil adalah 2
vabesar adalah 8 dan valkecil adalah 3
```

4.3. Ekspresi Matematika ++ dan - -

Pada operator kenaikan dan pengurangan dapat juga digunakan pada variabel dalam ekspresi matematika. Perhatikan potongan program berikut ini:

```
a = 2;
b = 5;
c = a * b++;
cout << a << " " << b << " " << c;
```

dalam pernyataan c = a * b++, c adalah hasil dari perkalian a dan b, dimana 10. Merupakan sebuah variabel b yang ditambahkan dengan satu. Kemudian dengan menggunakan pernyataan cout maka hasil ditampilkan sebagai berikut:

```
2 6 10
```

Jika pernyataannya c tersebut diatas berubah maka dapat dibaca:

```
c = a * ++b;
```

variabel b akan ditambahkan dengan satu sebelum keduanya dikalikan dengan a. dalam kasus ini c merupakan hasil dari nilai 2 dikalikan 6, sehingga pernyataan cout akan menampilkan:

```
2 6 12
```

Dengan satu saja dapat membawa beberapa aksi dalam satu pernyataan menggunakan operator kenaikan dan pengurangan, hal tersebut juga tidak terlalu rumit untuk digunakan. Perhatikan potongan program dibawah ini:

```
a = 2;
b = 5;
c = ++(a * b);     // Error!
```

```
x = 10 ;
if (x + +> 10)
  cout << "x lebih besar daripada 10. \n";
```

pernyataan tersebut diatas merupakan pernyataan sederhana yang tidak bekerja karena adanya operator kenaikan dan pengurangan. Pada operator kenaikan dan pengurangan biasanya ada variabel operand, tetapi umumnya, sesuatu yang ada di sebelah kiri dari operator yang disetujui.

Seperti sudah kita ketahui bahwa dalam bab ini, operator ++ dan -- digunakan sebagai kalimat penghubung. Sama seperti dalam ekspresi matematika, perbedaan antara mode postfix dan prefix sangat dekat. Perhatikan potongan program berikut ini:

Dua operasi yang terjadi dalam pernyataan IF tersebut diata adalah: (1) nilai yang diuji pada x untuk menentukan apakah nilainya lebih besar daripada 10, dan (2) x adalah diincremented. Karena kenaikan operator yang digunakan dalam mode postfix, Bandingan hal yang terjadi terlebih dahulu. Sejak 10 tidak lebih dari 10, pernyataan cout tidak akan dijalankan. Jika mode kenaikan operator berubah, sedangkan jika pernyataan akan membandingkan 11-10 dan pernyataan cout akan dijalankan.

4.4. Penghitung

Kadang-kadang penting bagi sebuah program untuk melacak jumlah Iterasi yang dilakukan dalam satu loop. Misalnya, Program dibawah menampilkan sebuah tabel yang terdiri dari angka 1 sampai dengan 10, jadi loop harus iterasi 10 kali. Program tersebut dibawah akan menampilkan angka 1 sampai dengan 10 dan kemudian dikuadratkan:

Program 4.2.

```
include <iostream>

using namespace std;

int main()
{
    int num = 1;                        // inisialisasi penghitung
    cout << " Angka      Angka kuadrat\n";
    cout << "-------------------------\n";
    while (num <= 10)
    {
        cout << num << "\t\t" << (num * num) << endl;
        num++;                          // penghitung Increment
```

```
    }
    return 0;
}
```

Keluaran programnya adalah sebagai berikut:

```
Angka    Angka kuadrat
-------------------------
1                1
2                4
3                9
4                16
5                25
6                36
7                49
8                64
9                81
10               100
```

Dalam Program diatas, variabel num, dimulai dari angka 1, kemudian diincrement pada setiap kali putaran loop. Ketika num mencapai 11 lingkaran berhenti. num digunakan sebagai counter variabel, yang berarti variabel tersebut secara teratur di increment pada setiap perulangan. Pada dasarnya, num terus menghitung jumlah lingkaran Iterasi yang telah dilakukan. Ketika variabel num digunakan sebagai pengendalian pada loop dan kapan akan keluar dari loop, disebut lingkaran kontrol variabel.

Dalam contoh program diatas, penambahan variabel num adalah pada lingkaran. Pendekatan lain adalah dengan menggabungkan laba operasi yang berhubungan dengan pengujian, seperti yang ditunjukkan dalam pada program dibawah ini.

Program 4.3

```
#include <iostream>

using namespace std;

int main()
{
    int num = 0;
    cout << "Angka      Angka kuadrat\n";
    cout << "-------------------------\n";
     while (num++ < 10)
          cout << num << "\t\t" << (num * num) << endl;
          return 0;
    }
```

Keluaran programnya adalah sebagai berikut:

```
Angka    Angka kuadrat
-------------------------
1                1
2                4
3                9
4                16
5                25
6                36
7                49
8                64
9                81
10               100
```

Perhatikan bahwa sekarang variabel num diinisialisasi ke 0, bukan 1, dan relatif menggunakan ekspresi <operator bukan <=. Hal ini karena cara operator untuk menaikan bekerja jika digabungkan dengan ekspresi penghubung.

Kenaikan operator yang digunakan dalam mode postfix, berarti pada variabel num menambah satu setelah berhubungan ujian. Ketika lingkaran pertama melaksanakan, num diatur kenilai 0, kemudian 0 dibandingkan dengan 10. Operator ++ kemudian akan menambahkan variabel num segera setelah dilakukan perbandingan. Bila pernyataan cout dieksekusi, num memiliki nilai 1.

num dibandingkan dengan 10, kemudian di-increment. Bila pernyataan cout dilaksanakan, num 1 bernilai lebih besar dari itu pengujian tersebut.

```
while (num++ < 10)
  cout << num << "\t\t" << (num * num) << endl;
```

Di dalam looping, num selalu memiliki nilai lebih dari 1 dibanding sebelumnya nilai 10. Itulah mengapa penghubung operator menggunakan < bukan <=. Ketika variabel num bernilai 9 maka diuji, sehingga akan menjadi bernilai 10 pada pernyataan cout.

4.5. Pernyataan FOR

Jenis loop dalam bahasa C++ adalah loop FOR. FOR sangat ideal untuk situasi yang memerlukan penghitung karena ekpresinya sudah *built-in* (menjadi satu) dengan memperbarui variabel. Berikut ini adalah format perulangan FOR

```
for loop.
for (initialization; test; update)
{
        pernyataan;
        pernyataan;
        // tempat banyaknya pernyataan
```

Bisa juga sebuah perulangan menggunakan pernyataan for ditulis dalam bentuk format potongan program sebagai berikut:

```
for (inisialisasi; syarat perulangan; pengubah nilai pencacah)
      pernyataan;
```

Bisa juga ditulis seperti dibawah ini:

```
for (ungkapan1; ungkapan2; ungkapan3)
      pernyataan;
```

Penjelasan mengenai format pernyataan FOR diatas adalah seperti dibawah ini:

- Ungkapan1 atau inisialisasi: digunakan untuk memberikan inisialisasi terhadap variabel pengendali loop.

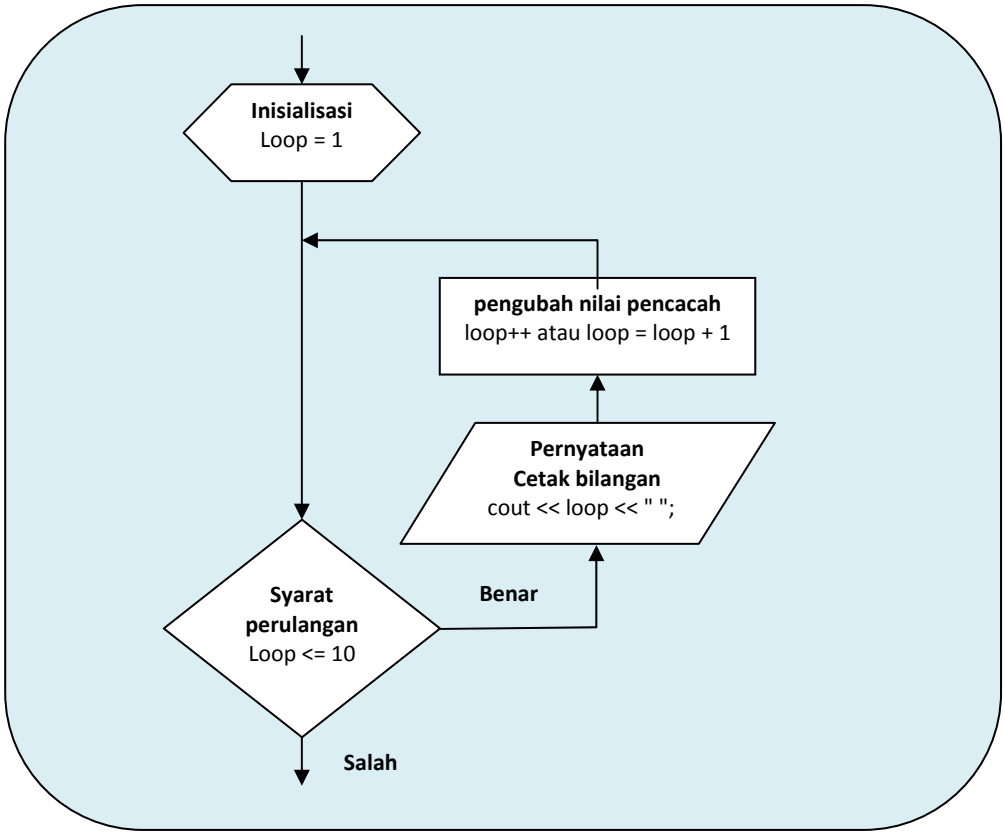
- Ungkapan2 atau sebagai syarat perulangan: dipakai sebagai pemegang kontrol terhadap pengulangan, karena bagian ini yang akan menentukan suatu kondisi perulangan untuk diteruskan atau dihentikan dari loop.

- Ungkapan3 atau pengubah nilai pencacah: dipakai sebagai pengatur perubahan nilai variabel pengendali loop. Perubahan nilai bisa kenaikan atau penurunan nilai pencacah

Ketiga ungkapan dalam FOR tersebut harus dipisahkan dengan titik koma (;). Pernyataan dalam for dapat berupa pernyataan tunggal maupun jamak (lebih dari satu). Jika pernyataannya berbentuk jamak, maka pernyataan-pernyataan tersebut harus diletakan didalam satu blok dengan memakai tanda kurung kurawal seperti berikut:

```
for (inisialisasi; syarat pengulangan; pengubah nilai pencacah)
{
      pernyataan_1;
      pernyataan_2;
      ……
      Pernyataan_n;
}
```

Berikut ini adalah sebuah diagram alir pernyataan FOR adalah sebagai berikut:



Gambar 4.1. Diagram Alir Pernyataan FOR

Seperti perulangan dua lainnya, jika hanya ada satu pernyataan dalam tubuh perulangan, kotak pernyataan (brace) akan dapat diabaikan. Perulangan FOR mempunyai tiga ekspresi didalam tanda kurung, dipisahkan oleh semicolon. (Perhatikan tidak ada koma setelah ketiga ekspresi.)

Pertama adalah ekspresi initialization expression. Ekspresi ini biasanya digunakan untuk sebuah initialize penghitung atau variabel yang harus memiliki nilai awal. Ini adalah tindakan yang pertama dilakukan oleh perulangan dan hanya dilakukan sekali.

Ekspresi yang kedua adalah test expression. Seperti ekspresi test dalam perulangan while dan do-while, test expression mengendalikan pelaksanaan perulangan. Selama ini ungkapan itu benar, mka tubuh perulangan FOR akan mengulang lagi.

Ekpresi ketiga adalah ekspresi update expression. Ekspresi ini akan melaksanakan diakhir setiap perulangan. Biasanya, ia akan menambahkan sebuah penghitung atau variabel yang harus diubah pada setiap perulangan. Program dibawah adalah Program yang menggunakan FOR sebagai pengganti dari while loop.

Program dibawah menggunakan pernyataan FOR untuk menampilkan angka 1 sampai dengan 10 dan kemudian mengkuadratkannya. Untuk lebih jelasnya perhatikan program dibawah ini:

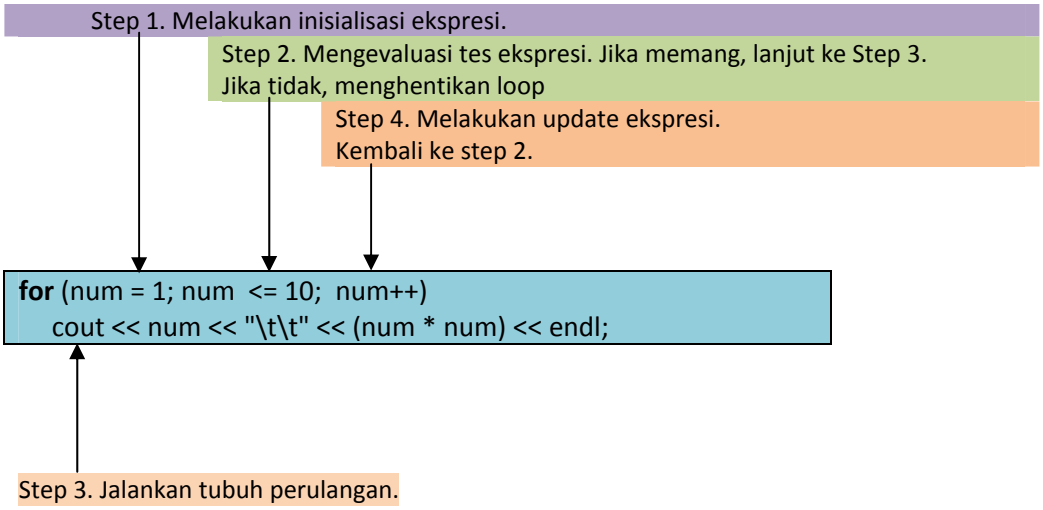
Program 4.4

```
#include <conio.h>
#include <iostream>

using namespace std;

int main()
{
    int num;
    cout << "Angka     Angka kuadrat \n";
    cout << "--------------------------------\n";
    for (num = 1; num <= 10; num++)
          cout << num << "\t\t" << (num * num) << endl;
    getch ();
    return 0;
}
```

Keluaran program diatas adalah sebagai berikut:

```
Angka     Angka kuadrat
------------------------------
1                 1
2                 4
3                 9
4                 16
5                 25
6                 36
7                 49
8                 64
9                 81
10                100
```

Gambar dibawah menjelaskan proses langkah-langkah program mengenai mekanisme looping dalam diatas. perulangan untuk menjelaskan



Gambar 4.2. Mekanisme Perulangan

Meskipun pada umumnya gaya pemrograman diatas dianggap jelek, tetapi satu atau beberapa kalimat perulangan dapat diabaikan. Initialization ekspresi yang mungkin dapat dihilangkan dari dalam kurung loop, jika sudah dilakukan atau jika tidak ada initialization diperlukan. Berikut dibawah ini adalah contoh program perulangan dalam melakukan pengaksesan sebelum perulangan:

```
int num = 1;
for ( ; num <= 10; num++)
      cout << num << "\t\t" << (num * num) << endl;
```

Kita juga dapat mengabaikan pembaruan ekspresi, jika sedang dilakukan di tempat lain didalam perulangan atau jika tidak ada yang diperlukan. perulangan berikut ini untuk bekerja seperti loop while.

```
int num = 1;
for ( ; num <= 10; )
{
      cout << num << "\t\t" << (num * num) << endl;
      num++;
}
```

Kita bahkan dapat tidak menggunakan atau mengabaikan semua tiga ekspresi dari tanda kurung loop FOR. Jika hal tersebut diabaikan maka akan terjadi peringatan bahwa kita membiarkan test expression. Perhatikan contoh dibawah ini:

```
for ( ; ; )
    cout << "Hello World\n";
```

Karena perulangan ini tidak memiliki cara untuk berhenti, maka program diatas akan menampilkan kata "Hello World \ n" selamanya atau sampai terjadi interupsi program.

Bentuk lain yang memperbarui Ekspresi dilakukan supaya kita tidak dibatasi dalam menggunakan pernyataan increment pada ekspresi update. Berikut ini adalah sebuah perulangan yang menampilkan semua atau bahkan angka 2 hingga 100 dengan menambahkan 2 pada setiap penghitung. Perhatikan program dibawah ini:

```
for (number = 2; number <= 100; number += 2)
    cout << number << endl;
```

program dibawah ini merupakan program perulangan yang melakukan hitungan turun mulai dari 10 turun sampai nilai 0.

```
for (number = 10; number >= 0; number--)
    cout << number << endl;
```

perulangan dibawah ini tidak memiliki badan resmi. Gabungan kenaikan operasi dan pernyataan cout pembaruan ekspresi akan melakukan semua pekerjaan pada setiap iterasi:

```
for (number = 1; number <= 10; cout << number++);
```

Jika perulangan memerlukan melakukan lebih dari satu pernyataan sebagai bagian dari inisialisasi, pernyataan dipisahkan dengan tanda koma. Program dibawah adalah versi Program sebelumnya yang diubah untuk memberitahu pengguna mengenai angka penjualan selama satu minggu.

Program dibawah ini akan berlangsung setiap hari selama satu minggu dan kemudian program akan menghitung totalnya. Perhatikan program berikut ini:

Program 4.5

```
#include <iostream>
#include <iomanip>

using namespace std;

int main()
{
const int NUM_DAYS = 7;

int count;
double total;
      for (count = 1, total = 0.0; count <= NUM_DAYS; count++)
      {
            double sales;
            cout << "Masukkan penjualan untuk hari  " << count << ": ";
            cin  >> sales;
            total += sales;
      }
       cout << fixed << showpoint << setprecision(2);
      cout << "Total penjualan adalah $" << total << endl;
      return 0;
}
```

Keluaran Program setelah memberikan masukan adalah sebagai berikut:

```
Masukkan penjualan untuk hari  1: 489.32[Enter]
Masukkan penjualan untuk hari  2: 421.65[Enter]
Masukkan penjualan untuk hari  3: 497.89[Enter]
Masukkan penjualan untuk hari  4: 532.37[Enter]
Masukkan penjualan untuk hari  5: 506.92[Enter]
Masukkan penjualan untuk hari  6: 489.01[Enter]
Masukkan penjualan untuk hari  7: 476.55[Enter]
Total penjualan adalah $ 3413.71
```

Dalam perulangan menggunakan FOR, penghitung diinisialisasi ke 1, maka total inisialisasi adalah 0,0. Kita dapat menempatkan lebih dari satu pernyataan untuk melakukan pembaharuan sebuah ekspresi. Perhatikan program dibawah ini:

```
double sales;
for (count = 1, total = 0.0; count <= NUM_DAYS; count++, total += sales)
{
   cout << "Masukkan penjualan untuk hari  " << count << ": ";
   cin  >> sales;
```

```
}
```

Dalam melakukan pembaruan ekspresi sebuah perulangan, penghitung di increment, maka nilai penjualan tersebut akan ditambahkan ke total pada akhir setiap perulangan.

Perhatikan contoh mengenai program looping menggunakan FOR. Program dibawah ini digunakan untuk mencetak bilangan dari 1 hingga 10 secara naik.

Program 4.6

```
#include<conio.h>
#include <iostream.h>

using namespace std;

int main()
{
      int loop;
      for (loop = 1; loop <= 10; loop++)
      cout << loop << " ";
      getch();
      return 0;
}
```

Keluaran program adalah sebagai berikut:

```
1 2 3 4 5 6 7 8 9 10
```

Pada program diatas terdapat

```
int loop;
```

tipe data variabel loop adalah integer

```
for (loop = 1; loop <= 10; loop++)
      cout << loop << " ";
```

Perintah di atas akan menampilkan angka dari 1 dan sebuah spasi kosong sampai angka 10 dan spasi kosong secara horizontal atau berjajar. Proses tersebut dilakukan secara iterasi sepuluh kali yang diawali dengan nilai loop = 1, kenaikan nilai loop satu (loop++), dan berakhir sampai syarat terpenuhi loop <= 10, , yaitu: loop = 10. Perhatikan tanda loop++ yang berarti loop = loop + 1. Proses perulangan potongan program diatas adalah sebagai berikut:

```
Pada loop = 1 ditampilkan : 1  disambung dengan loop ke 2
Pada loop = 2 ditampilkan : 2  disambung dengan loop ke 3
……
……
……
Pada loop = 10 ditampilkan : 10
```

Pada loop = 11 proses iterasi selesai atau berhenti karena nilai ini sudah tidak memenuhi syarat yang ditentukan, yaitu: 11 <= 10.

Selanjutnya adalah:

```
getch();
```

meminta masukan sembarang tombol, perintah ini dimaksudkan untuk menahan hasil tampilan dilayar supaya tetap nampak dan akan menutup sampai adanya sembarang tombol ditekan.

```
return 0;
```

untuk memberi nilai kembalian pada fungsi main. Pada program di atas, kenaikan terhadap variabel pengendali loop sebesar 1 (positif), yang dinyatakan dengan ungkapan:

```
loop ++
```

artinya ungkapan tersebut sama dengan:

```
loop = loop + 1
```

Kenaikan terhadap variabel pengendali loop bisa diatur jaraknya dengan mengatur stepnya. Misalnya untuk jarak kenaikkannya 2, maka dinyatakan dengan:

```
Loop += 2
```

Hal diatas sama artinya sama dengan program dibawah ini:

```
Loop = loop + 2
```

Pada contoh yang melibatkan pernyataan for diatas, kenaikan variabel pengendali loop berupa nilai positif. Sebenarnya kenaikan terhadap variabel pengendali loop dapat pula diatur supaya bernilai negatif, seperti dicontohkan pada program dibawah ini. Program dibawah akan mencetak bilangan dari 10 hingga 1 secara menurun.

Program 4.7

```
#include<conio.h>
#include <iostream.h>

using namespace std;

int main()
{
      int loop;
      for (loop = 10; loop >= 1; loop--)
          cout << loop << " ";
          getch();
      return 0;
}
```

Hasil keluaran program adalah sebagai berikut:

```
10 9 8 7 6 5 4 3 2 1
```

Program lainnya yang mennggunakan pernyataan FOR untuk menampilkan bilangan ganjil antara 1 hingga 10 adalah sebagai berikut:

Program 4.8

```
#include<conio.h>
#include <iostream.h>

using namespace std;

int main()
{
      int loop;
      for (loop = 1; loop <= 10; loop+=2)
          cout << loop << " ";
      getch();
      return 0;
}
```

Hasil keluaran program diatas adalah sebagai berikut:

```
1 3 5 7 9
```

Selain berupa angka, pencacah perulangan juga dapat berupa karakter. Perhatikan contoh program dibawah ini:

Program 4.9

```
#include<conio.h>
#include <iostream.h>

using namespace std;

int main()
{
 char huruf;
 for (huruf = 'A'; huruf <= 'Z'; huruf++)
      cout << "Huruf abjad = " << huruf << "\n";
 getch();
 for (huruf = 'A'; huruf <= 'Z'; huruf+=13)
      cout << "Huruf abjad = " << huruf << "\n";
 getch();
 for (huruf = 'z'; huruf >= 'a'; huruf--)
      cout << "Huruf abjad = " << huruf << "\n";
 getch();
```

```
for (huruf = 'z'; huruf >= 'a'; huruf-=8)
      cout << "Huruf abjad = " << huruf << "\n";
getch();
return 0;
}
```

Hal yang perlu diperhatikan pada program diatas adalah potongan program seperti dibawah ini:

```
for (huruf = 'A'; huruf <= 'Z'; huruf++)
      cout << "Huruf abjad = " << huruf << "\n";
```

Perintah di atas akan menampilkan teks Huruf abjad = mulai dari A sampai dengan Z. Perhatikan perintah pada **huruf++.**

```
Huruf abjad = A
Huruf abjad = B
Huruf abjad = C
Huruf abjad = D
Huruf abjad = E
Huruf abjad = F
Huruf abjad = G
Huruf abjad = H
Huruf abjad = I
Huruf abjad = J
Huruf abjad = K
Huruf abjad = L
Huruf abjad = M
Huruf abjad = N
Huruf abjad = O
Huruf abjad = P
Huruf abjad = Q
Huruf abjad = R
Huruf abjad = S
Huruf abjad = T
Huruf abjad = U
Huruf abjad = V
Huruf abjad =W
Huruf abjad = X
Huruf abjad = Y
Huruf abjad = Z
```

Perhatikan potongan program dibawah ini:

```
for (huruf = 'A'; huruf <= 'Z'; huruf+=13)
          cout << "Huruf abjad = " <<
huruf << "\n";
```

Perintah diatas akan menampilkan teks Huruf abjad = mulai dari A sampai dengan Z dengan step 13, maka yang muncul adalah:

```
Huruf abjad = A
Huruf abjad = N
```

Perhatikan pada instruksi **huruf+=13**

```
for (huruf = 'z'; huruf >= 'a'; huruf--)
cout << "Huruf abjad = " << huruf <<
"\n";
```

Perintah tersebut akan menampilkan teks Huruf abjad = mulai dari z sampai dengan a. Perhatikan pada perintah **huruf--**

```
Huruf abjad = z
Huruf abjad = y
Huruf abjad = x
Huruf abjad = w
Huruf abjad = v
Huruf abjad = u
Huruf abjad = t
Huruf abjad = s
Huruf abjad = r
Huruf abjad = q
```

```
Huruf abjad = p
Huruf abjad = o
Huruf abjad = n
Huruf abjad = m
Huruf abjad = l
Huruf abjad = k
Huruf abjad = j
Huruf abjad = i
Huruf abjad = h
Huruf abjad = g
Huruf abjad = f
Huruf abjad = e
Huruf abjad = d
Huruf abjad = c
Huruf abjad = b
Huruf abjad = a
```

Perhatikan potongan program dibawah ini:

```
for (huruf = 'z'; huruf >= 'a'; huruf-=8)
cout << "Huruf abjad = " << huruf <<
"\n";
```

Perintah itu akan menampilkan teks Huruf abjad = mulai dari z sampai dengan a dengan step -8, maka yang muncul adalah:

```
Huruf abjad = z
Huruf abjad = r
Huruf abjad = j
Huruf abjad = b
```

Perhatikan instruksi huruf -=8. Kadang-kadang dijumpai adanya pernyataan FOR yang tidak mengandung bagian ungkapan yang lengkap (beberapa ungkapan dikosongkan). Dengan cara ini, maka pernyataan adalah sebagai berikut:

```
for (loop = 1; loop <= 10; loop+=2)
      cout << loop << " ";
```

Dari potongan program diatas dapat ditulis menjadi :

```
loop = 1;
for ( ; loop <= 10; )
{
      cout << loop << " ";
      loop += 2;
}
```

Pengosongan juga dilakukan pada ungkapan untuk menaikkan nilai variabel pengendali loop. Sebagai gantinya, di dalam tubuh loop diberikan pernyataan untuk menaikkan nilai variabel pengendali loop, yaitu berupa:

```
loop += 2;
```

Ungkapan yang tidak dihilangkan berupa loop <=10. Ungkapan ini tetap disertakan karena dipakai sebagai kondisi untuk keluar dari loop. Sebenarnya ungkapan yang dipakai sebagai kondisi keluar dari loop juga bisa dihilangkan, sehingga bentuknya menjadi:

```
for (;;)
   pernyataan;
```

Lalu bagaimana cara keluar dari loop tersebut. Caranya adalah dengan menggunakan pernyataan yang dirancang khusus untuk keluar dari loop. Pernyataan itu adalah break yang akan dijelaskan dalam sub bahasan tersendiri. Pernyataan break digunakan untuk keluar dari satu blok loop for. Perhatikan program FOR untuk menampilkan warna seperti program berikut:

Program 4.10

```
#include<stdio.h>
#include<conio.h>

using namespace std;

int main(void)
{
      int i = 0;
      for (;;)
      {
          i++;
          textattr(i + ((i+1) << 4));
          cprintf("Warna atribut\r\n");
          if (i==20) break;
      }
      getch();
      return 0;
}
```

Keluaran program diatas adalah sebagai berikut:

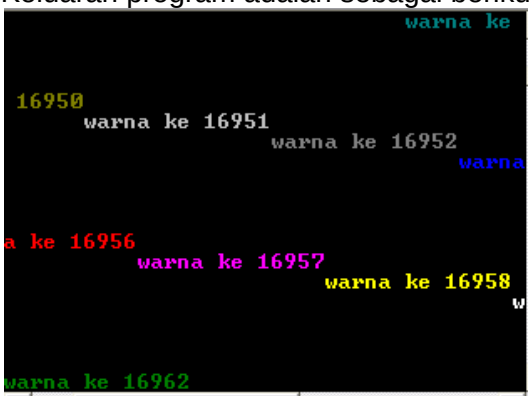
Pada contoh berikut merupakan penggunaan loop tanpa henti. Karena program memang dirancang untuk tidak berhenti maka untuk menghentikannya dilakukan dengan cara menekan tombol CTRL–PAUSE atau CTRL – BREAK. Perhatikan program FOR untuk menampilkan warna seperti berikut:

Program 4.11

```
#include<stdio.h>
#include<conio.h>
```

```
using namespace std;

main()
{
 int n = 0;
 for(;;)
 {
          ++n;
          gotoxy(n, n);
          textcolor(n);
          cprintf("warna ke %d\n",n);
 }
 getch();
 return 0;
}
```

Keluaran program adalah sebagai berikut:



4.6. Pernyataan NESTED - FOR

Pernyataaan nested for adalah suatu perulangan for didalam perulangan for dalam bentuk lain. Dalam mempelajari perulangan ini dituntut hrus teliti. Perulangan didalam perulangan sering kali masuk digunakan oleh program Bentuk umum pernyataan Nested for adalah sebagai berikut:

```
for ( inisialisasi; syarat pengulangan; pengubah nilai pencacah )
{
     for ( inisialisasi; syarat pengulangan; pengubah nilai pencacah)
     {
         pernyataan;
     }
}
```

Selain pernyataan diatas, nested For dapat juga ditulis seperti dibawah ini:

```
for ( inisialisasi; syarat pengulangan; pengubah nilai pencacah )
{
      for ( inisialisasi; syarat pengulangan; pengubah nilai pencacah)
      {
            for ( inisialisasi; syarat pengulangan; pengubah nilai pencacah)
            {
             ……..
            for ( inisialisasi; syarat pengulangan; pengubah nilai pencacah)
            {
            pernyataan;
      }
      ……..
}
```

Didalam penggunaan NESTED-FOR, perulangan yang berada didalam terlebih dahulu harus dihitung sampai selesai, kemudian perulangan yang diluar diselesaikan terus sampai perulangan yang paling luar. Perhatikan contoh program dengan NESTED FOR pada operasi bilangan naik berikut ini:

Program 4.12

```
#include<conio.h>
#include <iostream.h>

using namespace std;

int main()
{
          int a, b;
          for(a = 1; a <= 5; a++)
          {
                    cout << "\n ";
                    for(b = a; b <= 5; b++)
                    cout << a << " ";
          }
          getch();
          return 0;
}
```

Keluaran program diatas adalah sebai berikut:

```
1 1 1 1 1
2 2 2 2
```

```
3 3 3
4 4
5
```

4.7. Pernyataan WHILE

Bab sebelumnya telah memperkenalkan konsep statement kontrol, yang berupa aliran program langsung. Sebuah loop adalah struktur kontrol yang menyebabkan pernyataan atau kelompok untuk mengulang pernyataan. Bahasa C++ mempunyai tiga looping struktur kontrol: while loop, do-while loop, dan loop FOR.

Perbedaan antar masing-masing looping adalah bagaimana mereka melakukan kontrol pengulangannya. Selama melakukan loop ada dua bagian penting: (1) sebuah ekspresi yang diuji dengan nilai benar atau salah, dan (2) pernyataan atau blok yang berulang-ulang selama ekspresinya benar.

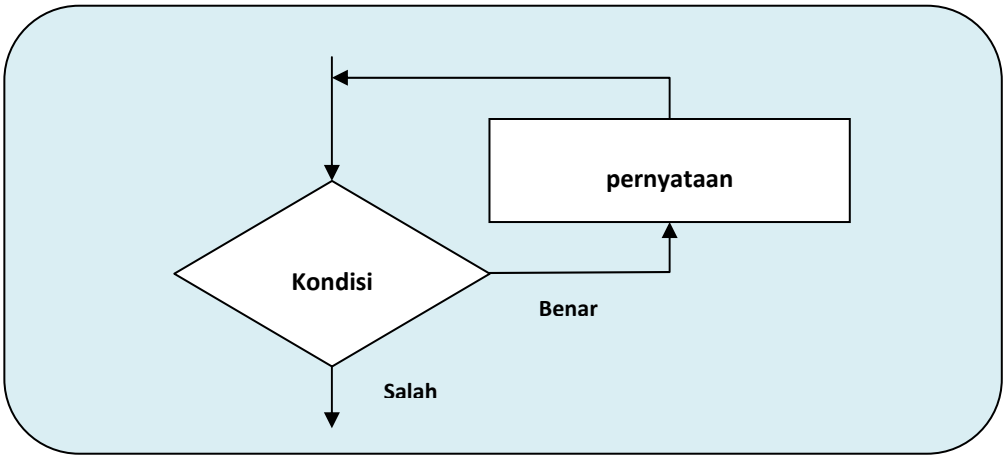
Perulangan dengan pernyataan **while** merupakan perulangan yang mirip dengan perulangan **for**. Perulangan for dipakai pada perulangan yang sudah diketahui berapa kali akan dijalankan. Sedangkan yang belum diketahui berapakali akan diulangi maka digunakan while. Pernyataan while digunakan ketika kondisi perulangan diperiksa terlebih dahulu sebelum menjalankan pernyataan. Pada pernyataan while, disini pemeriksaan terhadap loop dilakukan di bagian awal (sebelum tubuh loop). Pernyataan while akan mengulang proses secara terus menerus sampai kondisi bernilai benar atau akan diulangi selama kondisi bernilai benar, jika kondisi bernilai salah maka perulangan (loop) selesai. Lebih jelasnya, bentuk pernyataan while adalah sebagai berikut:

Bentuk perulangan **while** dikendalikan oleh syarat/kondisi tertentu, yaitu perulangan akan terus dilaksanakan selama syarat/kondisi tersebut terpenuhi. Pernyataan dalam while akan dilaksanakan berulangkali selama syarat/kondisi bernilai benar. Jika syarat/kondisi bernilai salah badan perulangan tidak akan dilaksanakan, yang berarti perulangan selesai. Yang harus diperhatikan adalah perulangan harus berhenti. Perulangan yang tidak pernah berhenti menandakan bahwa logika dari algoritma tersebut salah.

Bentuk umum perulangan **while**, sebagai berikut :

```
while ( syarat/kondisi )
pernyataan;
```

penjelasan mengenai hal tersebut diatas adalah bahwa syarat/kondisi merupakan ungkapan logika yang hanya bernilai benar atau salah, sehingga operator yang dipakai disini adalah operator relasi dan operator logika atau gabungan dari keduanya. Untuk lebih jelasnya gambar diagram alir dari pernyataan while adalah sebagai berikut:



Gambar 4.3. Diagram pernyataan While

Pernyataan dalam **while** dapat berupa pernyataan tunggal maupun jamak (lebih dari satu). Jika pernyataannya berbentuk jamak, maka pernyataan-pernyataan tersebut harus diletakan didalam satu blok dengan memakai tanda kurung kurawal.

Bentuk umum perulangan **while**, dengan satu dengan lebih dari satu pernyataan, adalah seperti berikut :

```
while ( syarat )
{
  Pernyataan;
  Pernyataan;
}
```

Untuk lebih jelasnya pengenai pernyataan while, dibawah ini merupakan contoh program untuk menaik bilangan. Program lengkapnya adalah sebagai berikut:

Program 4.13

```
#include<conio.h>
#include <iostream.h>

using namespace std;

int main()
{
      int loop = 1;
      while(loop <= 10)
      cout << loop++ << " ";
      getch();
      return 0;
}
```

Keluaran program adalah sebagai berikut:

```
1 2 3 4 5 6 7 8 9 10
```

Program dibawah ini merupakan sebuah penerapan pernyataan while yang digunakan untuk melakukan perhitungan turun. Program dibawah ini akan menghitung bilangan turun mulai dari angka 10, menjadi 9, 8, 7 sampai angka 1. Perhatikan program dibawah ini:

Program 3.14

```
#include<conio.h>
#include <iostream.h>

using namespace std;

int main()
{
      int loop = 10;
      while(loop >= 1)
      cout << loop-- << " ";
      getch();
      return 0;
}
```

Keluaran program diatas adalah sebagai berikut:

```
10 9 8 7 6 5 4 3 2 1
```

Sebuah program yang menggunakan pernyataan while juga bisa digunakan untuk menentukan bilangan ganjil. Program dibawah ini merupakan program yang keluaranya bilangan 1, 3, 5 dan seterusnya sampai batasan looping yang dimasukan. Batasan program dibawah ini adalah 10, untuk lebih jelasnya perhatikan program dibawah ini:

Program 4.15

```
#include<conio.h>
#include <iostream.h>

using namespace std;

int main()
{
   int loop = 1;
   while(loop <= 10)
   {
       cout << loop << " ";
```

```
        loop+=2;
    }
    getch();
    return 0;
}
```

Keluaran program diatas adalah:

```
1 3 5 7 9
```

Program dibawah ini penggunaan while untuk aplikasi manampilkan huruf. Huruf yang ditampilkan adalah huruf abjad. Untuk lebih jelasnya perhatikan program dibawah ini:

Program 4.16

```
#include<conio.h>
#include <iostream.h>

using namespace std;

int main()
{
        char huruf = 'A';
        while(huruf <= 'Z')
        cout << "Huruf abjad = " << huruf++ << "\n";
        getch();
        huruf = 'A';
        while(huruf <= 'Z')
        {
            cout << "Huruf abjad = " << huruf << "\n";
            huruf+=13;
        }
        getch();
        huruf = 'z';
        while(huruf >= 'a')
        cout << "Huruf abjad = " << huruf-- << "\n";
        getch();
        huruf = 'z';
        while(huruf >= 'a')
        {
            cout << "Huruf abjad = " << huruf << "\n";
            huruf-=8;
        }
        getch();
        return 0;
```

```
}
```

Keluaran program adalah sebagai berikut:

```
Huruf abjad = A
Huruf abjad = B
Huruf abjad = C
Huruf abjad = D
Huruf abjad = E
Huruf abjad = F
Huruf abjad = G
Huruf abjad = H
Huruf abjad = I
Huruf abjad = J
Huruf abjad = K
Huruf abjad = L
Huruf abjad = M
Huruf abjad = N
Huruf abjad = O
Huruf abjad = P
Huruf abjad = Q
Huruf abjad = R
Huruf abjad = S
Huruf abjad = T
Huruf abjad = U
Huruf abjad = V
Huruf abjad =W
Huruf abjad = X
Huruf abjad = Y
Huruf abjad = Z
```

4.8. Pernyataan NESTED-WHILE

Pernyataaan nested while adalah suatu perulangan while didalam perulangan while yang lainnya. Bentuk umum pernyataan Nested while sebagai berikut :

```
while (syarat)
{
    while (syarat)
    {
        pernyataan;
    }
}
```

Selain cara penulisan pernyataan ditulis diatas, dapat juga ditulis sebagai berikut:

```
while (syarat)
{
       while (syarat)
      {
          ……..
          while (syarat)
         {
            pernyataan;
          }
          ……..
      }
}
```

Didalam penggunaan NESTED-WHILE, perulangan yang berada didalam terlebih dahulu dihitung hingga selesai, kemudian perulangan yang diluar diselesaikan terus sampai perulangan yang paling luar. Perhatikan Contoh program dengan nested while dimana program tersebut merupakan program Segitiga Pascal dengan NESTED WHILE seperi berikut ini.

Program 4.17

```
#include<conio.h>
#include <iostream.h>
#include <math.h>

using namespace std;

int main(void)
      {
       int row = 0;
       while(row<=3)
      {
          int col = 1;
          while(col <= 3-row)
          {
              cout << "  ";
              col++;
          }
          col=0;
          while(col<=row)
```

```
        {
            cout << " " << (int)pow(2, col);
            col++;
        }
        col= row - 1;
        while (col>= 0)
        {
            cout << " " << (int)pow(2, col);
            col--;
        }
        cout << "\n";
        row++;
    }
    getch();
    return 0;
}
```

Keluaran program diatas adalah sebagai berikut:

```
      1
    1 2 1
  1 2 4 2 1
1 2 4 8 4 2 1
```

Contoh program dibawah ini digunakan untuk menjumlahkan sejumlah data angka. Angka yang akan dijumlahkan dimasukan satu-persatu. Proses pemasukan data angka akan berhenti ketika dimasukkan angka –1. Setelah itu tampil hasil penjumlahannya.

Program 4.18

```
#include <iostream.h>
#include <conio.h>

using namespace std;
void main()
    {
    int cacah = 0, data = 0, jumlah = 0;
    while (data != -1)
        {
            cout << "Masukkan data angka : ";
            cin >> data;
            jumlah += data;
            cacah++;
        }
    cout << "Jumlah data adalah : " << jumlah" << endl;
```

```
    cout << "Rata-rata : " << jumlah/cacah;
}
```

4.9. Perulangan DO-WHILE

Selain operasi loop menggunakan WHILE, bahasa C++ juga mempunyai DO-WHILE dan untuk perulangan. Setiap perulangan dapat tepat untuk mengatasi berbagai masalah pemrograman. Yang dilakukan DO-WHILE terlihat mirip dengan perulangan yang berpaling terbalik.

Perulangan dengan pernyataan do-while merupakan perulangan yang mirip dengan perulangan while ataupun for. Perulangan for dipakai pada perulangan yang sudah diketahui berapa kali akan dijalankan. Sedangkan yang belum diketahui berapakali akan diulangi maka digunakan while atau do-while. Pernyataan do-while digunakan untuk menjalankan pernyataan terlebih dahulu baru kemudian memeriksa kondisi perulangan.

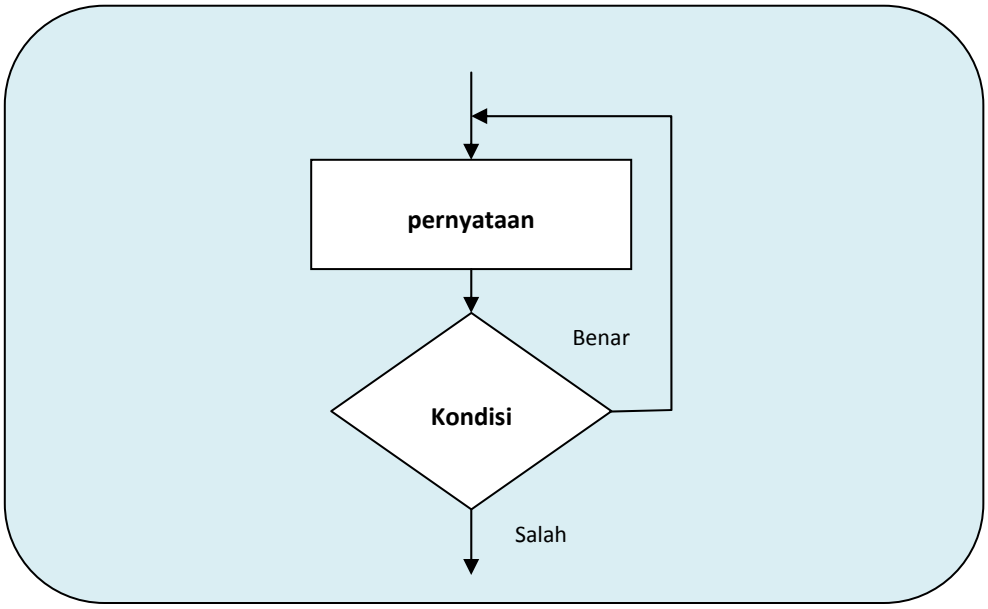
Pada pernyataan do-while, disini pemeriksaan terhadap loop dilakukan di bagian akhir (setelah tubuh loop). Pernyataan do-while akan mengulang proses secara terus menerus selama kondisi bernilai benar dan perulangan (loop) selesai jika kondisi bernilai salah.

Bentuk perulangan do-while dikendalikan oleh syarat/kondisi tertentu, yaitu perulangan akan terus dilaksanakan selama syarat/kondisi tersebut terpenuhi. Pernyataan dalam do-while akan dilaksanakan berulangkali selama syarat/kondisi bernilai benar. Jika syarat/kondisi bernilai salah badan perulangan tidak akan dilaksanakan, yang berarti perulangan selesai. Yang harus diperhatikan adalah perulangan harus berhenti. Perulangan yang tidak pernah berhenti menandakan bahwa logika dari algoritma tersebut salah.

Perbedaan dengan while sebelumnya yaitu bahwa pada do-while statement perulangannya dilakukan terlebih dahulu baru kemudian di cek kondisinya. Sedangkan while kondisi dicek dulu baru kemudia statement perulangannya dijalankan. Akibat dari hal ini adalah dalam do-while minimal terdapat sekali perulangan. Sedangkan while dimungkinkan perulangan tidak pernah terjadi yaitu ketika kondisinya langsung bernilai salah. Bentuk umum perulangan do-while, sebagai berikut :

```
do
    pernyataan;
while ( syarat/kondisi )
```

Syarat/kondisi: merupakan ungkapan logika yang hanya bernilai benar atau salah, sehingga operator yang dipakai disini adalah operator relasi dan operator logika atau gabungan dari keduanya. Gambar diagram alir dari pernyataan do-while adalah sebagai berikut:



Gambar 4.4. Diagram alir pernyataan do-while

Pernyataan dalam do-**while** dapat berupa pernyataan tunggal maupun jamak (lebih dari satu). Jika pernyataannya berbentuk jamak, maka pernyataan-pernyataan tersebut harus diletakan didalam satu blok dengan memakai tanda kurung kurawal. Bentuk umum perulangan do-**while**, dengan lebih dari satu pernyataan, seperti berikut dibawah ini:

```
do
{
      Pernyataan;
      Pernyataan;
}
while (syarat)
```

program dibawah ini merupakan aplikasi dengan menggunakan do-while pada operasi bilangan naik. Program dengan do-while dibawah ini merupakn program do-while bilangan naik. Untuk lebih jelasnya perhatikan contoh program berikut:

Program 4.19

```
#include<conio.h>
#include <iostream.h>
```

```
using namespace std;

int main()
{
    int loop = 1;
    do
    cout << loop++ << " ";
    while(loop <= 10);
    getch();
    return 0;
}
```

Hasil keluaran program adalah:

```
1 2 3 4 5 6 7 8 9 10
```

Seperti halnya program diatas, pernyataan do-while juga bisa digunakan untuk operasi bilangan turun. Program dibawah ini digunakan untuk menurunkan bilangan mulai dari 10 kemudian turun menjadi 9, 8 dan seterusnya. Untuk lebih jelasnya perhatikan program berikut ini:

Program 4.20

```
#include<conio.h>
#include <iostream.h>

using namespace std;

int main()
{
    int loop = 10;
    do
    cout << loop-- << " ";
    while (loop >= 1);
    getch();
    return 0;
}
```

Hasil keluaran program adalah:

```
10 9 8 7 6 5 4 3 2 1
```

Selain digunakan untuk menaikan dan menurunkan bilangan, dibawah ini juga diberikan contoh sebuah program yang menggunakan pernyataan perulangan do-while yang menampilkan bilangan ganjil.

Bilangan yang ditampilkan adalah bilangan ganjil dengan urutan mulai dari satu sampai batas yang ditentukan yaitu 10. Sehingga bilangan yang keluar dari program adalah 1, 3, 5, 7, dan 9. Untuk lebih jelasnya perhatikan program dibawah ini:

Program 4.21

```
#include<conio.h>
#include <iostream.h>

using namespace std;

int main()
    {
    int loop = 1;
    do
     {
       cout << loop << " ";
       loop+=2;
     }
    while (loop <= 10);
    getch();
    return 0;
}
```

Hasil keluaran program diatas adala sebagai berikut:

```
1 3 5 7 9 _
```

4.10. Pernyataan NESTED DO-WHILE

Pernyataaan nested do-while adalah suatu perulangan do-while didalam perulangan do-while lainnya. Bentuk umum pernyataan Nested do-while sebagai berikut :

```
do
{
      do
      {
          pernyataan;
      }
      while (syarat);
}
while (syarat);
```

pernyataan diatas dapat juga ditulis seperti potongan pernyataan dibawah ini:

```
do
{
 do
 {
   ……..
        do
        {
         pernyataan;
         }
        while (syarat);
   . ……..
  }
  while (syarat);
}
while (syarat);
```

Didalam penggunaan nested do-while, perulangan yang didalam terlebih dahulu dihitung hingga selesai, kemudian perulangan yang diluar diselesaikan terus sampai perulangan yang paling luar.

Program perulangan nested do-while sebenarnya lebih komplek dibandingkan program do while biasa. Dalam mengerjakan program nested, baik while, do-while tentunya harus lebih teliti, karena jika tidak seringkali terjadi kesalahan ketika de-compile. Program dibawah ini merupakan contoh aplikasi nested do-while yang digunakan untuk mencari Segitiga Pascal. Untuk lebih jelasnya perhatikan contoh program dibawah ini:

Program 4.22

```
#include<conio.h>
#include <iostream.h>
#include <math.h>
#include<stdlib.h>

using namespace std;

int main(void)
{
 int row = 0;
 do
 {
  int col = 0;
  do
  {
          cout << "  ";
          col++;
```

```
 }
 while(col <= 3-row);
 col=0;
 do
 {
          cout << " " << (int)pow(2, col);
          col++;
 }
 while(col<=row);
 col = row - 1;
 do
 {
          cout << " " << (int)pow(2, col);
          col--;
 }
 while (col>= 0);
 cout << "\n";
 row++;
}
while(row<=3);
getch();
return 0;
}
```

Hasil keluaran program adalah:

```
          1 0
        1 2 1
      1 2 4 2 1
    1 2 4 8 4 2 1
```

4.11. Perulangan Tidak Berhingga

Perulangan tidak berhingga merupakan perulangan (loop) yang tak pernah berhenti atau mengulang terus, hal ini sering terjadi disebabkan adanya kesalahan penanganan kondisi yang dipakai untuk keluar dari loop. Bahkan suatu ketika program perulangan ini memang dirancang untuk tidak pernah berhenti, walaupun hal ini sangat jarang sekali. Perulangan tidak terhingga biasanya karena adanya kesalahan penulisan program, sehingga ketika program dijalankan akan berjalan terus-menerus,.

Sebagai contoh saja pada program yang digunakan untuk penulisan perintah nilai pencacah salah seperti pada program berikut:

Program 4.23

```
#include<conio.h>
#include <iostream.h>

using namespace std;

int main()
{
      int b;
      for(b = 6; b >= 1; b++)
          cout << b;
      getch();
      return 0;
}
```

Keluaran program diatas adalah sebagai berikut:

Pada program tersebut diatas tidak akan berhenti sampai dilakukan penghentian dengan paksa. Pada pernyataan for diatas tidak akan berhenti untuk menampilkan bilangan menaik, kesalahan terjadi pada pengubah nilai pencacah, seharusnya penulisan yang benar adalah:

```
b –
```

Akan tetapi yang ditulis dalam program adalah :

```
b ++
```

Oleh karena kondisi b >= 1 selalu bernilai benar (karena b bernilai 6), maka pernyataan `cout << b;` akan terus dijalankan. Jika terjadi hal semacam ini, untuk menghentikan proses yang terus menerus semacam ini dilakukan dengan menekan

tombol CTRL+ PAUSE atau CTRL + BREAK.

4.12. Pernyataan Break

Kadang-kadang perlu untuk menghentikan satu looping sebelum meninggalkan semua Iterasi. Pernyataan break, dapat digunakan untuk beralih pada didalam satu perulangan. Bila menemui break , perulangan berhenti dan program melompat ke pernyataan setelah perulangan.

Perulangan dengan while pada segmen program untuk menjalankan 10 kali, namun pernyataan break dapat menyebabkannya berhenti setelah kelima perulangan atau perulangan tertentu. Untuk lebih jelasnya perhatikan potongan program dibawah ini:

```
int count = 0;
while (count++ < 10)
{
    cout << count << endl;
    if (count == 5)
    break;
}
```

Pernyataan break berfungsi untuk keluar dari perulangan baik for, while dan do-while serta struktur switch. Jika pernyataan break dikerjakan, maka eksekusi akan dilanjutkan ke pernyataan yang terletak sesudah akhir dari badan perulangan (loop).

Perhatikan contoh program pemakaian break dalam pernyataan for dibawah. Program digunakan untuk menghentikan bilangan deret menggunakan break

Program 2.24

```
#include<conio.h>
#include <iostream.h>

using namespace std;

int main(void)
{
    int jumlah= 0;
    int bilangan;
    for (bilangan=0;bilangan < 20;bilangan++)
    {
        jumlah += bilangan;
        if (jumlah >= 100) break;
    }
    cout << "Deret Bilangan : 1 + 2 + ... + " << bilangan << endl;
    cout << "Jumlah Deret Bilangan = " <<jumlah;
```

```
    getch();
    return 0;
}
```

Hasil keluaran program diatas adalah sebagai berikut:

```
Deret Bilangan : 1 + 2 + …… + 14
Jumlah Deret Bilangan = 105
```

Program dibawah ini menggunakan break dengan pernyataan while. Program deret bilangan yang menggunaan break adalah sebagai berikut:

Program 4.25

```
#include<conio.h>
#include <iostream.h>

using namespace std;

int main(void)
    {
     int jumlah= 0;
     int bilangan = 0;
     while (bilangan < 20)
     {
          bilangan++;
          jumlah += bilangan;
          if (jumlah >= 100) break;
     }
     cout <<  "Deret Bilangan : 1 + 2 + ... + " << bilangan << endl;
     cout << "Jumlah Deret Bilangan = " <<jumlah;
     getch();
     return 0;
}
```

Hasil keluaran program:

```
Deret Bilangan : 1 + 2 + …… + 14
Jumlah Deret Bilangan = 105
```

Contoh program dibawah ini menggunakan break dengan pernyataan do-while. Perhatikan program deret bilangan dengan menggunakan break berikut ini:

Program 4.26

```
#include<conio.h>
#include <iostream.h>

using namespace std;

int main(void)
{
    int jumlah= 0;
    int bilangan = 0;
    do
    {
       bilangan++;
       jumlah += bilangan;
       if (jumlah >= 100) break;
    }
    while (bilangan < 20);
    cout <<  "Deret Bilangan : 1 + 2 + ... + " << bilangan << endl;
    cout << "Jumlah Deret Bilangan = " <<jumlah;
    getch();
    return 0;
}
```

Hasil keluaran program diatas adalah sebagai berikut:

```
Deret Bilangan : 1 + 2 + …… + 14
Jumlah Deret Bilangan = 105
```

4.13. Pernyataan Continue

Pernyataan continue digunakan untuk mengarahkan eksekusi ke iterasi (proses loop) berikutnya yang berada pada loop yang sama, atau dengan kata lain mengembalikan proses yang sedang dilaksanakan keawal loop lagi, tanpa menjalankan sisa perintah dalam loop tersebut. Perhatikan contoh program yang menggunakan continue dalam pernyataan for berikut ini:

Program 4.27

```
#include<conio.h>
#include <iostream.h>

using namespace std;

int main(void)
{
```

```
    int jumlah= 0;
    int bilangan;
    for(bilangan = 1; bilangan <= 20;bilangan++)
    {
        if (bilangan == 10 || bilangan == 11) continue;
        jumlah += bilangan;
    }
    cout <<  "Deret Bilangan : 1 + 2 + ... + 9 + 12 + 13 + ... + " << bilangan-1 << endl;
    cout << "Jumlah Deret Bilangan = " <<jumlah;
    getch();
    return 0;
}
```

Keluaran program diatas adalah sebagai berikut:

```
Deret Bilangan : 1 + 2 + ..... + 9 + 12 + 13 + ... + 20
Jumlah Deret Bilangan = 189
```

Perhatikan contoh program yang menggunakan continue dengan pernyataan while sebagai berikut ini:

Program 4.28

```
#include<conio.h>
#include <iostream.h>

using namespace std;

int main(void)
{
    int jumlah= 0;
    int bilangan = 0;
    while (bilangan < 20)
    {
        bilangan++;
        if (bilangan == 10 || bilangan == 11) continue;
        jumlah += bilangan;
    }
    cout <<  "Deret Bilangan : 1 + 2 + ... + 9 + 12 + 13 + ... + " << bilangan << endl;
    cout << "Jumlah Deret Bilangan = " <<jumlah;
    getch();
    return 0;
}
```

Hasil keluaran program:

```
Deret Bilangan : 1 + 2 + ..... + 9 + 12 + 13 + ... + 20
```

```
Jumlah Deret Bilangan = 189
```

Berikut ini merupakan contoh program menggunakan continue dalam sebuah pernyataan do-while. Perhatikan program bilangan deret dengan mengguakan continue seperti dibawah:

Program 4.29

```
#include<conio.h>
#include <iostream.h>

using namespace std;

int main(void)
{
    int jumlah= 0;
    int bilangan = 0;
    do
    {
        bilangan++;
        if (bilangan == 10 || bilangan == 11) continue;
        jumlah += bilangan;
    }
    while (bilangan < 20);
    cout <<  "Deret Bilangan : 1 + 2 + ... + 9 + 12 + 13 + ... + " << bilangan << endl;
    cout << "Jumlah Deret Bilangan = " <<jumlah;
    getch();
    return 0;
}
```

Hasil keluaran program:

```
Deret Bilangan : 1 + 2 + …… + 9 + 12 + 13 + … + 20
Jumlah Deret Bilangan = 189
```

4.14. Pernyataan Goto

Pernyataan goto merupakan instruksi untuk mengarahkan eksekusi program menuju pernyataan yang diawali dengan suatu label. Label merupakan suatu pengenal (identifier) yang diikuti dengan tanda titik dua (:).

Bentuk pemakaian goto adalah sebagai berikut:

```
goto label;
```

Contoh penggunaan instruksi goto dapat dilihat pada contoh program berikut ini:

Program 4.30

```
#include<iostream.h>
#include<stdio.h>
#include<conio.h>

using namespace std;

main()
{
    int a, b;
    char lagi;
    atas:
    cout << "Masukkan Bilangan = ";
    cin >> a;
    b = a % 2;
    cout <<"Nilai " << a << " % 2 adalah = " << b;
    cout << "\n\nIngin Hitung Lagi [Y/T] : ";
    lagi = getche() ;
    if (lagi == 'Y' || lagi == 'y')goto atas;
    getch();
    return 0;
}
```

Keluaran program tersebut diatas adalah:

```
Masukan Bilangan      =
Nilai 5 % 2 adalah    = 1
Ingin Hitung lagi (Y/T)   : t
```

4.15. Soal Latihan

Jawablah soal latihan dibawah ini dengan baik dan benar.

1. Struktur perulangan secara umum terdiri dari dua bagian, sebutkan
2. Sebutkan fungsi operator increment dan decrement
3. Jelaskan bentuk pernyataan for dalam bahasa pemrograman
4. Jelaskan apa yang dimaksud dengan potongan program dibawah ini:

```
int num = 1;
for ( ; num <= 10; )
{
      cout << num << "\t\t" << (num * num) << endl;
      num++;
}
```

5. Sebutkan fungsi break
6. Sebutkan fungsi goto
7. Apa yang dimaksud dengan NESTED-WHILE

BAB 5
STATEMENT KENDALI

5.1. Pengertian Statement

Penyeleksian kondisi digunakan untuk mengarahkan perjalanan suatu proses. Penyeleksian kondisi dapat diibaratkan sebagai katup atau kran yang mengatur jalannya air. Bila katup terbuka maka air akan mengalir dan sebaliknya bila katup tertutup air tidak akan mengalir atau akan mengalir melalui tempat lain. Seleksi kondisi adalah proses penentuan langkah berikutnya berdasarkan proses yang terjadi sebelumnya.

Seleksi kondisi ini sangat penting dalam pemrograman sebab dengan adanya seleksi kondisi, program dapat menentukan proses apa yang harus dilakukan selanjutnya berdasarkan keadaan sebelumnya. Sehingga nampak seolah olah program dapat berpikir dan mengambil keputusan. Disinilah letak kekurangan komputer yaitu tidak mampu berpikir sendiri, semua hal yang dilakukan adalah berdasarkan perintah.

Dalam memprogram seringkali digunakan suatu percabangan untuk pengambilan keputusan dari sejumlah pilihan yang mungkin. Bahasa pemrograman menyediakan pernyataan IF…THEN dan kata kunci yang lain seperti SWITCH...CASE untuk melakukan suatu percabangan.

Dalam percabangan, keputusan diambil berdasarkan ekspresi kondisi. Ekpresi berkondisi adalah sebagian dari pernyataan program yang menanyakan pertanyaan True atau False (Benar atau Salah) mengenai

properti, variabel, atau data lain pada kode program.

Kode-kode perintah yang diberikan dari suatu bahasa pemrograman untuk melakukan suatu proses. Blok merupakan satu atau lebih statemen yang harus dikerjakan setelah suatu kondisi dipenuhi.

Pernyataan Percabangan digunak-an untuk memecahkan persoalan untuk mengambil suatu keputusan diantara sekian pernyataan yang ada. Untuk keperluan pengambilan keputusan, Borland C++ menyediakan beberapa perintah antara lain.

5.2. Operator Relasi

Dalam melakukan pemrograman, kebanyakan programmer tentunya sudah banyak tahu bahwa program-program komputer dibuat supaya mengikuti alur. Tentunya kita juga berfikir bagaimana jika komputer diprogram ternyata tidak mengikuti alur atau urutan yang telah dibuat?

Maka komputer tersebut akan tidak bisa berjalan sesuai dengan yang kita inginkan. Alur atau urutan pada komputer ini antara lain:

- Adanya masukan dari pengguna.
- Dapat melakukan Satu atau lebih perhitungan atau proses.
- Menampilkan hasilnya pada layar.

Sebuah komputer yang baik, selain dapat melakukan perhitungan, tetapi juga sangat ahli dalam membandingkan sebuah nilai untuk menentukan apakah lebih besar, kurang dari, atau sama dengan. Jenis operasi tersebut sangat berharga untuk memeriksa tugas-tugas seperti angka-angka penjualan, penentuan laba rugi, memeriksa untuk memastikan angka tersebut dalam rentang yang dapat diterima, dan memvalidasi input yang diberikan oleh pengguna.

Data Numerik pada bahasa C++ akan dibandingkan dengan menggunakan sebuah operator. Sebuah karakter juga dapat dibandingkan dengan menggunakan operator, karena dianggap sebagai karakter yang mempunyai nilai numerik pada C++. Penghubung pada setiap operator akan menentukan apakah ada hubungan antara dua nilai. Sebagai contoh, operator lebih (>) menentukan jika nilai lebih besar daripada yang lain. Operator kesetaraan (==) menentukan apakah dua nilai sama. Tabel dibawah merupakan semua operator penghubung pada bahasa C++.

Table 5.1. Operator Relasional

RELASI	OPERATOR
>	Lebih besar daripada
<	Lebih kecil daripada
>=	Lebih besar dari sama dengan
<=	Kurang dari samadengan
==	Sama dengan
!=	Tidak sama dengan

Semua operator relasional tersebut diatas, bisa juga disebut dengan operator biner. Hal ini karena mereka menggunakan dua operand. Berikut adalah contoh yang menggunakan operator ekspresi yang lebih besar daripada:

```
x > y
```

Ekspresi tersebut merupakan ekspresi penghubung atau *relational expression*. Digunakan untuk menentukan apakah x lebih besar daripada y. Berikut ekspresi untuk menentukan jika x kurang dari y:

```
x <y
```

Selanjutnya bagaimana menentukan nilai sebuah operator penghubung tersebut dan bagaimana relational expression digunakan dalam program? Untuk menjawah hal tersebut ada yang perlu diingat yaitu: semua memiliki nilai. Relational ekspresi adalah ekspresi Boolean, yang berarti mereka hanya terdapat nilai benar atau salah. Jika x lebih besar dari y, maka ekspresi x> y akan benar, sedangkan kalimatnya dapat ditulis sebgai berikut:

```
Y == X
```

Pernyataan diatas akan akan salah jika, operator == digunakan untuk menentukan apakah operand sebelah kiri adalah sama dengan operand di kanan. Jika kedua operand memiliki nilai yang sama, ungkapan itu benar. Dengan asumsi bahwa a adalah 4 merupakan ungkapan yang benar:

```
a == 4
```

Namun berikut ini adalah salah:

```
a == 2
```

pasangan sesuatu hal yang berhubungan merupakan sebuah operator yang digunakan untuk menguji dua hubungan. Pada operator >= digunakan untuk menentukan jika operand disebelah kiri lebih besar dari atau sama dengan operand disebelah kanan. Dengan asumsi bahwa jika a adalah 4, dan b adalah 6, serta c adalah 4, maka ekspresi yang paling benar adalah sebagai berikut:

```
b> = a
a> = c
```

Namun ekpresi berikut dibawah ini adalah salah:

```
a> = 5
```

operator <= digunakan untuk menentukan jika operand disebelah kiri kurang dari atau sama dengan operand disebelah kanan. Perlu ditekankan sekali lagi, bahwa dengan asumsi a adalah 4, b adalah 6, dan c adalah 4, maka ekspresi yang benar adalah sebagai berikut:

```
a <= c
b <= 10
```

Namun ekpresi berikut adalah salah:

```
b <= a
```

Operator penghubung terakhir adalah !=, yang artinya adalah operator tidak

sama. Operator ini untuk menentukan jika operand disebelah kiri tidak sama dengan operand disebelah kanan, sedangkan kebalikan dari operator tersebut adalah operator ==. Seperti contoh sebelumnya, dengan asumsi a adalah 4, b adalah 6, dan c adalah 4, maka ekspresi yang benar adalah sebagai berikut:

```
a != b
b != c
```

Kalimat tersebut adalah benar karena a tidak sama dengan b dan b tidak sama dengan c. Namun berikut ini adalah ekspresi salah karena sama dengan c:

```
a != c
```

Tabel dibawah ini menunjukkan operator dimana nilai-nilai benar atau salah.

Table 5.2. Operator nilai benar atau salah.

EKPRESI	NILAI
x < y	salah, karena x lebih kecil daripada y
x > y	Betul, karena x lebih besar daripada y
x >= y	Betul, karena x lebih besar atau sama dengan y
x <= y	Salah, karena x lebih kecil atau sama dengan y
y != x	Betul, karena y tidak sama dengan x

Setelah diamati, sebenarnya terdapat sebuah pertanyaan yang sering kita tidak tahu yaitu: "apa yang dimaksud dengan benar?". Pertanyaan tersebut merupakan suatu hal yang sangat perlu untuk dipertimbangkan. Jika ekspresi relatif dapat dilakukan secara benar atau salah, bagaimana sesuatu yang mewakili nilai-nilai internal dalam sebuah program? Pada sebuah komputer misalnya, bagaimana komputer menyimpan benar?. Bagaimana pula cara menyimpan data yang salah?. Hal ini membutuhkan suatu tanda atau kode yang disepakati sesuai dengan kesepakatan umum pada sebuah komputer.

Dalam bahasa C++, nilai benar akan diwakili dengan angka 1 dan salah dengan angka 0. Untuk menggambarkan lebih lengkap, perhatikan program dibawah ini:

Program 5.1.

```
#include<conio.h>
#include <iostream>

using namespace std;

int main()
  {
  bool nilaiBetul, nilaiSalah;
```

```
   int x = 5, y = 10;

   nilaiBetul = x < y;
   nilaiSalah = y == x;

   cout << "Benar adalah " << nilaiBetul << endl;
   cout << "Salah adalah " << nilaiSalah << endl;
   getch();
   return 0;
}
```

Keluaran Programnya adalah sebagai berikut:

```
Benar adalah 1
Salah adalah 0
```

Selanjutnya coba kita memeriksa pernyataan-pernyataan yang berisi ekspresi relasi sedikit lebih dekat:

```
nilaiBetul= x < y;
nilaiSalah = y == x;
```

pernyataan-pernyataan tersebut tampaknya aneh karena menempatkan nilai perbandingan ke dalam sebuah variabel. Pada pernyataan pertama, variabel **trueValue** dengn diberi hasil x <y. Ketika x kurang dari y, ungkapan itu benar, dan variabel trueValue memberikan nilai 1. Sedangkan pada pernyataan yang kedua ungkapan y == x adalah salah, sehingga variabel falseValue akan keluar nilai 0. Perhatikan bahwa dalam kedua kasus operasi relasional dilaksanakan sebelum tugas operasi dilakukan.

Selain itu tanda kurung dapat digunakan untuk mengubah urutan operasi, hal ini selalu terjadi karena adanya operator relasional tingkatannya relatif lebih tinggi dan harus didahulukan dalam operasinya daripada operator yang tingkatnya lebih rendah. Demikian juga, ada operator aritmatika yang harus lebih diutamakan daripada operator relasional. Seperti dalam pernyataan berikut ini:

```
NilaiSalah = x < y – 8;
```

Dari peryataan diatas misalnya, yang pertama Y - 8 akan dievaluasi terlebih dahulu untuk menghasilkan nilai 2. Kemudian X yang bernilai 5 akan dibandingkan dengan 2. Jika nilai 5 kurang dari 2, nilai nol akan masukan ke falseValue. Tabel dibawah menunjukkan contoh lainnya termasuk pernyataan yang relatif biasa.

Tabel 5.3. Pernyataan ekpresi relasional

PERNYATAAN	KELUARAN
z = x < y	z sama dengan 0 karena x tidak kurang daripada y
cout << (x > y);	Tampilkan 1 karena x lebih besar daripada y.
z = x >= y;	z samadengan 1 karena x lebih besar samadengan y.
cout << (x <= y);	Tampilkan 0 karena x tidak kurang samadengan y
z = y != x;	z samadengan 1 karena y tidak samadengan x
cout << (x == y + 3);	Tampilkan 1 karena x sama dengan y + 3

Operator relasional juga memiliki urutan yang mempunyai prioritas diantara mereka sendiri. Kedua operator dalam tes kesamaan atau tidak sama (== dan ! =) mempunyai tingkat kesamaan yang sama antara satu sama lain.

Empat penghubung operator lainnya, ada yang mempunyai prioritas tinggi atau rendah antara satu sama lain. Keempat operator relasional seperti dalam. Tabel dibawah menunjukkan diutamakan yang berhubungan operator.

Tabel 5.4. Prioritas operator relasional (dari tertinggi ke rendah)

== !=
> >= < <=

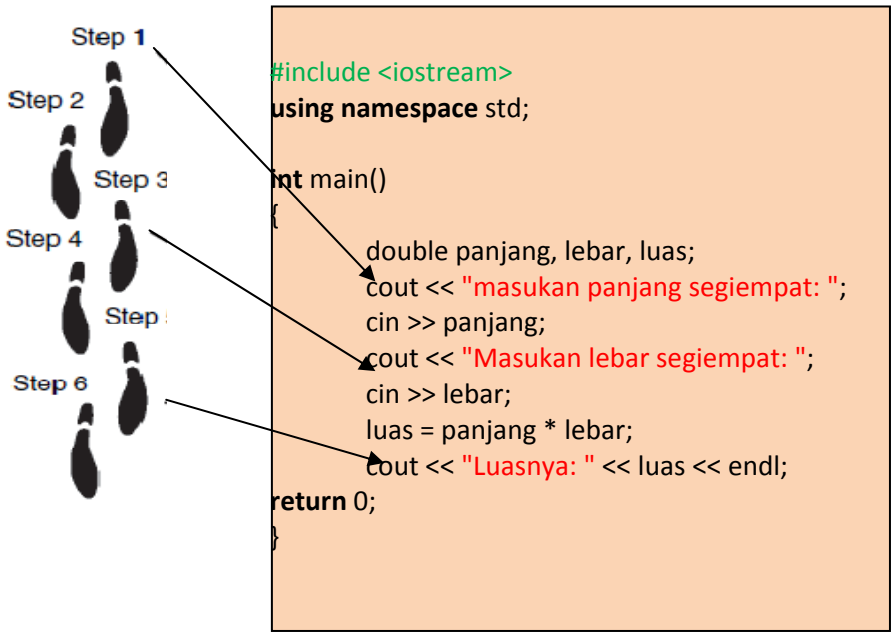
Berikut ini adalah sebuah contoh bagaimana operator diterapkan. Jika a = 9, b = 24, dan c = 0, pernyataan berikut ini akan menyebabkan a bernilai 1 akan ditampilkan.

```
cout <<(c == a> b);
```

Karena adanya nilai yang relatif diutamakan dari operator dalam berekspresi ini, a > b akan dievaluasi terlebih dahulu. Ketika 9 tidak lebih besar dari 24, maka hal tersebut akan mengevaluasi salah atau 0. Kemudian ketika C == 0 maka akan dihasilkan. Ketika c tidak sama dengan 0, hal tersebut akan menghasilkan nilai betul, atau 1. Sehingga nilai 1 akan dimasukkan ke dalam output stream dan ditampilkan.

5.3. Statement IF

Anda mungkin berpikir mengenai pernyataan dalam program procedural seperti langkah-langkah yang diambil sebagai individu. Untuk mencapai tujuan tersebut , maka harus dimulai dari awal dan mengikuti setiap langkah, satu setelah yang lain hingga dapat mencapai tujuan. Program-program yang ditulis ini adalah seperti halnya sebuah "jalan" dimana pelaksanaannya harus diikuti. Perhatikan langkah program dibawah ini:



Gambar 5.1. Langkah-langkah program

Seperti ditunjukkan dalam Gambar diatas, pelaksanaan program mengalir secara berurutan dari satu pernyataan ke pernyataan berikutnya. Jenis program ini sering disebut-garis lurus karena program yang dijalankan dalam pernyataan yang lurus "baris," tanpa simpangan kearah yang lain. Apakah hal tersebut tidak akan berguna, jika sebuah program dapat memiliki lebih dari satu "jalur atau arah" pelaksanaan? Bagaimana jika sebuah program dapat menjalankan beberapa pernyataan hanya dalam kondisi tertentu?.

Hal yang dapat dicapai dengan pernyataan IF, seperti digambarkan oleh Program dibawah ini. Pengguna memasukan tiga nilai ujian dan program menghitung rata-rata nilai tersebut. Jika rata-rata lebih besar dari 95, program memberikan selamat pada pengguna mendapatkan nilai tinggi tersebut. Untuk lebih jelasnya perhatikan program untuk mencari rata-rata pada tiga nilai masukan dibawah:

Program 5.2

```
#include <iostream>
#include <iomanip>
```

```
using namespace std;

int main()
      {
      int score1, score2, score3;
      double average;

      cout << "Masukan 3 nilai Ujian dan saya akan mencari rata-ratanya: ";
      cin >> score1 >> score2 >> score3;
      average = (score1 + score2 + score3) / 3.0;

      cout << fixed << showpoint << setprecision(1);
      cout << "Rata-rata Nilai Anda  Adalah" << average << endl;
      if (average == 100)
      {
          cout << "Selamat….! ";
          cout << "Nilai Anda Sempurna!\n";
      }
      return 0;
}
```

Keluaran program diatas adalah sebagai berikut:

```
Masukan 3 nilai Ujian dan saya akan mencari rata-ratanya: 80 90 70[Enter]
Rata-rata Nilai Anda  Adalah 80.0
```

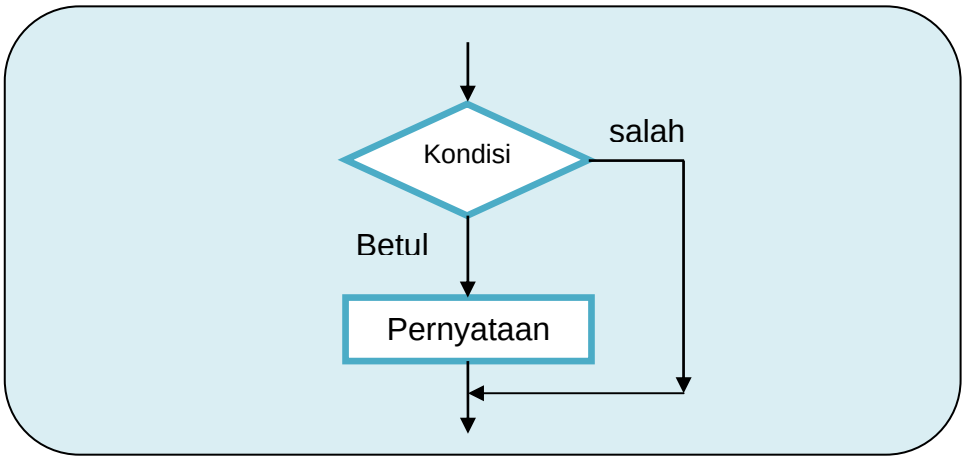
Keluaran program diatas adalah sebagai berikut:

```
Masukan 3 nilai Ujian dan saya akan mencari rata-ratanya: 100 100 100[Enter]
Rata-rata Nilai Anda  Adalah 100.0
Selamat….! Nilai Anda Sempurna!
```

Jika dalam program tersebut pada potongan program seperti dibawah ini:

```
if (average == 100)
{
      cout << "Selamat….! ";
      cout << "Nilai Anda Sempurna!\n";
}
```

Maka potongan program tersebut diatas akan menampilkan data tulisan "selamat….! "….., jika data yang dimasukan sama dengan seratus. Jika tidak maka hanya akan menampilkan rata-rata nilainya saja tanpa ada ucapan selamat atau " Selamat….! ". Gambar dibawah ini menunjukkan bentuk pernyataan yang digunakan pada pernyataan IF dan flowchart visual dapat dijelaskan cara kerjanya sebagai berikut:



Gambar 5.2. Diagram alir pernyataan IF

Dari gambar tersebut diatas pernyataan IF dapat ditulis sebagai berikut:

```
if (ekspresi)
{
      Pernyataan 1;
      Pernyataan 2;
      .
      .
      Pernyataan n;
}
```

Pernyataan IF adalah sangat sederhana cara kerjanya. Jika ekspresi didalam tanda kurung yang benar, maka pernyataan didalam braces (kotak) yang akan dijalankan, jika tidak, maka mereka diabaikan. Blok pernyataan ini adalah merupakan *conditionally executed* karena pernyataan hanya dijalankan didalam kondisi yang diekspresikan dalam tanda kurung yang benar. Program diatas menggambarkan sebuah pernyataan IF. Dalam pernyataan cout hanya dilaksanakan dibawah kondisi yang sama dengan rata-rata 100.

Misalnya kalau diuraikan dalam sebuah pernyataan dalm keseharian adalah sebagai berikut: Jika bensin mobil sudah hampir habis, maka berhenti dipompa bensin untuk mengisi bensin.

- Jika diluar hujan, pergi kedalam rumah.
- Jika Anda lapar, carilah sesuatu untuk dimakan.

Jika blok pernyataan *conditionally executed* hanya satu

pernyataan, maka braces dapat diabaikan. Misalnya, dalam Program diatas, jika dua pernyataan cout yang digabungkan menjadi satu pernyataan, mereka dapat menulis seperti yang ditunjukkan di sini.

```
if (average == 100)
    cout << "selamat ! Nilai anda Sempurna!\n";
```

Perintah IF yang menyatakan pernyataan kondisional (bersyarat) dapat ditulis sintaks sederhana IF adalah

```
if (kondisi) statement;
```

Statement pada sintaks di atas akan dilakukan jika kondisinya bernilai TRUE (tidak sama dengan nol). Apabila statement yang akan dilakukan lebih dari satu, maka sintaksnya menjadi seperti dibawah ini:

```
if (kondisi)
{
      statement1;
      statement2;
      ..
}
```

Contoh sederhana penggunaan IF adalah untuk menentukan boleh tidaknya seseorang melihat film bioskop. Seseorang diperbolehkan menonton jika usianya 17 tahun ke atas. Perhatikan contoh program C++ berikut ini:

Program 5.3

```
#include <iostream.h>
#include <conio.h>

using namespace std;

int main()
{
      int usia;
      cout << "Berapa usia Anda : ";
      cin >> usia;
      if (usia < 17)
          cout << "Anda tidak boleh menonton bioskop";
      getch();
      return 0;
}
```

Keluaran program diatas adalah:

```
Berapa usia Anda : 9
Anda tidak boleh menonton bioskop
```

Statement IF juga dapat ditambahkan ELSE sebagai konsekuensi alternatif jika kondisi tidak dipenuhi (FALSE). Untuk lebih jelasnya perhatikan sintaks program dibawah ini:

```
if (kondisi)
{
      statement1;
      statement2;
      ..}
else {
      statement1;
      statement2;
}
```

Anda dapat pula memodifikasi program C++ untuk menentukan boleh tidaknya seseorang menonton bioskop seperti program dibawah ini:

Program 5.4

```
#include <iostream.h>
#include <conio.h>

using namespace std;

int main()
{
      int usia;
      cout << "Berapa usia Anda : ";
      cin >> usia;
      if (usia < 17)
      cout << "Anda tidak boleh menonton bioskop";
      else
      cout << "Anda boleh menonton bioskop";
      getch();
      return 0;
}
```

Keluaran programnya adalah sebagai berikut:

```
Berapa usia Anda : 16
Anda tidak boleh menonton bioskop
```

Untuk menyatakan kondisi atau syarat, akan dicek pada IF, Anda dapat menggunakan operator logika dan operator relasional seperti yang telah dijelaskan sebelumnya. Perhatikan contoh penulisan program dibawah ini:

```
if ((a >= 2) && (b == 3))
    {
    ....
    }
```

Jangan dituliskan seperti ini:

```
if (a >= 2) && (b == 3)
    {
    ....
    }
```

Dan juga tidak boleh seperti ini:

```
if ((a >= 2) && (b = 3))
    {
    ....
    }
```

Perintah b = 3 merupakan assignment bukan relasional. C++ selalu memperlakukan nilai tidak sama dengan nol sebagai TRUE atau benar dan nilai nol sama dengan FALSE atau nilai salah. Oleh karena itu, dua perintah dibawah ini adalah identik. Perhatikan potongan program dibawah ini:

```
if (bil % 2 != 0)
    cout << "Bilangan ganjil";
if (bil % 2)
    cout << "Bilangan ganjil"
```

Selain itu, IF juga dapat berbentuk seperti program dibawah ini:

```
if (kondisi1)
    statement1;
else if (kondisi2)
    statement2;
else if (kondisi3)
    statement3;
    .
    .
else statement;
```

IF dapat juga menggunakan operator pembanding (*comparison operators*) untuk mengeksekusi suatu pilihan dari 2 pilihan yang ada, tergantung hasil pembandingannya. Penulisan Instruksi atau sintaknya adalah sebagai berikut:

```
if (Kondisi)
{
```

```
        Perintah yang akan dieksekusi jika kondisi bernilai true
}
else
{
        Perintah yang akan dieksekusi jika kondisi bernilai false
}
```

Kondisi menunjukkan berbagai *tipe statement* atau fungsi yang menghasilkan nilai benar atau salah. Berikut ini diberikan contoh kondisi yang berupa perbandingan antara variabel dengan nilai, variabel lain atau fungsi. Perhatikan potongan program dibawah ini:

```
if (x > 5)
{
        y  = 10;
}

if (x > y)
{
        y = x;
}

if (x > val(angka))
{
        y = 20;
}
```

Suatu ketika akan ditemui pula penggunaan instruksi *IF .. THEN* statements, dimana isntruksi ini digunakan untuk mengevaluasi lebih dari satu kondisi. Untuk itu digunakan tambahan *blok IF..THEN..ELSE IF* statements. Perhatikan contoh program dibawah ini:

```
if (x < 5)
        cout<<“Nilai x kurang dari 5”;
else
if (x < 10)
        cout<<“Nilai X antara 5 dan 9”;
else
        cout<<“Nilai x lebih dari 9”;
```

Jika suatu kondisi tergantung pada kondisi lain yang sudah bernilai true (seperti if hari = senin dan if jam = 6.30) maka harus digunakan percabangan bersarang (nested If statements). Perhatikan contoh berikut ini:

```
if (hari = senin)
{
    if (jam = 6.30)
    {
        ...
    }
}
```

Untuk lebih jelasnya mengenai pernyataan IF, perhatikan sebuah kasus yang digunakan untuk menentukan besarnya potongan dari pembelian barang yang diberikan seorang pembeli, dimana kasus tersebut mempunyai kriteria sebagai berikut:

- Tidak ada potongan jika total pembelian kurang Rp. 50.000,-
- Jika total pembelian lebih dari atau sama dengan Rp. 50.000,- potongan yang diterima sebesar 20% dari total pembelian.

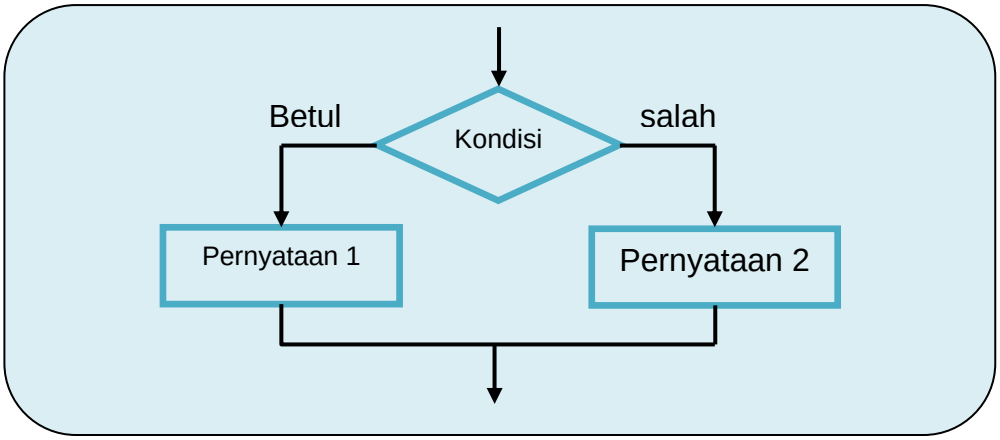
Perhatikan Program yang menggunakan pernyataan IF.

Program 5.5

```
#include<stdio.h>
#include<conio.h>
#include<iostream>

using namespace std;

main()
{
        double tot_beli, potongan=0, jum_bayar=0;
        cout<<"Total Pembelian Rp. ";
        cin>>tot_beli;
        if (tot_beli >= 50000)
                potongan = 0.2 * tot_beli;
                cout<<"Besarnya Potongan Rp. ";
    jum_bayar = tot_beli - potongan;
                cout<<"Jumlah yang harus dibayarkan Rp. ",jum_bayar;
        getch();
  return 0;
}
```

5.4. Pernyataan IF/ELSE

Pernyataan IF/ELSE merupakan sebuah pengembangan pernyataan IF. Secara umum bentuk pernyataan ini juka digambarkan dalam bentuk flowchart sebagai berikut:



Gambar 5.3. Diagram alir IF-ELSE

Jika diuraikan dalam bentuk kalimat, gambar diatas dapat ditulis menjadi sebuah pernyataan seperti dibawah ini:

```
if (kondisi)
{
    Pernyataan 1
}
else
{
    Pernyataan 2
}
```

Seperti pada pernyataan IF diatas, sebuah kondisi akan dilihat terlebih dahulu. Jika kondisi adalah benar, maka sebuah blok yang berisi satu atau lebih pernyataan akan dijalankan. Jika ekspresi adalah salah, maka pernyataan grup berbeda yang akan dijalankan. Program dibawah merupakan contoh penggunaan pernyataan bersama operator modulus untuk menentukan apakah angka yang dihasilkan ganjil atau genap.

Program 5.6

```
#include <conio.h>
#include <iostream>

using namespace std;

int main()
```

```
{
   int number;
   cout << "masukan bilangan bulat dan saya akan memilahnya dengan IF\n";
   cout << "adalah ganjil atau genap. ";
   cin >> number;
   if (number % 2 == 0)
            cout << number << " adalah genap.\n";
   else
      cout << number << " adalah ganjil.\n";
   getch();
   return 0;
}
```

Keluaran program adalah sebagai berikut:

```
masukan bilangan bulat dan saya akan memilahnya dengan IF
adalah ganjil atau genap. 17[Enter]
17 adalah ganjil
```

Bagian lain diakhir pernyataan IF akan menentukan pernyataan yang akan dijalankan ketika kondisi yang ada tidak sesuai atau salah. Bila angka 2% tidak sama 0, pesan yang dicetak menunjukkan angka ganjil. Perlu diketahui bahwa program ini hanya akan mengambil salah satu dari dua jalur pernyataan IF/ELSE.. Jika Anda berpikir tentang pernyataan dalam program komputer, langkah-langkah yang diambil berdasarkan pertimbangkan pernyataan IF/ELSE sebagai penentu jalan. Bahkan kadang terjadi jalanya program memutar, seperti halnya pada sebuah pernyataan IF, sedangkan pada pernyataan IF/ELSE menyebabkan pelaksanaan program mengikuti salah satu dari dua jalur yang tersedia.

Perhatikan gaya pemrograman yang digunakan untuk membangun pernyataan IF/ELSE. Atau dengan kata lain adalah pada tingkat yang sama seperti halnya perbedaan. Pernyataan eksekusi yang dikendalikan oleh orang lain kadang berbeda satu tingkat.

Seperti halnya pada pernyataan IF, jika tidak menggunakan kotak sebagai bagian kontrol sebuah pernyataan. Program diatas menggambarkan hal tersebut ini. Ia juga memperlihatkan salah satu cara untuk menangani sebuah masalah pemrograman yaitu pembagian oleh nol.

Pembagian dengan nol sangat mustahil untuk dilakukan secara matematis dan biasanya menyebabkan program crash. Hal Ini berarti program akan terhenti sebelum waktunya, kadang-kadang akan muncul dengan pesan kesalahan. Program diatas menunjukkan salah satu cara untuk menguji nilai dari pembagi sebelum dilakukan pembagian. Nilai num2 akan diuji sebelum pembagian dilakukan. Jika pengguna memasukkan angka 0, maka baris dikontrol oleh bagian IF ketika program dijalankan, menampilkan

pesan yang menunjukkan program tidak dapat melakukan pembagian nol. Jika tidak, bagian lain mengambil kontrol, dan kemudian membagi num1 oleh num2 serta menampilkan hasilnya.

Salah satu aplikasi penggunaan IF/ELSE misalnya digunakan pada suatu permasalahan untuk menentukan besarnya potongan dari pembelian barang yang diberikan seorang pembeli, dengan kriteria sebagai berikut:

- jika total pembelian kurang dari Rp. 50.000,- potongan yang diterima sebesar 5% dari total pembelian.
- Jika total pembelian lebih dari atau sama dengan Rp. 50.000,- potongan yang diterima sebesar 20% dari total pembelian.

Untuk lebih jelasnya perhatikan contoh program IF/ELSE dibawah ini:

Program 5.7

```
#include<stdio.h>
#include<conio.h>
#include<iostream>

using namespace std;

main()
{
      double tot_beli, potongan=0, jum_bayar=0;

      cout<<"Total Pembelian Rp. ";
      cin>>tot_beli;

      if (tot_beli >= 50000)
      potongan = 0.2 * tot_beli;
      else
      potongan = 0.05 * tot_beli;
      cout<<"Besarnya Potongan Rp. "<<potongan<<endl;
      jum_bayar = tot_beli - potongan;
      cout<<"Jumlah yang harus dibayarkan Rp. ",jum_bayar;
      getch();
       return 0;
}
```

Program dibawah ini digunakan untuk menentukan sebuah operasi pembagian yang menggunakan IF/ELSE , jika suatu bilangan dibagi dengan nol maka program akan memberitahukan bahwa program tidak bisa melakukan operasi.

Program 5.8

```
#include <conio.h>
#include <iostream>

using namespace std;

int main()
{
        double num1, num2, quotient;
        cout << "Masukan Angka: ";
        cin >> num1;
        cout << "Masukan Angka Lain: ";
        cin >> num2;

        if (num2 == 0)
        {
        cout << "Pembagian oleh 0 tidak mungkin dilakukan.\n";
        cout << "silakan masukan angka lagi, ";
        cout << "Angka lain yang lebih besar daripada 0.\n";
        }
        else
        {
        quotient = num1 / num2;
        cout << "The quotient of " << num1 << " divided by ";
        cout << num2 << " is " << quotient << ".\n";
        }
        getch();
        return 0;
}
```

Keluaran program setelh diberi masukan adalah sebagai berikut:

```
Masukan Angka: 10[Enter]
Masukan Angka Lain: 0[Enter]
Pembagian oleh 0 tidak mungkin dilakukan.
silakan masukan angka lagi, Angka lain yang lebih besar daripada 0
```

5.5. Pernyataan IF/ELSE IF

Pernyataan IF/ELSE IF merupakan sebuah pernyataan dimana programmer dapat membuat beberapa keputusan dengan menggunakan aturan yang berbeda tetapi ada kaitannya dengan pernyataan sebelumnya. Misalnya, programmer akan menentukan jenis mantel atau jaket yang dipakai dalam

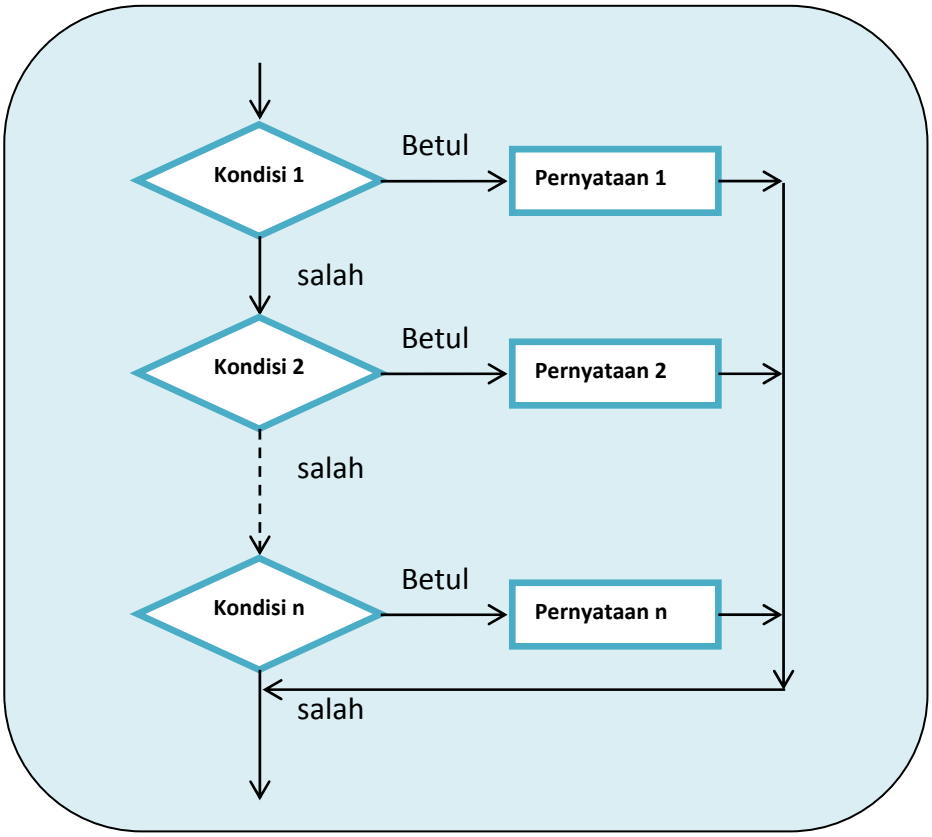
konsultasi sehingga aturanya adalah sebagai berikut:

if sangat dingin, pakailah mantel yang tebal,
else, jika dingin, memakai jaket yang tipis,
else, jika berangin, memakai baju hem,
else, jika panas, tidak perlu memakai jaket.

Tujuan aturan tersebut adalah untuk menentukan jenis pakaian luar yang akan dipakai. Jika sangat dingin, pertama menharuskan menggunakan aturan yang memakai mantel tebal. Semua peraturan lainnya kemudian dapat diabaikan. Jika aturan pertama tidak berlaku, dan jika tidak dingin sekali, maka aturan kedua digunakan. Jika aturan tidak berlaku, aturan yang ketiga digunakan, dan seterusnya.

Suatu cara yang digunakan untuk menhubungkan peraturan tersebut sangat penting. Jika mereka hanya berdiri sendiri, dimungkinkan akan keluar rumah dengan memakai jaket yang salah, atau mungkin, lebih dari satu jaket. Misalnya, jika berangin, maka aturannya harus berpakaian baju hem. Bagaimana jika ada keduanya yaitu sangat dingin dan berangin? Apakah akan memakai sebuah baju hem?. Karena aturan harus dipatuhi, pertama akan menentukan aturan memakai mantel tebal dan memakai baju hem karena berangin.

Jenis pengambilan keputusan juga sangat umum dalam pemrograman. Dalam bahasa C++ hal tersebut dicapai melalui pernyataan IF/ELSE IF. Gambar dibawah menunjukan format cara kerjanya.



Gambar 5.4. Pernyataan IF/ELSE IF

Dari gambar diatas dapat diuraikan dalam pernyataan umu IF/ELSE IF adalah sebagai berikut:

```
if (kondisi 1)
{
      Pernyataan 1;
}
else if (kondisi 2)
{
      Pernyataan 2;
}
            .
            .
else if (kondisi n)
{
      Pernyataan n;
}
```

Susunan ternyataan tersebut seperti halnya sebuah pernyataan berbentuk rantai IF/ELSE. Pada bagian ELSE sebuah pernyataan akan terkait dengan IF dari bagian yang lain. Ketika digabungkan cara ini, maka IF/ELSE dari rantai akan menjadi sebuah pernyataan. Program dibawah menunjukkan contoh, dimana Pengguna diminta untuk memasukkan angka skor tes dan program menampilkan huruf yang besar.

Program dibawah ini menggunakan pernyataan IF/ELSE yang digunakan untuk operasi file huruf kapital.

Program 5.9

```
#include <conio.h>
#include <iostream>

using namespace std;

int main()
{
        int testScore;
        char grade;

        cout << "Masukan Nilai Angka maka saya akan mengujinya\n";
        cout << "Nilai Konversi huruf adalah: ";
        cin >> testScore;

        if (testScore < 60)
        grade = 'F';
        else if (testScore < 70)
        grade = 'D';
        else if (testScore < 80)
        grade = 'C';
        else if (testScore < 90)
        grade = 'B';
        else if (testScore <= 100)
        grade = 'A';
        cout << " Grade Anda Adalah " << grade << ".\n";
        getch();
        return 0;
}
```

Keluaran program setelah memasukan data adalah sebagai berikut:

```
Masukan Nilai Angka maka saya akan mengujinya
Nilai Konversi huruf adalah: 90
```

```
Grade Anda Adalah A.
```

Pernyataan IF/ELSE mempunyai sejumlag karakterisitik khusus. Perhatikan dan analisis kerja dari ekprsesi relasional pengujian sebuah testscore < 60 dibawah ini:

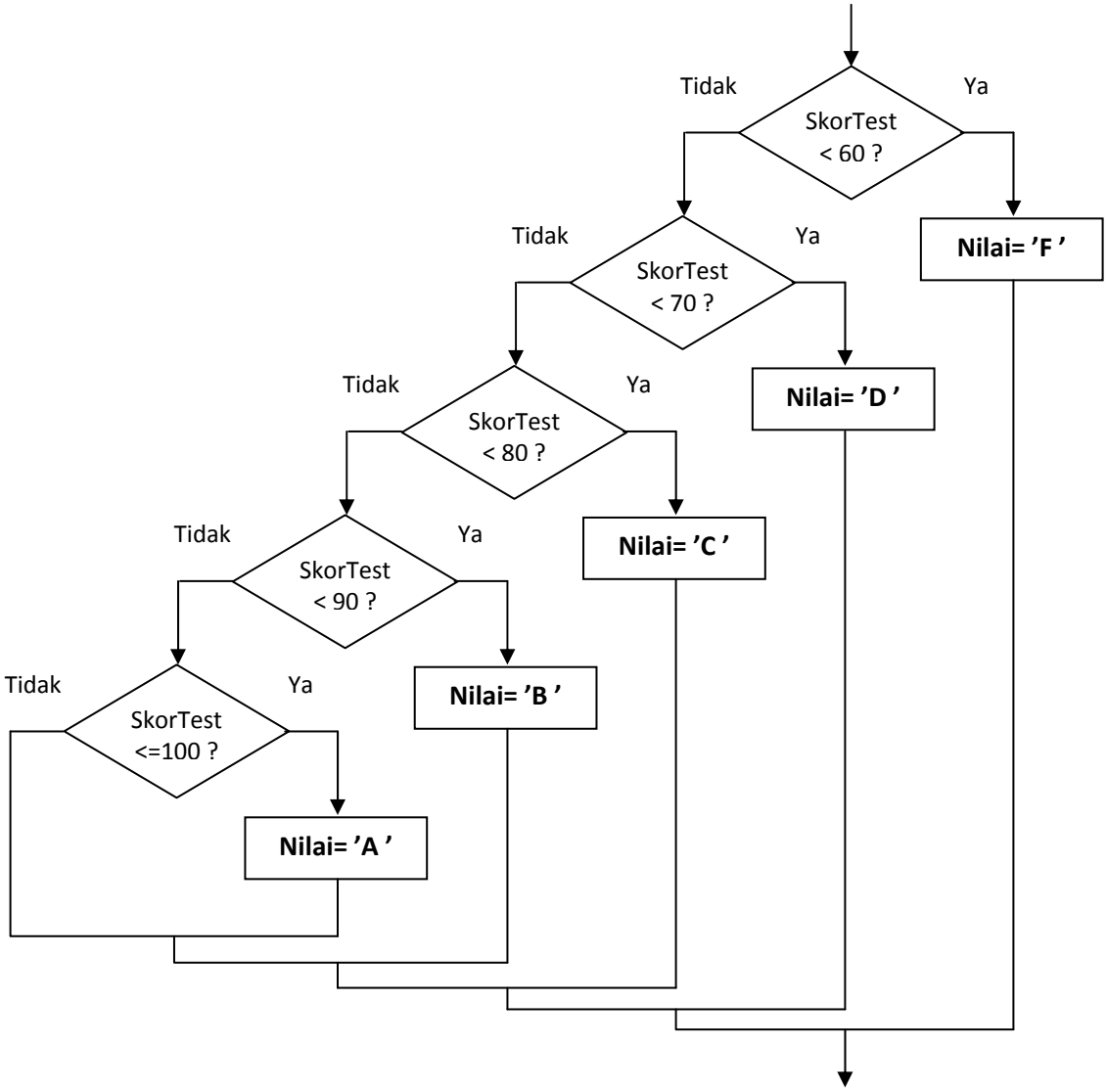
```
if (testScore < 60)
nilai = 'F';
```

Jika testScore kurang dari 60, huruf 'F' dimasukan ke nilai dan sisanya dari pernyataan yang terkait dengan IF diabaikan. Jika TestScore tidak kurang dari 60, maka tugsa akan diambil alih oleh bagian lain pernyataan IF berikutnya yang akan dijalankan.

```
else if (testScore < 70)
nilai = 'D';
```

Pertama jika seluruh pernyataan disaring untuk memilih nilai kurang dari 60, sehingga saat ini berikutnya pernyataan IF memilih, variabel testScore akan memiliki nilai 60 atau lebih besar. Jika "testScore" kurang dari 70, huruf 'D' dimasukan ke nilai dan sisanya dari Pernyataan IF/ELSE IF ini diabaikan. lingkaran peristiwa ini terus berlangsung sampai salah satu ekspresi kondisional ditemukan benar atau akhir pernyataan yang dihadapi. Dalam kedua kasus, program ini dilanjutkan kembali pada pernyataan yang mengikutinya.

Pernyataan IF/ELSE IF, merupakan pernyataan untuk mencetak kondisi yang berhasil. Gambar dibawah menunjukkan jalan yang dapat diambil melalui pernyataan IF/ELSE IF.



Gambar 5.5. Pernyataan IF/ELSE IF

pernyataan IF pada setiap struktur bergantung pada semua pernyataan IF sebelumnya yang salah.

Pernyataan ELSE IF berikutnya dijalankan apabila ekspresi kondisional mengikuti ELSE IF itu benar dan semua ekspresi kondisional sebelumnya adalah salah. Untuk mendemonstrasikan bagaimana bekerjanya, perhatikan program dibawah, yang menggunakan pernyataan IF yang independen bukan sebuah pernyataan IF/ELSE. Program

tersebut digunakan untuk mengilustrasikan kesalahan yang terjadi ketika pernyataan IF/ELSE yang digunakan untuk menentukan nilai huruf ke nilai angka. Perhatikan program dibawah ini:

Program 5.10

```
#include <conio.h>
#include <iostream>

using namespace std;

int main()
{
        int skortest;
        char nilai;

        cout << "Masukan Nilai Angka maka saya akan mengujinya\n";
        cout << "Nilai Konversi huruf adalah: ";
        cin >> skortest;

        if (skortest < 60)
                nilai = 'F';
        if (skortest < 70)
                nilai = 'D';
        if (skortest < 80)
                nilai = 'C';
        if (skortest < 90)
                nilai = 'B';
        if (skortest <= 100)
                nilai = 'A';
        cout << " Nilai Anda Adalah " << nilai << ".\n";
        getch();
        return 0;
}
```

Keluaran program setelah dilakukan pengisian data pada masukan adalah:

```
Masukan Nilai Angka maka saya akan mengujinya
Nilai Konversi huruf adalah: 40[Enter]
Nilai Anda Adalah A.
```

Pada program dibawah, semua pernyataan IF dijalankan karena hal tersebut merupakan pernyataan yang berdiri sendiri. Pada contoh keluaran, testScore diberikan nilai 40, namun ada siswa yang menerima A. Berikutnya apa yang terjadi ?. Karena skor siswa kurang dari 60,

yang pertama pernyataan IF akan menyebabkan nilai 'F' memasukannya ke variabel grade. Namun, karena pernyataan berikutnya jika tidak tersambung ke pernyataan pertama, maka hal tersebut akan dieksekusi dengan baik. ketika testScore kurang dari 70, dan menyebabkan nilai 'D' ditugaskan ke grade, dan mengganti 'F' yang sebelumnya disimpan di sana. Ini terus sampai semua jika ada pernyataan dijalankan. Yang terakhir akan menyebabkan 'A' akan ditugaskan untuk "grade".

Pada program dibawah menggunakan pernyataan if/else if untuk melaksanakan konversi nilai hurif (A, B, C, D, or F) menjadi nilai angka. Perhatikan program dibawah ini:

Program 5.11

```
#include <conio.h>
#include <iostream>

using namespace std;

int main()
{
        int testScore;
        char grade;
        bool goodScore = true;
        cout << "Masukan Nilai Angka maka saya akan mengujinya\n";
        cout << "Nilai Konversi huruf adalah: ";
        cin >> testScore;

        if (testScore < 60)
                grade = 'F';
        else if (testScore < 70)
                grade = 'D';
        else if (testScore < 80)
                grade = 'C';
        else if (testScore < 90)
                grade = 'B';
        else if (testScore <= 100)
                grade = 'A';
        else
                goodScore = false;

        if (goodScore)
        cout << " Grade Anda Adalah " << grade << ".\n";
        else
        {
```

```
                cout << testScore << " Adalah skor anda yang tidak sah.\n";
                cout << "Silakan masukan skor dibawah 100. \n";
        }
        getch();
        return 0;
}
```

Keluaran program ddiatas setelah memasukan data adalah sebagai berikut:

```
Masukan Nilai Angka maka saya akan mengujinya
Nilai Konversi huruf adalah: 110
110 Adalah skor anda yang tidak sah.
Silakan masukan skor dibawah 100.
```

5.6. Pernyataan IF/ELSE Majemuk

Bentuk dari IF-ELSE bertingkat sebenarnya serupa dengan NESTED IF, keuntungan penggunaan IF-ELSE bertingkat dibanding dengan NESTED IF adalah cara atau bentuk penulisan yang lebih sederhana. Untuk lebih jelasnya perhatikan bentuk penulisan program dibawah ini:

```
if (syarat)
{
    … perintah;
    … perintah;
}
else if (syarat)
{
    … perintah;
    … perintah;
}
else
{
    … perintah;
    … perintah;
}
```

Untuk lebih jelasnya, perhatikan sebuah kasus di suatu perusahaan memberikan komisi kepada para selesman dengan ketentuan sebagai berikut:

- Bila salesman dapat menjual barang hingga Rp. 200.000 ,- , akan diberikan uang jasa sebesar Rp. 10.000 ditambah dengan uang komisi Rp. 10% dari pendapatan yang diperoleh hari itu.
- Bila salesman dapat menjual barang diatas Rp. 200.000 ,- , akan diberikan uang jasa sebesar

Rp. 20.000 ditambah dengan uang komisi Rp. 15% dari pendapatan yang diperoleh hari itu.

- Bila salesman dapat menjual barang diatas Rp. 500.000 ,- , akan diberikan uang jasa sebesar Rp. 30.000 ditambah dengan uang komisi Rp. 20% dari pendapatan yang diperoleh hari itu.

Kasus yang ada di perusahaan tersebut dapat diatasi dengan menggunakan program IF-Else-IF, dimana programnya adalah sebagai berikut:

Program 5.12

```
#include<stdio.h>
#include<conio.h>
#include<iostream.h>

using namespace std;

int main()
{
 float pendptan, jasa=0, komisi=0, total=0;

 cout<<"Pendapatan Hari ini Rp. ";
 cin>>pendptan;
 if (pendptan >= 0 && pendptan <= 200000)
 {
   jasa=10000;
   komisi=0.1*pendptan;
 }
 else if(pendptan<=500000)
 {
   jasa=20000;
   komisi=0.15*pendptan;
 }
 else
 {
   jasa=30000;
   komisi=0.2*pendptan;
 }

  /* menghitung total */
  total = komisi+jasa;

  cout<<"Uang Jasa    Rp. "<<jasa<<endl;
  cout<<"Uang Komisi  Rp. "<<komisi<<endl;
  cout<<"============================"<<endl;
```

```
  cout<<"Hasil Total  Rp. "<<total<<endl;

  getch();
  return 0;
}
```

Keluaran program diatas adalah sebagai berikut:

```
Pendapatan Hari ini Rp. 10000
Uang Jasa    Rp. 10000
Uang Komisi  Rp. 1000
============================
Hasil Total  Rp. 11000
```

5.7. Pernyataan NESTED IF

Ketika sebuah pernyataan IF muncul didalam pernyataan IF lain, maka hal ini dikategorikan sebagai nested. Kenyataannya dalam struktur IF/ELSE adalah pernyataan nested if. Jika masing-masing setelah if pertama adalah nested di bagian lain IF sebelumnya. Karena nested if merupakan pernyataan if yang berada didalam pernyataan if yang lainnya, sehingga utuk lebih jelasnya bentuk penulisan pernyataan Nested if dapat ditulis seperti berikut dibawah ini:

```
if(syarat)
if(syarat)
  … perintah;
else
  … perintah;
else
if(syarat)
  … perintah;
else
  … perintah;
```

Sebagai contoh saja misalnya program dibawah digunakan untuk menentukan apakah pelanggan bank memenuhi syarat khusus untuk mendapatkan bunga pinjaman, dimana syarat khusus ini ditujukan untuk pelanggan orang-orang yang baru lulus dari sekolah dan bekerja. Program berikut ini mendemontasikan pernyataan nested if.

Program 5.13

```
#include<conio.h>
```

```
#include <iostream>

using namespace std;

int main()
{
        char pekerja, barululus;
        cout << "Jawablah pertanyaan dibawah ini\n";
        cout << "dengan jawaban Y untuk Ya atau ";
        cout << "T untuk Tidak\n";
        cout << "Apakah Anda Bekerja? ";
        cin >> pekerja;
        cout << "telahkan Anda selesai sekolah ";
        cout << "dalam dua tahun ini? ";
        cin >> barululus;

        if (pekerja == 'Y')
        {
                if (barululus == 'Y')
                {
                cout << "Kwalitas anda khusus ";
                cout << "tertarik ?\n";
                }
        }
         getch();
        return 0;
}
```

Keluaran program diatas setelah diberi masukan adalah:

```
Jawablah pertanyaan dibawah ini
dengan jawaban Y untuk Ya atau T untuk Tidak
Apakah Anda Bekerja? Y
```

Karena pernyataan IF pertama kondisinya melaksanakan masalah kedua, baik variabel pekerja dan barululus harus diatur ke 'Y' supaya pesan mengenai pengguna dengan kualifikasi khusus dicetak sesuai dengan tingkat suku bunga. Jenis pernyataan nested if yang baik untuk pilihan yang kecil dengan mengkategorisasikan data. Satu-satunya cara untuk menjalankan program pernyataan IF yang kedua adalah dengan ekspresi kondisional yang pertama harus benar. Namun kadang ada fitur yang tidak dikehendaki atau sering dikenal sebagai kesalahan dalam sebuah program. Jika seorang pengguna

memasukkan 'N' (atau karakter selain 'Y') sebagai pekerja atau barululus, program tidak mencetak pesan yang memberitahukan bahwa mereka tidak memenuhi syarat.

Pernyataan lain yang dapat digunakan untuk mengatasi masalah ini digambarkan dalam program dibawah. Program dibawah ini merupakan sebuah pernyataan NESTED IF.

Program 5.14

```
#include<conio.h>
#include <iostream>

using namespace std;

int main()
{
        char pekerja, barululus;
        cout << "Jawablah pertanyaan dibawah ini\n";
        cout << "dengan jawaban Y untuk Ya atau ";
        cout << "T untuk Tidak\n";
        cout << "Apakah Anda Bekerja? ";
        cin >> pekerja;
        cout << "telahkan Anda selesai sekolah ";
        cout << "dalam dua tahun ini? ";
        cin >> barululus;

        if (pekerja == 'Y')
        { // Nested if
                if (barululus == 'Y')
                {
                        cout << "Kwalitas anda khusus ";
                        cout << "Anda tertarik\n";
                }
                else
                {
                        cout << "Anda harus Lulus dari ";
                        cout << "sekolah paling lama 2 tahun\n";
                        cout << "dan berkwalitas.\n";
                }
        }
        else
        {
                cout << "Anda harus menjadi pekerja yang berkwalitas.\n";
        }
  getch();
```

```
        return 0;
}
```

Keluaran program diatas setelah diberi masukan adalah:

```
Jawablah pertanyaan dibawah ini
dengan jawaban Y untuk Ya atau T untuk Tidak
Apakah Anda Bekerja? Y
telahkan Anda selesai sekolah dalam dua tahun ini? Y
Kwalitas anda khusus Anda tertarik
```

contoh lainnya misalnya pada suatu perusahaan yang memberikan komisi kepada para selesman dengan ketentuan sebagai berikut:

- Bila salesman dapat menjual barang hingga Rp. 20.000 ,- , akan diberikan uang jasa sebesar Rp. 10.000 ditambah dengan uang komisi Rp. 10% dari pendapatan yang diperoleh hari itu.
- Bila salesman dapat menjual barang diatas Rp. 20.000 ,- , akan diberikan uang jasa sebesar Rp. 20.000 ditambah dengan uang komisi Rp. 15% dari pendapatan yang diperoleh hari itu.
- Bila salesman dapat menjual barang diatas Rp. 50.000 ,- , akan diberikan uang jasa sebesar Rp. 30.000 ditambah dengan uang komisi Rp. 20% dari pendapatan yang diperoleh hari itu.

Perhatikan contoh program dengan menggunakan pernyataan Nested-If dibawah:

Program 5.15

```
#include<stdio.h>
#include<conio.h>
#include<iostream.h>

using namespace std;

int main()
{
 float pendptan, jasa=0, komisi=0, total=0;

 cout<<"Pendapatan Hari ini Rp. ";
 cin>>pendptan;

 if (pendptan >= 0 && pendptan <= 200000)
 {
   jasa=10000;
   komisi=0.1*pendptan;
 }
 else
```

```
 {
   if(pendptan<=500000)
   {
      jasa=20000;
      komisi=0.15*pendptan;
   }
   else
   {
      jasa=30000;
      komisi=0.2*pendptan;
   }
  }

  /* menghitung total */
  total = komisi+jasa;

  cout<<"Uang Jasa    Rp. "<<jasa<<endl;
  cout<<"Uang Komisi  Rp. "<<komisi<<endl;
  cout<<"============================"<<endl;
  cout<<"Hasil Total  Rp. "<<total<<endl;

  getch();
  return 0;
}
```

Keluaran program diatas adalah:

```
Pendapatan Hari ini Rp. 5600
Uang Jasa    Rp. 10000
Uang Komisi  Rp. 560
============================
Hasil Total  Rp. 10560
```

5.8. Operator Logika

Selain operator relasional, bahasa pemrograman umumnya mendukung tambahan operator yaitu AND, OR, dan NOT. Operator And, Or, dan Not dikenal dengan nama operator logika. Dengan menggunakan operator ini dua atau lebih test perbandingan dapat dilakukan. Pada bagian sebelumnya telah ditulis sebuah program tes dengan dua kondisi pada pernyataan IF. Dalam bagian ini ditulis program yang menggunakan operator logis untuk menggabungkan dua atau lebih menjadi satu kalimat relasional. Tabel berikut merupakan daftar operator logika pada bahasa C++.

Tabel 5.5. Operator Logika

OPERATOR	ARTI	PENJELASAN
&&	AND	Menghubungkan dua ekpresi menjadi satu. Kedua ekpresi tersebut harus benar supaya hasilnya benar
||	OR	Menghubungkan dua ekpresi menjadi satu. Hasil akan menjadi benar jika salah satu atau keduanya ada yang benar.
!	NOT	Operator yang melakukan kebalikan dari ekpresi. Jika ekpresi benar maka hasil salah atau sebaliknya.

5.8.1. Operator &&

Operator && yang dikenal sebagai operator logika. Dua kalimat diperlukan sebagai operands dan membuat kalimat benar, jika kedua sub-kalimatnya benar. Berikut dibawah ini adalah contoh program pernyataan IF yang menggunakan operator &&:

```
if (suhu < 20 && waktu > 12)
    cout << "suhu telah mencapai level yang berbahaya";
```

Perhatikan bahwa kedua kalimat yang diANDkan bersama adalah kalimat yang lengkap untuk mengevaluasi benar atau salah. Pertama suhu <20 dievaluasi untuk menghasilkan hasil benar atau salah. Kemudian menit> 12 dievaluasi untuk menghasilkan hasil benar atau salah. Kemudian kedua hasil tersebut diANDkan sehingga akan mendapatkan hasil akhir untuk seluruh ekspresi. Pernyataan cout hanya akan dijalankan jika suhu kurang dari 20 dan waktunya lebih besar daripada 12. Jika salah satu penghubung teresebut salah, seluruh ekspresi yang dihasilkan juga salah dan pernyataan cout tidak dijalankan.

Tabel dibawah meruapkan tabel kebenaran untuk operator &&. Kebenaran tabel berisi semua kemungkinan kombinasi nilai yang mungkin dimiliki oleh dua kalimat, Sebagaimana dalam tabel menunjukkan, kedua sub-ekspresi harus benar untuk operator && sehingga akan menghasilkan nilai yang benar pula.

Table 5.6. Logika AND

EKPRESI	NILAI KEBENARAN	LOGIKA
Salah && salah	Salah	0
Salah && benar	Salah	0
Benar && salah	Salah	0
Benar && benar	benar	1

Operator && dapat digunakan untuk mempermudah program-program lain yang akan menggunakan pernyataan NESTED IF. Program dibawah ini adalah versi Program sebelumnya yang kemudian ditulis ulang dengan operator logika. Perhatikan program dibawah ini:

Program 5.16

```
#include<conio.h>
#include <iostream>

using namespace std;

int main()
{
        char pekerja, barululus;
        cout << "Jawablah pertanyaan dibawah ini\n";
        cout << "dengan jawaban Y untuk Ya atau ";
        cout << "T untuk Tidak\n";
        cout << "Apakah Anda Bekerja? ";
        cin >> pekerja;
        cout << "telahkan Anda sekolah ";
        cout << "dalam dua tahun ini? ";
        cin >> barululus;

        if (pekerja == 'Y' && barululus == 'Y')          // menggunakan &&
        {                                                 // logical operator
                cout << "Kwalitas anda spesial ";
                cout << "dan anda tertarik.\n";
        }
        else
        {
                cout << "anda harus bekerja dan mempunyai\n";
                cout << "lulusan dari sekolah SMK dalam\n";
                cout << "waktu paling lama 3 tahun.\n";
        }
        getch();
```

```
        return 0;
}
```

Keluaran program diatas setelah diberi masukan adalah

```
Jawablah pertanyaan dibawah ini
dengan jawaban Y untuk Ya atau T untuk Tidak
Apakah Anda Bekerja? T
telahkan Anda sekolah dalam dua tahun ini? Y
anda harus bekerja dan mempunyai
lulusan dari sekolah SMK dalam
waktu paling lama 3 tahun.
```

5.8.2. Operator ||

Operator || yang dikenal sebagai operator logika OR. Dua kalimat yang diperlukan sebagai operand akan membuat kalimat benar bila salah satu dari sub-kalimat yang benar. Berikut dibawah ini adalah contoh sebuah pernyataan IF yang menggunakan operator ||

```
if (suhu < 20 || suhu > 100)
    cout << "suhu dalam level yang berbahaya.";
```

Pernyataan cout akan dijalankan jika suhu kurang dari 20 ATAU suhu lebih besar dari 100. Jika salah satunya benar, maka seluruh ekspresi akan menghasilkan keluaran benar dan pernyataan cout akan dijalankan. Setelah cout dijalankan maka akan menghasilkan keluaran "suhu dalam level yang berbahaya". Table dibawah ini merupakan table kebenaran operator OR

Table 5.7. Logika OR

EKPRESI	NILAI KEBENARAN	LOGIKA
Salah && salah	Salah	0
Salah && benar	Benar	1
Benar && salah	Benar	1
Benar && benar	benar	1

Semua ekspresi pada operator OR akan menjadi salah jika semua dari sub-kalimat salah, tetapi akan benar jika salah satu sub-kalimatnya benar. Tidak peduli apakah salah satu sub-ekspresinya salah atau benar. Program dibawah ini akan melakukan tes untuk mengetahui orang yang memenuhi syarat untuk mendapatkan pinjaman. Program ini akan menentukan apakah pelanggan mempunyai gaji setidaknya $35,000

per tahun atau telah bekerja selama lebih dari lima tahun. Perhatikan program dibawah ini:

Program 5.17

```
#include<conio.h>
#include <iostream>

using namespace std;

int main()
{
        double income;
        int years;

        cout << "Apakah ini masukan tahunan Anda? ";
        cin >> income;
        cout << "berapa banyak anda bekerja setiap tahunnya "
        << "Pekerjaan anda sekarang? ";
        cin >> years;

        if (income >= 35000 || years > 5) // Uses the || logical operator
                cout << "Kwalifikasi Anda.\n";
        else
        {
            cout << "Anda akan mendapatkan gaji paling tidak $35,000 atau lebih\n";
            cout << "setelah bekerja lebih dari lima tahun.\n";
        }
        getch();
        return 0;
}
```

Keluaran program diatas setelah diberi masukan adalah

```
Apakah ini masukan tahunan Anda? 30000
berapa banyak anda bekerja setiap tahunnya Pekerjaan anda sekarang? 5
Anda akan mendapatkan gaji paling tidak $35,000 atau lebih
setelah bekerja lebih dari lima tahun.
```

5.8.3. Operator !

Operator ! akan melakukan operasi logika NOT. Dibutuhkan sebuah operand atau sebeliknya seperti benar atau salah. Dengan kata lain, jika ungkapan itu benar, maka operator ! akan membalik menjadi salah, dan jika ekspresi salah, maka akan dibalik menjadi benar. Dibawah ini adalah suatu

pernyataan IF yang meggunakan operator !:

```
if (!(suhu > 100))
cout << "anda dibawah suhu
maksimum.\n";
```

pada potongan program diatas, ekspresi (suhu> 100) yang akan diuji .

benar atau salah. Maka operator ! akan diterapkan pada nilai. Jika ekspresi (suhu> 100) itu benar, maka operator ! akan membalik menjadi salah. Jika hal tersebut salah, maka operator ! akan kembali benar. Tabel dibawah merupakan table kebenaran operator !

Tabel 5.8. operator !

EKPRESI	HASIL EKPRESI	LOGIKA
! salah	benar	1
! benar	Salah	0

Program dibawah sama dengan program sebelumnya, yaitu akan melakukan operasi dengan operator !. Pernyataan IF tersebut menggunakan operator ! untuk menentukan apakah pengguna tidak membuat rekening di bank yang setidaknya ada $ 35.000 atau belum mempunyai pekerjaan yang lebih dari lima tahun. Perhatikan program dibawah ini:

Program 5.18

```
#include <iostream>

using namespace std;

int main()
{
        double income;
        int years;

        cout << "Apakah ini masukan tahunan Anda? ";
        cin >> income;
        cout << "berapa banyak anda bekerja setiap tahunnya "<< "Pekerjaan anda?
";
        cin >> years;

        if (!(income >= 35000 || years > 5))       // menggunakan operator !
```

```
        {
                cout << "Anda akan mendapatkan gaji paling tidak $35,000 atau
lebih\n";
                cout << "setelah bekerja lebih dari lima tahun.\n";
        }
        else
        cout << "Kwalifikasi Anda.\n";
        return 0;
}
```

Keluaran program diatas sama dengan program sebelumnya

5.8.3. Variabel Boolean dan Operator !

Fitur yang menarik dari variabel Boolean adalah nilainya dapat diuji. Misalnya moreData adalah sebuah variabel Boolean. Kemudian pengujiannya adalah:

```
if (moreData == true)
    can be written simply as
if (moreData)
    and the test
if(moreData == false)
    can be written simply as
if(!moreData)
```

Program diatas biasanya diselesaikan dengan menggunakan operator !.

5.9. Operator Kondisional

Operator kondisional merupakan operator kuat dan unik. Operator tersebut menyediakan sebuah metode sedernana yang mengekpresikan pernyataan IF/ELSE. Operator ini terdiri dari tanda tanya (?) dan titik dua (:). Untuk lebih jelasnya mengenai operator ini perhatikan ekpresi dibawah inI:

```
ekspresi ? ekspresi : ekspresi;
```

disini diberikan sebuah contoh pernyataan yang menggunakan operator kondisional

```
x < 0 ? y = 10 : z = 20;
```

pernyataan diatas sering disebut dengan *conditional expression dan terdiri dari tiga* sub-expressions ang dipisahkan dengan menggunakan tanda Tanya dan symbol titik dua (:). Sebuah ekspresi diatas < 0, y = 10, dan z = 20 dapat diuraikan sebagai berikut:

X< 0	?	Y=10	:	Z = 20

Gambar 5.6 conditional expression

Sebuah Ekspresi kondisional yang diatas melakukan operasi yang sama seperti halnya pernyataan IF/ELSE. Perhatikan pernyataan dibawah ini:

```
if (x < 0)
      y = 10;
else
      z = 20;
```

Bagian dari ekspresi kondisional sebelum tanda tanya adalah ekspresi yang akan diuji. Hal ini merupakan kalimat dalam kurung seperti pada sebuah pernyataan IF. Jika ekspresi benar, bagian dari pernyataan antara tanda tanya (?) dan titik dua (:) akan dijalankan. Jika tidak, bagian setelah : yang dijalankan. Gambar dibawah ini menggambarkan aturan yang diakukan oleh tiga sub-ekspresi adalah sebagai berikut:

Ekspresi pertama yang akan diuji		Ekspresi kedua yang akan dieksekusi jika ekspresi pertama benar		Ekspresi ketiga yang akan dieksekusi jika ekspresi pertama salah
X< 0	**?**	**Y=10**	**:**	**Z = 20**

Gambar 5.7. aturan yang diakukan oleh tiga sub-ekspresi

Anda dapat meletakkan tanda kurung di sekitar sub-ekspresi, seperti contoh berikut ini:

```
(x < 0) ? (y = 10) : (z = 20);
```

Dalam menggunakan nilai ekspresi dalam bahasa C++ yang perlu diingat, bahwa pada bahasa C++ semua ekspresi memiliki nilai, dan hal ini termasuk ekspresi kondisional. Jika sub-ekspresi pertama benar, maka nilai dari ekspresi kondisional adalah nilai kedua sub-ekspresi tersebut. Sebaliknya Jika hal tersebut bukan berarti nilai sub-ekspresi yang ketiga. Berikut dibawah ini adalah contoh sebuah pernyataan yang menggunakan nilai ekspresi kondisional

```
a = x > 100 ? 0 : 1
```

Nilai akan ditetapkan baik 0 atau 1, tergantung apakah x lebih besar dari 100. Pernyataan tersebut dapat dinyatakan dalam pernyataan IF/ELSE sebagai berikut:

```
if (x > 100)
a = 0;
else
a = 1;
```

Program dibawah dapat digunakan untuk membantu seorang konsultan untuk menghitung biaya. Konsultasn tersebut meminta bayaran $ 50,00 per jam, tetapi minimum selama lima jam, jika kurang dari 5 jam maka tidak akan dibayar. Operator kondisional digunakan dalam pernyataan tersebut untuk memastikan bahwa jumlahnya tidak kurang atau dibawah lima. Untuk lebih jelasnya perhatikan program dibawah ini

Program 5.19

```
#include<conio.h>
#include <iostream>
#include <iomanip>

using namespace std;

int main()
{
        const double PAY_RATE = 50.0;
        double hours, charges;

        cout << "Berapa jam anda bekerja? ";
        cin >> hours;

        hours = hours < 5 ? 5 : hours;   // Conditional operator
        charges = PAY_RATE * hours;

        cout << fixed << showpoint << setprecision(2);
        cout << "Besar gaji anda adalah $" << charges << endl;
        getch();
        return 0;
}
```

Program Output with Example Input Shown in Bold

```
Berapa jam anda bekerja? 5
Besar gaji anda adalah $250.00
```

5.10. Statement SWITCH

Statement SWITCH juga berfungsi sama dengan IF. Perintah SWITCH sama dengan perintah CASE OF dalam PASCAL. Dimana sintaksnya menggunakan:

```
switch (variabel)
{
      case value1 : statement1;
      break;
      case value2 : statement2;
      break;
      ..
      default : statement; /* optional */
      break;
}
```

Perhatikan contoh program menggunakan statement switch adalah sebagai berikut:

Program 5.20

```
#include <conio.h>
#include <iostream>

using namespace std;

void main()
{
          int bil;
          cout << "Masukkan bilangan : ";
          cin >> bil
       switch (bil)
          {
              case 1 : cout << "Anda memasukkan bil. satu";
              break;
              case 2 : cout << "Anda memasukkan bil. dua";
              break;
              case 3 : cout << "Anda memasukkan bil. tiga";
              break;
              default: cout << "Anda memasukkan bil selain 1, 2, dan 3";
              break;
          }
}
```

Selanjutnya coba kalian hapus semua break program di atas dan kalian jalankan. Apa yang terjadi? Keanehan akan muncul. Mengapa?

5.11. Pernyataan SWITCH … CASE

Cara lain untuk menangani pengambilan keputusan dalam sebuah program adalah dengan menggunakan Statement switch…. case, yang mampu menangani sejumlah kondisi dari satu variabel. switch .... case serupa dengan If ... Then ..... ElseIf, tetapi lebih efisien apabila percabangan bergantung kepada satu kondisi saja. Dengan menggunakan switch .... case sebagai pengganti dari If ... Then ... Else If, akan membuat program menjadi lebih sederhana. Format penggunaan switch .... case:

```
switch nama_variabel
{
  case nilai_1:{
      Perintah yang akan dieksekusi jika memenuhi nilai_1;
  break;}
  case nilai_2:{
      Perintah yang akan dieksekusi jika memenuhi nilai_2;
  break;}
  default:{
      Perintah yang akan dieksekusi jika tidak memenuhi semua;
   exit(0);}
}
```

Struktur Select Case dimulai dengan kata Select Case dan diakhiri dengan kata End Select. nama_variabel dapat diganti dengan variabel, properti, atau ekpresi lain yang akan dijadikan sebagai kondisi. Sedangkan nilai_1, nilai 2 dapat diganti dengan angka, string atau nilai lain yang berkaitandengan nama_variabel. Jika salah satu nilai sesuai dengan variabel, maka pernytaan di bawah kata case akan dijalankan dan Visual Basic akan melanjutkan mengeksekusi program setelah End. Anda dapat menggunakan kata case sebanyak mungkin dalam struktur select... case, dan anda juga dapat menyertakan lebih dari satu nilai pada kata case.

Jika anda menyertakan banyak nilai setelah case, pisahkan dengan tanda koma.

```
Select Case intAge
Case  5      : lblTitle.Caption = "Kindergarten"
Case 6       : lblTitle.Caption = "1st Grade"
```

```
Case 7      : lblTitle.Caption = "2nd Grade"
Case 8      : lblTitle.Caption = "3rd Grade"
Case 9      : lblTitle.Caption = "4th Grade"
Case 10     : lblTitle.Caption = "5th Grade"
Case 11     : lblTitle.Caption = "6th Grade"
Case Else: lblTitle.Caption = "Advanced"
End Select
```

Bentuk dari SWITCH - CASE merupakan pernyataan yang dirancangan khusus untuk menangani pengambilan keputusan yang melibatkan sejumlah atau banyak alternatif penyelesaian. Pernyataan SWITCH - CASE ini memiliki kegunaan sama seperti IF – ELSE bertingkat, tetapi penggunaannya untuk memeriksa data yang bertipe karakter atau integer. Bentuk penulisan perintah ini sebagai berikut :

```
switch (ekspresi integer atau karakter )
  {
    case konstanta-1 :
            … perintah;
            … perintah;
           break;
   case konstanta-2 :
           … perintah;
           … perintah;
           break;
           ……
           ……
    default :
           … perintah;
            … perintah;
  }
```

Setiap cabang akan dijalankan jika syarat nilai konstanta tersebut dipenuhi dan default akan dijalankan jika semua cabang diatasnya tidak terpenuhi. Pernyataan *break* menunjukan bahwa perintah siap keluar dari *switch*. Jika pernyataan ini tidak ada, maka program akan diteruskan kecabang – cabang yang lainnya. Perhatikan contoh program dibawah ini menggunakan Switch-Case, untuk menentukan alat yang dikehendaki, seperti alat olahraga, alat elektronik atau alat masak.

Program 5.21

```
include<stdio.h>
#include<conio.h>
#include<iostream.h>

using namespace std;

main()
{
  char kode;
  cout<<"Masukkan Kode Barang [A..C] : ";
  cin>>kode;

  switch(kode)
  {
   case 'A' :
        cout<<"Alat Olah Raga";
        break;
   case 'B' :
        cout<<"Alat Elelktronik";
        break;
   case 'C' :
        cout<<"Alat Masak";
   break;
   default:
        cout<<"Anda Salah Memasukan kode";
    break;
  }
  getch();
}
```

Keluaran program diatas adalah:

```
Masukkan Kode Barang [A..C] : B
Alat Elelktronik
```

Program diatas juga bisa ditulis dengan program dibawah ini, dimana kelebihan program dibawah bisa membaca baik kode huruf kecil maupun huruf besar. Pada program diatas hanya bisa membaca kode huruf besar saja. Kedua-duanya sama-sama menggunakan switch-case. Jika dalam memasukan kodenya salam maka akan muncul pesan “Anda Salah Memasukan kode”. Untuk lebih jelasnya perhatikan program dibawah ini:

Program 5.22

```
#include<stdio.h>
#include<conio.h>
#include<iostream.h>

using namespace std;

main()
{
  char kode;
  cout<<"Masukkan Kode Barang [A..C] : ";
  cin>>kode;

  switch(kode)
  {
   case 'A' :
   case 'a' :
         cout<<"Alat Olah Raga";
         break;
    case 'B' :
   case 'b' :
         Cout<<"Alat Elelktronik";
         break;
   case 'C' :
   case 'c' :
         cout<<"Alat Masak";
     break;
   default:
         cout<<"Anda Salah Memasukan kode";
     break;
  }
  getch();
  return 0;
}
```

Struktur kondisi switch....case.... default digunakan untuk penyeleksian kondisi dengan cukup banyak kemungkinan yang terjadi. Struktur ini akan melaksanakan salah satu dari beberapa pernyataan 'case' tergantung nilai kondisi yang ada di dalam switch. Selanjutnya proses diteruskan hingga ditemukan pernyataan 'break'. Jika tidak ada nilai pada case yang sesuai dengan nilai kondisi, maka proses akan diteruskan kepada pernyataan yang ada di bawah 'default'. Bentuk umum dari struktur kondisi ini adalah:

```
switch(kondisi)
{
      case 1 : pernyataan-1;
      break;
      case 2 : pernyataan-2;
      break;
      .....
      .....
      case n : pernyataan-n;
      break;
      default : pernyataan-m
}
```

Pada program dibawah merupakan program dengan menggunakan switch, dimana program tersebut digunakan untuk menentukan jumlah hari. Perhatikan program dibawah ini:

Program 5.23

```
#include <conio.h>
#include <iostream>

using namespace std;

int main() {
int AngkaBulan,Tahun, JumlahHari;

   cout<<"Angka Bulan : ";
   cin >> AngkaBulan;

   cout <<"Tahun : ";
   cin >> Tahun;
   switch (AngkaBulan) {
                case 1:
                case 3:
                case 5:
                case 7:
                case 8:
                case 10:
                case 12: JumlahHari = 31;break;
                case 4:
                case 6:
                case 9:
```

```
            case 11 : JumlahHari = 30;break;
            case 2 : if (Tahun % 4 == 0) JumlahHari = 29;
            else JumlahHari = 28;break;
    }
    cout<<"Jumlah hari "<< JumlahHari;
    getch();
    return 0;
}
```

Keluaran program diatas adalah:

```
Angka Bulan : 4
Tahun : 1990
Jumlah hari 30
```

Program dibawah merupakan program yang digunakan untuk menentukan bilangan Genap atau Ganjil dengan case:

Program 5.24

```
#include <conio.h>
#include <iostream>

using namespace std;

main() {
        int Op1, Op2, Pilih;
        float Hasil;
        cout << "Operand 1: ";
        cin >> Op1;
        cout << "Operand 2: ";
        cin >> Op2;
        cout << "1. Operator +\n";
        cout << "2. Operator -\n";
        cout << "3. Operator *\n";
        cout << "4. Operator /\n";
        cout << "Pilih Operator : ";
        cin  >> Pilih;
        switch (Pilih) {
                case 1 : {Hasil = Op1 + Op2; break;}
                case 2 : {Hasil = Op1 - Op2; break;}
                case 3 : {Hasil = Op1 * Op2; break;}
                case 4 : {if (Op2 != 0)
                cout <<Op1/Op2;
```

```
                    else
                    cout <<"error";
      break;}
   }
   printf("Hasil dari perhitungan tersebut adalah %f",Hasil);
   getch();
   return 0;
}
```

Keluaran program diatas adalah sebagai berikut:

```
Operand 1: 1
Operand 2: 3
1. Operator +
2. Operator -
3. Operator *
4. Operator /
Pilih Operator : 3
Hasil dari perhitungan tersebut adalah 3.000000
```

Dari beberapa contoh program diatas, maka dapat diketahui bentuk bahasa umum mengenai pernyataan If–Else adalah sebagai berikut:

```
If Kondisi  Then
      Perintah yang akan dieksekusi jika kondisi bernilai true
ELSE
      Perintah yang akan dieksekusi jika kondisi bernilai false
End If
```

5.12. IF...THEN, IF...THEN...ELSE dan Nested IF

Struktur IF, kita dapat membuat algoritma-algoritma yang melakukan perintah berdasarkan kondisi tertentu. Perintah akan dilakukan jika hanya jika ekspresi boolean bernilai true (jika ekspresi boolean bernilai false maka perintah tidak akan dikerjakan).

```
IF <ekspresi boolean> then
      Perintah_1
Else
         Perintah_2
      Endif
         IF <ekspresi boolean 1> then
         Perintah 1
```

```
    Else if <ekspresi boolean 2> then
        Perintah 2
    Else if <ekspresi boolean n> then
        Perintah n
    Else Perintah n+1
Endif
```

Program statement kondisi IF-ELSE untuk mencari bilangan Genap dan Ganjil. Perhatikan contoh program dibawah ini:

Program 5.25

```
#include <conio.h>
#include <iostream>

using namespace std;

main() {
        int Bil;
        cout << "masukan bilangan : ";
  cin >> Bil;
        if (Bil % 2 == 0)
                {
       cout << "bilangan genap";
    }
        else
                {
       cout << "bilangan ganjil";
    }
  getch();
  return 0;
}
```

Keluaran program diatas adalah

```
masukan bilangan : 5
bilangan ganjil
```

Program dibawah ini digunakan untuk menghitung atau mencari bilangan terbesar.

Program 5.26

```
#include <conio.h>
#include <iostream>
```

```
using namespace std;

main() {
        int A, B, C,maks;
        cout << "A : ";
        cin >> A;
        cout << "B : ";
        cin >> B;
        cout << "C : ";
        cin >> C;
        if ((A>B) && (A>C))
                maks = A;
        else if ((B>A) && (B>C))
                maks = B;
        else
                maks = C;
                cout <<"bilangan terbesar adalah " << maks;
        getch();
        return 0;
}
```

Keluaran program diatas adalah sebagai berikut:

```
A : 40
B : 56
C : 59
bilangan terbesar adalah 59
```

5.13. Aplikasi Pernyataan IF pada Menu

Menu adalah layar yang menampilkan beberapa pilihan sehingga pengguna dapat memilih. Sebagai contoh pada sebuah program yang dapat memberikan menu pilihan berikut ini:

1. Menambahkan nama ke dalam daftar.
2. Hapus nama dari daftar.
3. Mengubah nama dalam daftar.
4. Mencetak daftar.
5. Keluar dari program ini.

Pengguna dapat memilih salah satu operasi dengan memasukkan nomor. Misalnya memasukan nomer 4, sehingga akan menyebabkan daftar dicetak, dan memasukkan 5 akan menyebabkan keluar program. struktur IF/ELSE IF dapat digunakan untuk membuat menu seperti diatas. Setelah pengguna memasukkan nomor, maka nomor akan digunakan untuk membandingkan pilihan yang tersedia dan melaksanakan sebuah pernyataan yang melakukan operasi. program dibawah digunakan untuk menghitung biaya keanggotaan disebuah klub kesehatan. Klub memiliki tiga paket keanggotaan pilihan: keanggotaan standar

dewasa, anggota anak, dan anggota orang senior.

Program ini menyajikan menu yang memungkinkan pengguna untuk memilih paket yang diinginkan, kemudian menghitung biaya keanggotaannya. Untuk lebih jelasnya perhatikan program dibawah ini:

Program 5.27

```
#include <conio.h>
#include <iostream>
#include <iomanip>

using namespace std;

int main()
{
   int choice, months;
   double charges;

   // Display menu pilihan
   cout << "\t\tMenu Anggota Klub Kesehatan\n\n";
   cout << "1. Keanggotaa Standard Dewasa\n";
   cout << "2. Keanggotaan Anak\n";
   cout << "3. Keanggotaan Masyarakat Senior\n";
   cout << "4. Keluar Program\n\n";
   cout << "Masukan Pilihan Anda: ";
   cin >> choice;
   cout << fixed << showpoint << setprecision(2);

   if (choice == 1)
   {
      cout << "Untuk berapa Bulan? ";
      cin >> months;
      charges = months * 40.00;
      cout << "Total pembayaranya adalah: $" << charges << endl;
   }
   else if (choice == 2)
   {
      cout << "Untuk berapa Bulan? ";
      cin >> months;
      charges = months * 20.00;
      cout << "Total pembayaranya adalah: $" << charges << endl;
      }
   else if (choice == 3)
```

```
  {
     cout << "Untuk berapa Bulan? ";
     cin >> months;
     charges = months * 30.00;
     cout << "Total pembayaranya adalah: $" << charges << endl;
  }
  else if (choice != 4)
  {
     cout << "Pilihan yang valid adalah antara 1 sampai 4\n";
     cout << "Program akan memilih lagi dari salah satu menu diatas\n";
  }
  getch();
  return 0;
}
```

Keluaran program diatas seperti dibawah ini:

```
Menu Anggota Klub Kesehatan

1. Keanggotaa Standard Dewasa
2. Keanggotaan Anak
3. Keanggotaan Masyarakat Senior
4. Keluar Program

Masukan Pilihan Anda: 3
Untuk berapa Bulan? 3
Total pembayaranya adalah: $90.00
```

Dalam program tersebut memungkinkan pengguna yang tidak valid dapat membuat pilihan. Jika angka lain seperti 1, 2, 3, atau 4 dimasukkan, maka akan ada pesan kesalahan yang dicetak. Kesalahan ini dikenal sebagai masukan validasi (*input validation).*

5.14. Soal Latihan

Jawablah soal latihan dibawah ini dengan baik dan benar.

1. Apa yang dimaksud dengan statement
2. Sebutkan beberapa operator relasionaldalam bahasa c++
3. Jelaskan cara kerja pernyataan if pada bahasa c++
4. Apa perbedaan antara pernyataan if dangan if-else
5. Apa yang dimaksud dengan if-else majemuk
6. Apa yang dimaksud dengan nested if
7. Sebutkan beberapa operator logika
8. Tuliskan perintah yang digunakan oleh pernyataan switch

BAB 6
PROSEDUR DAN SUBROUTIN

6.1. Prosedur

Prosedur adalah sederetan instruksi algoritmik yang diberi nama, dan akan menghasilkan efek neto yang terdefinisi. Prosedur menyatakan suatu aksi dalam konsep algoritma yang dibicarakan pada cerita “Mengupas kentang”. Dimana contoh ini dari aksi yang dilakukan pleh Ibu Tati yang mengupas kentang untuk mempersiapkan makan malam sebagai berikut. Pernyataan ini mencakup hal yang luas ruang lingkupnya, misalnya :

- Apakah kentangnya harus dibeli dulu atau sudah ada di dapur ?
- Apakah yang dimaksud dengan mengupas kentang untuk makan malam berarti sampai dengan kentang terhidang ?
- Ketika kentangnya terhidang, jadi sup, digoreng atau direbus saja ?

Maka kita harus membatasi dengan jelas keadaan awal yang menjadi titik tolak mengupas kentang dan keadaan akhir yang ingin dicapai supaya dapat “merencanakan” efek neto yang diinginkan. Sehingga hal tersebut dapat ditentukan:

- *Initial state* (I.S. keadaan awal), T0, adalah kentang sudah ada di kantong kentang, yang ditaruh di rak di dapur, di mana ibu Tati akan mengupasnya
- *Final state* (F.S. keadaan akhir), T1, kentang dalan keadaan terkupas di panci, siap untuk dimasak dan kantong kentangnya harus dikembalikan ke rak lagi.

Pengandaian yang lain adalah bahwa persediaan kentang di ibu selalu cukup untuk makan malam. Penambahan kentang ke dapur di luar tinjauan masalah ini. Ini adalah contoh bagaimana kita menentukan batasan dari persoalan yang akan diprogram. Suatu kejadian dapat dipandang sebagai urut-urutan dari beberapa kejadian, berarti dapat diuraikan dalan beberapa (sub) aksi yang terjadi secara sekuensial. Dengan sudut pandang ini makan efek kumulatifnya sama dengan efek neto dari seluruh kejadian. Dikatakan bahwa kejadian tersebut dianggap sebagai sequential process atau disingkat proses.

Mendefinisikan (membuat spesifikasi) prosedur berarti menentukan nama prosedur serta parameternya (jika ada), dan mendefinisikan keadaan awal (Initial State, I.S.) dan keadaan akhir (Final State, F.S.) dari prosedur tersebut. Prosedur didefinisikan (dituliskan spesifikasinya) dalam kamus. Cara penulisan spesifikasi : prosedur diberi nama, dan parameter formal (jika ada) yang juga diberi nama dan dijelaskan typenya.

Secara sederhana, dapat diartikan bahwa sebuah prosedur yang terdefinisi “disimpan” di tempat lain, dan ketika “dipanggil” dengan menyebutkan namanya “seakan-akan” teks yang tersimpan di tempat lain itu menggantikan teks pemanggilan. Pada saat itu terjadi asosiasi parameter (jika ada). Dengan konsep ini, maka I.S dan F.S dari prosedurlah yang menjamin bahwa eksekusi program akan menghasilkan efek neto yang diharapkan. Jadi, setiap prosedur harus :

- Didefinisikan (dibuat spesifikasinya) dan dituliskan kode programnya
- Dipanggil, pada saat eksekusi oleh prosedur lain atau oleh program utama

6.2. Parameter Prosedur

Prosedur tanpa parameter memanfaatkan nilai dari nama-nama yang terdefinisi pada kamus global. Pemakaiannya biasanya harus “hati-hati”, apalagi jika teks program sudah sangat besar dan implementasinya menjadi banyak file. Prosedur berparameter dirancang, agar sepotong kode yang sama ketika eksekusi dilakukan, dapat dipakai untuk nama parameter yang bebeda-beda. Nama parameter yang dituliskan pada definisi/spesifikasi prosedur disebut sebagai parameter formal. Sedangkan parameter yang dituliskan pada pemanggilan prosedur disebut sebagai parameter aktual.

Parameter formal adalah nama-nama variabel (list nama) yang dipakai dalam mendefinisikan prosedur, dan membuat prosedur tersebut dapat dieksekusi dengan nama-nama yang berbeda ketika dipanggil. Parameter formal adalah list nama yang akan dipakai pada prosedur, yang nantinya akan diasosiasikan terhadap nama variabel lain pada saat pemanggilan. Sesuai dengan ketentuan nilainya, ada tiga type parameter formal:

- parameter Input, yaitu parameter yang diperlukan prosedur sebagai masukan untuk melakukan aksi yang efektif.
- parameter Output, yaitu parameter yang nilainya akan dihasilkan oleh prosedur. Hasil nilai akan disimpan pada nama parameter Output ini.

parameter Input/Output, yaitu parameter yang nilainya diperlukan prosedur sebagai masukan untuk melakukan aksi, dan pada akhir prosedur akan dihasilkan nilai yang baru.

6.3. Pemanggilan Prosedur

Memakai, atau "memanggil" prosedur adalah menuliskan nama prosedur yang pernah didefinisikan, dan memberikan harga-harga yang dibutuhkan oleh prosedur itu untuk dapat melaksanakan suatu aksi terdefinisi. Sebuah prosedur juga boleh "memakai" atau memanggil prosedur. Pada saat pemanggilan terjadi “passing parameter”. **Parameter aktual** adalah nama-nama informasi yang dipakai ketika prosedur itu dipakai ("dipanggil"). Parameter aktual dapat berupa nama atau harga, tetapi harus berupa nama jika parameter tersebut adalah parameter Output (karena hasilnya akan disimpan dalam nama tersebut). Sesuai dengan jenis parameter formal, parameter aktual pada saat pemanggilan :

- parameter input harus terdefinisi nilainya (karena dibutuhkan oleh prosedur untuk menghasilkan nilai). Karena yang dibutuhkan untuk eksekusi hanya nilai, maka parameter input dapat digantikan dengan suatu nilai tanpa menggunakan nama.
- parameter output tidak perlu terdefinisi nilainya, tetapi justru setelah pemanggilan prosedur akan dimanfaatkan oleh deretan instruksi berikutnya, karena nilainya akan dihasilkan oleh prosedur. Karena parameter output menampung hasil, maka harus berupa nama, dan tidak boleh diberikan nilai saja.
- parameter input/output harus terdefinisi nilainya dan nilai baru yang diperoleh karena eksekusi prosedur akan dimanfaatkan oleh deretan instruksi berikutnya. Seperti halnya parameter output, maka paramteter aktual harus berupa nama.

Pada saat eksekusi, terjadi asosiasi nama parameter formal dengan nama parameter aktual. Pada notasi algoritmik, asosiasi dilakukan dengan cara “by position”. Urutan nama pada parameter aktual akan diasosiasikan sesuai dengan urutan parameter formal. Karena itu, type harus kompatibel. Beberapa bahasa pemrograman, dapat dilakukan asosiasi dengan nama dan memperbolehkan adanya *nilai default*. Beberapa bahasa pemrograman (Ada, C) juga memperbolehkan nama prosedur yang sama tetapi parameternya berbeda (*overloading*).

Prosedur dapat mempunyai kamus lokal, yaitu pendefinisan nama yang dipakai dan hanya berlaku dalam ruang lingkup prosedur

tersebut. Jika nama yang dipakai di dalam prosedur tidak terdefinisi dalam list parameter formal atau dalam kamus lokal, maka nama tersebut harus sudah terdefinisi pada prosedur yang memakainya. Penulisan kamus lokal sama dengan kamus global, yang berbeda adalah lingkup berlakunya nama yang didefinisikan:

- pada kamus "global", nama berlaku untuk program dan semua prosedur/fungsi yang didefinisikan.
- pada kamus lokal, nama berlaku untuk prosedur/fungsi yang bersangkutan dan prosedur / fungsi yang didefinisikan di dalamnya.
- nilai yang disimpan dalam nama yang didefinisikan pada kamus lokal, hanya akan terdefinisi selama eksekusi prosedur, dan tidak dikenal lagi oleh pemanggilnya.

Program yang moduler adalah program yang dibagi-bagi menjadi modul-modul yang terdefinisi dengan baik dalam bentuk prosedur-prosedur. Setiap prosedur harus jelas definisi dan ruang lingkupnya, supaya dapat dipanggil secara independent. Pembagian program besar dalam prosedur-prosedur akan mempermudah pembagian kerja di antara beberapa pemrogram. Penulisan prosedur juga akan memudahkan program untuk dibaca oleh "manusia" karena kita tidak perlu terpaku pada detil kode prosedur untuk mengerti efek neto yang dihasilkannya.

Bahkan dalam beberapa hal, pemrogram tidak perlu tahu sama sekali "isi" atau kode dari prosedur dengan mengetahui spesifikasinya, beberapa bahasa pemrograman bahkan menyediakan prosedur terdefinisi yang sering dipakai dalam memrogram sehingga pemrogram tidak perlu lagi menuliskan kodenya. Prosedur berlaku untuk ruang lingkup (*universe*) tertentu, terutama untuk prosedur yang tidak mempunyai parameter. Dalam dua bab berikut, yaitu Mesin Gambar dan Mesin Karakter, akan diberikan gambaran lebih jelas dan lengkap tentang pendefinisian dan pemakaian prosedur karena keduanya adalah mesin abstrak yang tertentu.

Contoh 1-Prosedur: VOLTAGE

Tuliskanlah program yang membaca tahanan (Ohm) dan arus (Ampere), kemudian menghitung tegangan yang dihasilkan dan menuliskan hasilnya. Perhitungan tegangan harus dituliskan menjadi suatu prosedur bernama PROSES, supaya struktur program jelas: Input - Proses – Output.

```
Input   : R : integer, tahanan (Ohm) dan A : integer, arus (Ampere)
Proses  : menghitung V= R * A
Output  : V : integer, tegangan (Volt)
```

Pelajarilah dua buah solusi yang diberikan berikut ini, dan berikan komentar anda.

Solusi 1 : Prosedur tanpa parameter

Program VOLTAGE1

```
{Program yang membaca tahanan dan arus, menghitung Voltage dan
mencetak hasil perhitungan}

Kamus :
R       : integer { tahanan dalam ohm}
A       : integer { arus dalam ohm}
V       : integer { tegangan dalam ohm}
```

procedure PROSES1

```
{ Prosedur untuk "memproses" tahanan dan arus menjadi tegangan }

Algoritma :
input (R,A)
PROSES1
output (V)
```

Procedure PROSES1

```
{ I.S     : diberikan harga R dan A yang telah terdefinisi}
{ FS      : memproses R dan A sehingga dihasilkan V yaitu tegangan dengan
rumus   : V = R * A}
Kamus lokal :

Algoritma :
V ← R * A
```

Solusi 2 : Prosedur dengan parameter

Program VOLTAGE2

```
{Program yang membaca tahanan dan arus, menghitung Voltage dan
mencetak hasil perhitungan}

Kamus :
R       : integer { tahanan dalam ohm}
A       : integer { arus dalam ohm}
V       : integer { tegangan dalam ohm}
```

Procedure PROSES2

```
(Input : R,A : integer; Output V:integer)
{ Prosedur untuk "memproses" tahanan R dan arus A menjadi
tegangan V}

Algoritma :
input (R,A)
```

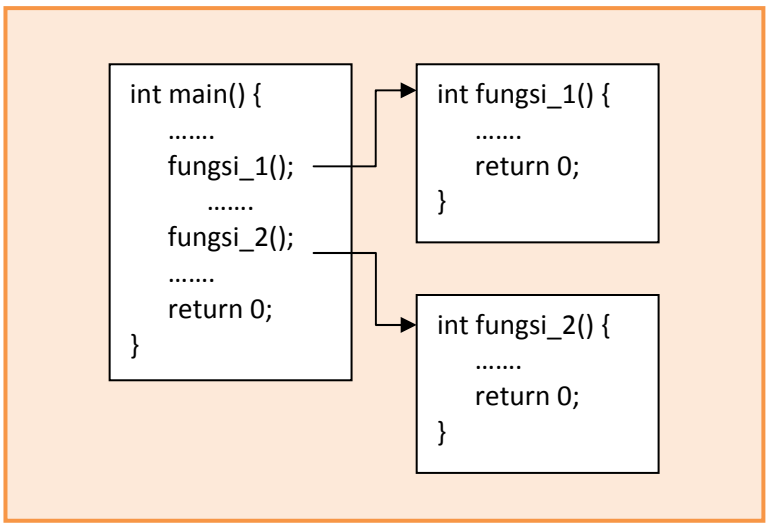
PROSES2 (R,A,V)
output (V)

6.4. Sub Routin

Suatu program komputer biasanya merupakan suatu sistem besar yang terdiri dari sub sistem - sub sistem yang mempunyai tugas sendiri-sendiri, saling bekerja sama dan berinteraksi untuk menyelesaikan suatu permasalahan. Dengan adanya pembagian tugas oleh sub sistem – sub sistem ini maka suatu permasalahan dapat diselesaikan dengan lebih cepat dan kesalahan-kesalahan yang mungkin terjadi selama proses penyelesaian masalah dapat dideteksi dan diketahui sedini mungkin, termasuk di sub sistem mana kesalahannya terjadi.

Program komputer (terutama yang komplek) juga sebaiknya "dipecah-pecah" menjadi program-program kecil. Program-program kecil tersebut disebut dengan sub rutin. Sub rutin dibagi menjadi dua macam, yaitu sub rutin yang mengembalikan nilai dan sub rutin yang tidak mengembalikan nilai. Dalam Pascal kedua sub rutin yang mengembalikan nilai disebut dengan function, sedangkan yang tidak mengembalikan nilai disebut dengan procedure. Tetapi untuk C++ dan Java, kedua sub rutin tersebut dijadikan satu tetapi dapat diatur untuk dapat mengembalikan nilai maupun tidak mengembalikan nilai.

Untuk C++, sub rutin tersebut ada dalam suatu function, sedangkan pada Java, sub rutin berbentuk suatu method yang disebut dengan function method.



Gambar 6.1.SubRutin

Baik C++ maupun Java terdapat dua macam fungsi, yaitu user-defined function dan built-in function. User-defined function merupakan fungsi yang didefinisikan sendiri atau dibuat sendiri oleh pemrogram. Sedangkan built-in function adalah fungsi yang sudah ada atau sudah disediakan oleh kompiler dan siap digunakan oleh pemrogram.

6.5. Sub Rutin dalam Bahasa Pemrograman

Bentuk umum sub rutin (function) pada C++ dan Java sangat mirip. Untuk function yang mengembalikan nilai, setiap function harus didahului oleh tipe data yang sesuai dengan jenis data yang akan dikembalikan, kecuali tipe data array. Setiap function juga mempunyai daftar parameter dimana setiap parameter dipisahkan dengan tanda koma (,). Parameter-parameter ini digunakan untuk menerima data (nilai) dari program yang memanggilnya.

Kita dapat mendeklarasikan banyak parameter atau tidak sama sekali. Untuk fungsi yang tidak mempunyai parameter merupakan fungsi yang nilainya tetap (tidak berubah). Sedangkan fungsi yang mempunyai parameter nilai fungsinya dinamis (dapat berubah-ubah). Parameter-parameter ini merupakan variabel-variabel yang harus dideklarasikan sendiri-sendiri meskipun tipe datanya sama.

Bentuk umum dari function yang mengembalikan nilai adalah sebagai berikut

```
tipe_data nama_fungsi(daftar_parameter)
{
    isi dari fungsi
    return<ekspresi>
}
```

Perhatikan contoh progam function yang benar

```
int contoh(int a, int b) {
    ……..
    return(c);
}
```

Sedangkan program yang menggunakan function salah adalah sebagai berikut:

```
int contoh(int a, b) {
    ……..
    return(c);
}
```

Setiap function mempunyai kode-kode program sendiri-sendiri karena tugas yang harus diselesaikan oleh setiap function juga

berbeda-beda. Variabel-variabel yang digunakan dalam function disebut dengan variabel lokal. Oleh karena itu variabel tersebut hanya dapat digunakan didalam function itu sendiri tidak bisa digunakan oleh function lain atau program utama.

Variabel lokal ini berfungsi pada saat function tersebut aktif dan akan hilang (dihapus) jika function sudah tidak aktif lagi atau setelah function selesai melakukan tugasnya (kecuali variabel yang digunakan dalam function adalah variabel global yang dapat digunakan oleh semua function dan program utama).

Untuk dapat digunakan, function biasanya mempunyai parameter-parameter yang digunakan untuk menerima masukan dari program yang memanggilnya. Parameter-parameter ini disebut dengan parameter formal. Parameter formal ini termasuk dalam variabel lokal yang akan berfungsi pada saat function aktif dan akan dihapus pada saat function selesai melakukan tugasnya. Perhatikan contoh function dalam bahasa C++ berikut ini:

```
/* function akan menghasilkan nilai 1 jika c sama dengan s, sebaliknya bernilai 0 jika c tidak sama dengan s */
int cek(char s, char c) {
    if(s==c) return 1;
    return 0;
}
```

Function cek() mempunyai dua buah parameter formal, yaitu s dan c yang mempunyai tipe data yang sama, char. Function cek() merupakan function yang mengembalikan nilai sehingga dia harus mempunyai tipe data dimana tipe data-nya dalam hal ini adalah integer.

Function cek() ini bertugas untuk memeriksa apakah variabel c sama dengan variabel s. Jika sama maka function cek() akan bernilai 1, sebaliknya jika tidak maka akan bernilai 0. Variabel s dan c ini merupakan variabel lokal yang hanya dapat digunakan dalam function cek() saja, tidak bisa digunakan oleh function atau program lain. Permasalahan yang sama untuk Java adalah sebagai berikut:

```
class ricek {
    public int cek(char s, char c) {
        if(s==c) return 1;
        return 0;
    }
}
```

Telah disebutkan sebelumnya bahwa sub rutin dalam Java berbentuk class dimana didalam class tersebut dimungkinkan untuk mempunyai satu atau lebih function method. Class ricek() ini dapat

digunakan oleh class lain, dalam hal ini untuk mengecek suatu karakter karena class ricek() mempunyai function method cek(). Berikut ini adalah kode lengkap dari kedua program di atas

.

Perhatikan Program dalam bahasa C++ :

```
#include <iostream>
using namespace std;

int cek(char s, char c) {
        if(s==c) return 1;
        return 0;
}
int main(void) {
        char a,b;
        a = 'a';
        b = 'a';
        cout << cek(a,b) << endl;
        return 0;
}
```

Keluaran program adalah :

```
1
```

Dari kode program C++ di atas terlihat bahwa function cek() dipanggil melalui argumen pada program utama (main()) dimana argumen tersebut mempunyai variabel masukan untuk function cek() yaitu variabel a dan b.

Variabel a dan b ini kemudian disalin oleh parameter formal function cek() yaitu s dan c. Parameter formal ini kemudian diproses lebih lanjut yaitu pengecekan apakah parameter formal s sama dengan parameter formal c. Jika sama maka function cek() akan bernilai 1, sebaliknya jika tidak sama maka function cek() akan bernilai 0. Nilai dari function cek() ini kemudian langsung dicetak ke layar oleh program utama. Kode program untuk bahasa Java dari permasalahan yang sama adalah:

```
class ricek {
        public int cek(char s, char c) {
          if(s==c) return 1;
          return 0;
        }
}
class ricekApp {
        public static void main
```

```
(String[ ] args) {
          ricek ck = new ricek();
          char a='a',b='b';
          System.out.println(ck.cek(a,b));
        }
}
```

Keluaran program adalah :

```
0
```

Dalam Java, class ricek() yang mempunyai function method cek() tidak bisa langsung kita gunakan (kita panggil), tetapi harus dibuat / diciptakan dahulu obyek dari class ricek() seperti yang terlihat pada baris ke-9 dimana obyek baru dari class ricek() dalam class ricekApp() diberi nama ck. Dengan obyek ck inilah kita dapat mengakses method yang dipunyai oleh class ricek() karena secara otomatis obyek ck juga mempunyai method cek() yang dimiliki oleh class ricek().

Contoh di atas merupakan contoh function dimana function tersebut mempunyai parameter, dalam hal ini parameter s dan c yang bertipe char. Function cek() tersebut nilainya akan berubah-ubah sesuai dengan nilai masukannya. Untuk contoh di atas nilai function dari cek() cuma ada dua, yaitu 0 atau 1. Berikut ini adalah contoh function yang tidak mempunyai parameter sama sekali sehingga nilai function-nya tidak akan berubah-ubah seperti halnya pada contoh sebelumnya.

Contoh program dalam bahasa C++ :

```
#include <iostream>
using namespace std;
int fpb(){
        int a=24,b=18,hasil;
        int r = a % b;
        if (r==0) hasil = b;
        else {
          while(r!=0) {
            a = b;
            b = r;
            r = a % b;
            hasil = b;
          }
        }
        return(hasil);
}
void main() {
```

```
        cout << "FPB-nya = ";
        cout << fpb() <<endl;
}
```

Keluaran programnya adalah :

```
FPB-nya = 6
```

Contoh program dalam bahasa Java :

```
1.  class hitung {
2.     public int fpb(){
3.       int a=78,b=24,hasil=0;
4.       int r = a % b;
5.       if (r==0) hasil = b;
6.       else {
7.         while(r!=0) {
8.           a = b;
9.           b = r;
10.          r = a % b;
11.          hasil = b;
12.        }
13.     }
14.     return hasil;
15.  }
16. }
17. class fpbApp {
18.    public static void main(String[ ] args) {
19.      hitung sekutu = new hitung();
20.      System.out.println("Bilangan terbesarnya="+ sekutu.fpb());
21.   }
22. }
```

Keluaran programnya adalah :

```
Bilangan terbesarnya = 6
```

Nilai dari function fpb() di atas untuk bahasa C++ pasti 6 karena nilai dari a dan b telah ditetapkan besarnya, yaitu 24 dan 18. Sedangkan untuk bahasa Java nilai method fpb() dari class hitung() juga pasti tetap, yaitu 6 karena nilai a dan b juga telah ditentukan (a=78 dan b=24).

6.6. Function yang Mengembalikan Nilai

Yang dimaksud dengan function yang mengembalikan nilai adalah suatu sub rutin yang bila dipanggil oleh suatu program (argumen) maka argumen tersebut akan memperoleh nilai balikan dari function tersebut.

Atau dengan kata lain, suatu function yang mempunyai nilai.

Karena mempunyai nilai inilah maka suatu function yang mengembalikan nilai harus mempunyai tipe data sesuai dengan nilai yang dihasilkannya. Perhatikan baris ke-2 pada function cek() pada contoh sebelumnya. Function cek() digunakan menghasilkan nilai integer (lihat baris ke-3 dan ke-4) yaitu 0 atau 1 (return 0 dan return 1), sehingga tipe data-nya juga harus integer.

Dengan demikian dapat disimpulkan bahwa ciri dari function yang mengembalikan nilai adalah :

- Function tersebut mempunyai tipe data.
- Diakhiri dengan klausa return.

Berikut contoh program C++ yang menggunakan function dimana function-nya dapat mengembalikan nilai

```
1.  #include <iostream>
2.  using namespace std;
3.  int fpb(int a, int b) {
4.      int hasil;
5.      int r = a % b;
6.      if (r==0) hasil = b;
7.      else {
8.          while(r!=0) {
9.              a = b;  b = r;
10.             r = a % b;
11.             hasil = b;
12.         }
13.     }
14.     return(hasil);
15. }
16. void main() {
17.     int m,n;
18.     do {
19.         cout << "Bilangan pertama = ";
20.         cin >> m;
21.         cout << "Bilangan kedua   = ";
22.         cin >> n;
23.     } while (m < n);
24.     cout << "FPB-nya = " << fpb(m,n)<<endl;
25. }
```

Keluaran programnya :

```
Biangan pertama     = 24
Bilangan kedua      = 20
FPB-nya             = 4
```

Baris ke-3 sampai dengan ke-16 merupakan sub rutin (function) yang bernama fpb(). Sedangkan baris ke-17 sampai dengan ke-26 merupakan program utamanya. Program utama ini akan memanggil function fpb() melalui suatu argumen (lihat baris ke-25). Function fpb() bertugas untuk melakukan pencarian faktor persekutuan besar dari dua buah bilangan yang dimasukkan di program utama (lihat baris ke-19 sampai dengan ke-24). Setelah selesai melakukan tugasnya, maka function fpb() akan mempunyai nilai yang langsung ditampilkan pada program utama.

Function fpb() mempunyai tipe data integer dan mempunyai dua buah parameter formal yang bertipe data integer juga, yaitu a dan b (baris ke-3). Function tersebut juga mempunyai variabel hasil yang bertipe data integer (baris ke-4). Function fpb() ini nilainya akan sama dengan variabel hasil (baris ke-15).

Variabel a, b, dan hasil merupakan variabel lokal dimana ketiga variabel ini hanya berfungsi pada function fpb() saja. Variabel a dan b bertugas untuk menerima data yang dikirim oleh program lain yang memanggilnya sedangkan variabel hasil digunakan untuk menyimpan data hasil pencarian faktor persekutuan besar (baris ke-6 dan baris ke-12). Untuk permasalahan yang sama dengan menggunakan bahasa Java adalah sebagai berikut:

```
1.  import java.util.Scanner;
2.  import java.io.*;
3.  class hitung {
4.      public int fpb(int a, int b) {
5.        int hasil=0;
6.        int r = a % b;
7.        if (r==0) hasil = b;
8.        else {
9.          while(r!=0) {
10.           a = b;
11.           b = r;
12.           r = a % b;
13.           hasil = b;
14.         }
15.     }
16.     return hasil;
17.   }
18. }
19. class sekutuBesar {
20.   public static void main(String[ ] args) {
21.       hitung sekutu = new hitung();
22.       int m,n;
```

```
23.         Scanner input = new Scanner(System.in);
24.         do {
25.           System.out.print("Bilangan pertama = ");
26.           m = input.nextInt();
27.           System.out.print("Bilangan kedua   = ");
28.           n = input.nextInt();
29.         } while(m < n);
30.         System.out.println("Bilanganterbesarnya="+sekutu.fpb(m,n));
31.     }
32. }
```

Keluaran programnya :

```
Bilangan pertama      = 36
Bilangan kedua        = 28
Bilangan terbesarnya  = 4
```

Pada program Java di atas terlihat bahwa pencarian faktor persekutuan besar-nya dilakukan oleh Class hitung(). Class hitung() ini mempunyai method fpb(), yang merupakan function fpb() pada C++, dan bertugas untuk melakukan pencarian faktor persekutuan besar. Class sekutuBesar() kemudian membuat obyek baru dari Class hitung() dengan nama sekutu (lihat baris ke-21). Atau dengan kata lain Class hitung() "dipanggil" oleh program utamanya yaitu Class sekutuBesar().

Program Java tersebut jika di-compile akan menghasilkan dua buah class, yaitu hitung.class dan sekutuBesar.class dimana program sekutuBesar.class akan memanggil hitung.class untuk melakukan perhitungan faktor persekutuan besar dari dua buah bilangan.

6.7. Function yang Tidak Mengembalikan Nilai

Untuk sub rutin (function) yang tidak mengembalikan nilai bentuknya sangat mirip dengan function yang mengembalikan nilai. Perbedaannya adalah penggunaan kata kunci atau klausa void pada function yang tidak mengembalikan nilai baik pada bahasa C++ maupun bahasa Java. Untuk lebih jelasnya, perhatikan program C++ berikut ini untuk permasalahan yang sama, yaitu mencari faktor persekutuan besar.

```
1.  #include <iostream>
2.  using namespace std;
3.  void fpb(int a, int b) {
4.      int hasil;
5.      int r = a % b;
6.      if (r==0) hasil = b;
```

```
7.      else {
8.        while(r!=0) {
9.          a = b;
10.         b = r;
11.         r = a % b;
12.         hasil = b;
13.       }
14.   }
15.   cout << "FPB-nya = " << hasil <<endl;
16. }
17. void main() {
18.   int m,n;
19.   do {
20.     cout << "Bilangan pertama = ";
21.     cin >> m;
22.     cout << "Bilangan kedua   = ";
23.     cin >> n;
24.   } while (m < n);
25.   fpb(m,n);
26. }
```

Keluaran programnya :

```
Bilangan pertama      = 30
Bilangan kedua        = 18
FPB-nya               = 6
```

Program di atas, function fpb() (baris ke-3 sampai dengan baris ke-16) tidak mempunyai tipe data dan klausa return diakhir program, sehingga dapat kita simpulkan bahwa function fpb() bukan merupakan suatu function yang mengembalikan nilai. Sebaliknya, function fpb() diawali dengan klausa void sehingga function tersebut merupakan suatu function yang tidak mengembalikan nilai.

Hasil perhitungan faktor persekutuan besar dari dua buah bilangan tidak dikembalikan ke program utama yang memanggilnya tetapi ditampilkan sendiri oleh function tersebut (baris ke-15). Dengan argumen pada program utama yang memanggil fuction fpb() (baris ke-25) tidak menghasilkan suatu nilai seperti halnya pada contoh sebelumnya (function yang mengembalikan nilai). Untuk contoh function yang tidak mengembalikan nilai pada bahasa Java dengan permasalahan yang sama adalah sebagai berikut:

```
1. import java.util.Scanner;
2. import java.io.*;
```

```
3.  class hitung {
4.      public void fpb(int a, int b) {
5.          int hasil=0;
6.          int r = a % b;
7.          if (r==0) hasil = b;
8.          else {
9.              while(r!=0) {
10.               a = b;
11.               b = r;
12.               r = a % b;
13.               hasil = b;
14.             }
15.         }
16.         System.out.println("Bilangan terbesarnya  = " + hasil);
17.     }
18. }
19. class sekutuBesar {
20.     public static void main(String[ ] args) {
21.       hitung sekutu = new hitung();
22.       int m,n;
23.       Scanner input = new Scanner(System.in);
24.       do {
25.         System.out.print("Bilangan pertama = ");
26.         m = input.nextInt();
27.         System.out.print("Bilangan kedua   = ");
28.         n = input.nextInt();
29.       } while(m < n);
30.       sekutu.fpb(m,n);
31.     }
32.}
```

Keluaran programnya :

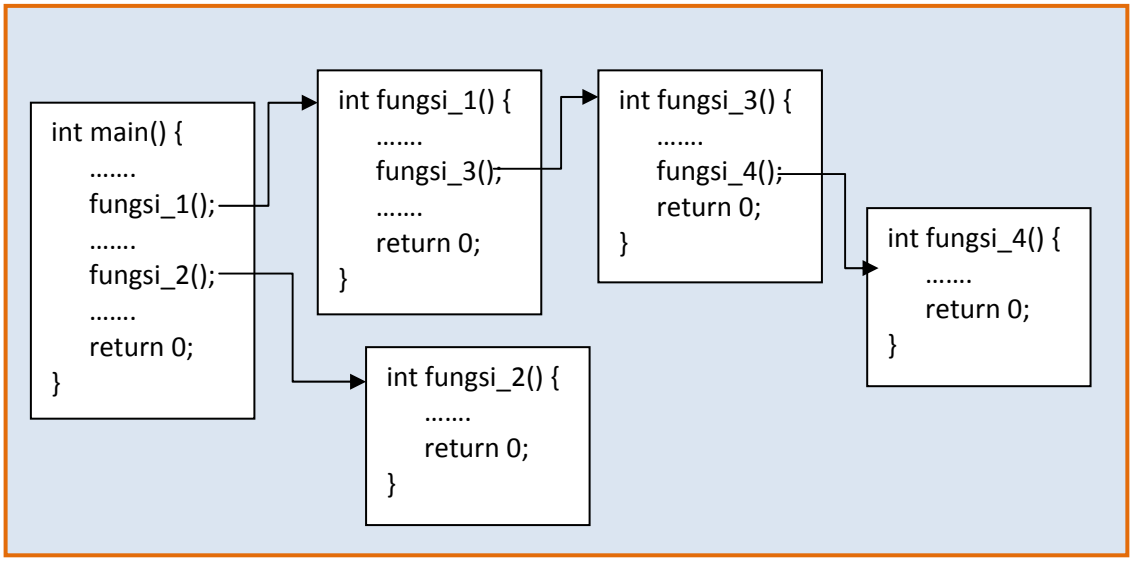
```
Bilangan pertama        = 44
Bilangan kedua          = 36
Bilangan terbesarnya    = 4
```

Sama seperti pada bahasa C++, class yang mempunyai function method untuk yang tidak mengembalikan nilai pada bahasa Java juga tidak mempunyai tipe data, tetapi diawali dengan klausa void (baris ke-4).

Hasil perhitungan faktor persekutuan besar oleh class hitung() juga tidak ditampilkan oleh class sekutuBesar() yang memanggilnya (baris ke-30), tetapi oleh class hitung() itu sendiri (baris ke-16).

6.8. Function Call Function

Sub rutin dalam suatu program tidak hanya dapat dipanggil oleh program utama saja tetapi antar sub rutin juga dapat saling memanggil.



Gambar 6.2. Sub rutin yang tidak hanya dipanggil oleh program utama

Berikut adalah contoh sub rutin yang memanggil sub rutin lainnya.

```
1.  #include <iostream>
2.  using namespace std;
3.  void fpb(int a, int b){
4.     int hasil;
5.     int r = a % b;
6.     if (r==0) hasil = b;
7.     else {
8.       while(r!=0) {
9.         a = b;
10.        b = r;
11.        r = a % b;
12.        hasil = b;
13.      }
14.   }
15.   cout << "FPB-nya = " << hasil <<endl;
16. }
17. void input_data(){
```

```
18.    int m,n;
19.    do {
20.      cout << "Bilangan pertama = ";
21.      cin >> m;
22.      cout << "Bilangan kedua   = ";
23.      cin >> n;
24.    } while (m < n);
25.    fpb(m,n);
26. }
27. void main() {
28.    input_data();
29. }
```

Keluaran programnya adalah :

```
Bilangan  pertama       = 56
Bilangan kedua          = 24
FPB-nya                 = 8
```

Program di atas mempunyai dua buah function, yaitu function fpb() dan function input_data(). Pertama kali function yang dipanggil oleh program utama adalah function input_data() (baris ke-28). Kemudian function input_data() melakukan pemanggilan function lain yaitu function fpb() (baris ke-25) setelah user memasukkan data untuk bilangan pertama dan bilangan kedua. Sedangkan kode program dalam bahasa Java untuk permasalahan yang sama adalah:

```
1.  import java.util.Scanner;
2.  import java.io.*;
3.  class hitung {
4.     public void fpb(int a, int b){
5.       int hasil=0;
6.       int r = a % b;
7.       if (r==0) hasil = b;
8.       else {
9.         while(r!=0) {
10.          a = b;
11.          b = r;
12.          r = a % b;
13.          hasil = b;
14.        }
15.    }
16.    System.out.println("Bilangan terbesarnya  = " + hasil);
17.  }
```

```
18. }
19. class input_data {
20.     public void data_input() {
21.       hitung sekutu = new hitung();
22.       int m,n;
23.       Scanner input = new Scanner(System.in);
24.       do {
25.         System.out.print("Bilangan pertama = ");
26.         m = input.nextInt();
27.         System.out.print("Bilangan kedua   = ");
28.         n = input.nextInt();
29.       } while(m < n);
30.       sekutu.fpb(m,n);
31.     }
32. }
33. class sekutuBesar {
34.     public static void main(String[ ] args) {
35.       input_data masukan = new input_data();
36.       masukan.data_input();
37.     }
38. }
```

Keluaran programnya adalah :

```
Bilangan  pertama       =76
Bilangan kedua          =18
Bilangan terbesarnya    = 2
```

6.9. Call by Value dan Call by References

Ada dua cara bagaimana suatu argumen dalam suatu program dapat memanggil sub rutin (function), yaitu call by value dan call by reference. Yang dimaksud dengan call by value adalah metode yang menyalin data (nilai) dari argumen yang memanggil function ke parameter dari function tersebut. Akibatnya jika ada perubahan nilai pada parameter function tidak akan berpengaruh pada nilai aslinya (nilai yang ada pada argumen program yang memanggil function tersebut).

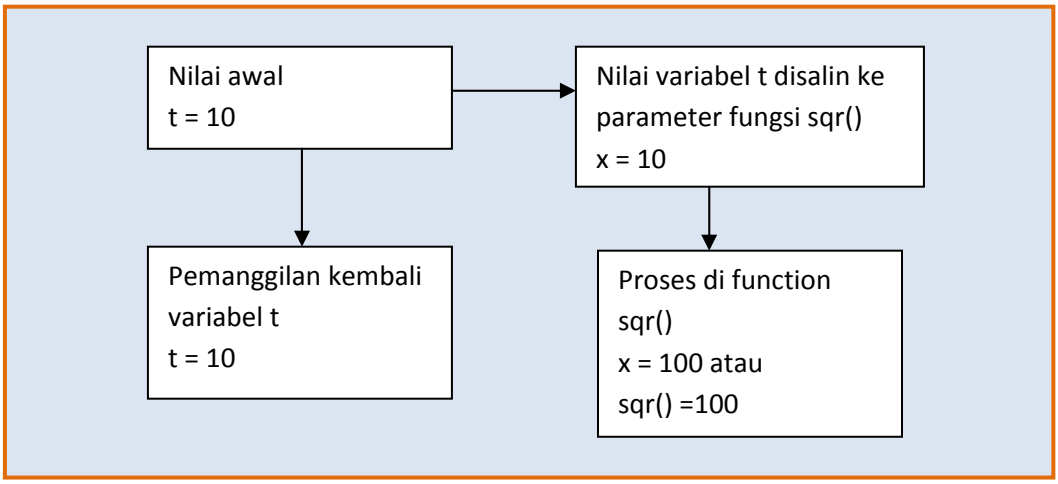
Sebaliknya untuk call by reference yang disalin bulan nilainya tetapi alamat memori yang menyimpan nilai tersebut sehingga jika terjadi perubahan-perubahan nilai pada parameter function, maka secara otomatis nilai argumennya juga akan ikut berubah. Untuk lebih jelasnya, perhatikan contoh call by value dengan C++ berikut ini:

```
1.  #include <iostream>
2.  using namespace std;
3.  int sqr(int x) {
4.      x = x*x;
5.      return(x);
6.  }
7.  int main(void) {
8.      int t=10;
9.      cout << sqr(t) << ", " << t <<endl;
10.     return 0;
11. }
```

Keluaran programnya adalah :

```
100, 10
```

Nilai argumen sqr() pada program utama yaitu t (bernilai 10) disalin ke parameter x function sqr(). Didalam function sqr(), nilai x dirubah (x=x*x) sehingga function sqr() bernilai 100. Nilai function sqr() ini langsung ditampilkan oleh program yang memanggilnya (baris ke-9). Akan tetapi nilai t, yang juga ditampilkan oleh program utama (bariske-9), tetap 10. Keluaran program ini adalah 100, 10 dimana 100 adalah nilai dari function fpb() dan 10 adalah nilai variabel t.



Gambar 6.3. call by value

Untuk permasalahan yang sama dalam bahasa Java adalah sebagai berikut :

```
1.  import java.io.*;
2.  class kuadrat {
```

```
3.      public int sqr(int x) {
4.        x = x*x;
5.        return(x);
6.      }
7.   }
8.   class power {
9.      public static void main(String[ ] args) {
10.       kuadrat a = new kuadrat();
11.       int t=10;
12.       System.out.println(a.sqr(t) + ", " + t);
13.    }
14. }
```

Keluaran programnya adalah :

```
100, 10
```

Untuk call by reference, tipe data yang digunakan adalah tipe data pointer karena yang disalin adalah alamat dari memori dimana data disimpan (pembahasan mengenai pointer ini ada di bab tersendiri). Untuk bahasa Java tidak menggunakan call by reference karena tidak ada pointer dalam bahasa Java. Contoh call by reference (dengan menggunakan bahasa C++) adalah sebagai berikut:

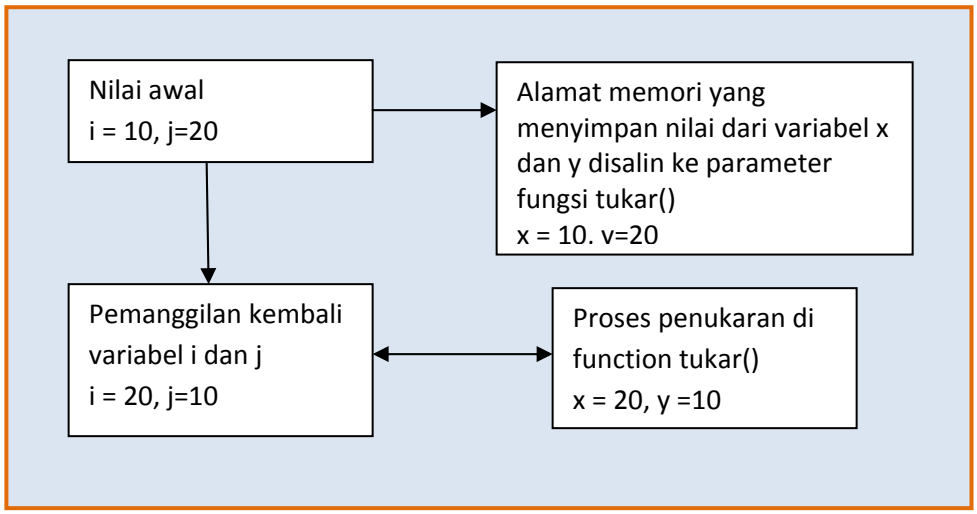
```
1.  #include <iostream>
2.  using namespace std;
3.  void tukar(int *x, int *y) {
4.     int temp;
5.     temp = *x;
6.     *x = *y;
7.     *y = temp;
8.  }
9.  int main(void) {
10.    int i, j;
11.    i = 10;
12.    j = 20;
13.    cout << "Mula-mula : " << endl;
14.    cout << "Nilai i : " << i << endl;
15.    cout << "Nilai j : " << j << endl;
16.    tukar(&i, &j);
17.    cout << endl << "Setelah ditukar : "          << endl;
18.    cout << "Nilai i : " << i << endl;
19.    cout << "Nilai j : " << j << endl;
20.    return 0;
21. }
```

Tanda * pada variabel di function tukar() merupakan variabel pointer.

Keluaran dari program di atas adalah :

```
Mula-mula :
Nilai I    : 10
Nilai j    : 20

Setelah ditukar :
Nilai I    : 20
Nilai j    : 10
```

Baris ke-14 dan 15, digunakan untuk menampilkan nilai dari variabel i dan j yang nilainya masing-masing 10 dan 20. Setelah dilakukan penukaran dengan memanggil function tukar() pada baris ke-16, program kemudian memanggil kembali variabel i dan j (baris ke-18 dan ke-19). Ternyata variabel i dan j telah tertukar. Hal ini terjadi karena pada function tukar() menyalin alamat memori yang menyimpan variabel i dan j dan kemudian menukar isinya sehingga jika variabel i dan j jika dipanggil nilainya akan berubah karena isinya telah ditukar.



Gambar 6.4. Call by reference

6.10. Parameter dengan Nilai Default

Nilai parameter dalam suatu function C++ dapat kita beri nilai default, yaitu suatu nilai yang diberikan secara default oleh function jika parameter dari argumen yang memanggil function tersebut tidak

kita berikan. Untuk lebih jelasnya perhatikan contoh berikut ini:

```
1.  #include <iostream>
2.  using namespace std;
3.  double luas_segi3(int alas, int tinggi=10){
4.     return(0.5 * alas * tinggi);
5.  }
6.  void main() {
7.     int a=15,t=20;
8.     cout << "Luas segitiga = ";
9.     cout << luas_segi3(a,t) <<endl;
10.    cout << endl;
11.    cout << "Luas segitiga = ";
12.    cout << luas_segi3(a) <<endl;
13. }
```

Keluaran programnya adalah :

```
Luas segitiga     = 150
Luas segitiga     = 75
```

Program di atas mempunyai satu buah function yaitu function luas_segi3() dimana function tersebut mempunyai dua buah parameter formal yaitu alas dan tinggi. Parameter alas tidak mempunyai nilai default, tetapi untuk parameter tinggi mempunyai nilai default, yaitu 10.

Program utama dari program tersebut telah menentukan bahwa nilai dari variabel a yang nantinya digunakan sebagai variabel masukan untuk parameter alas dari function luas_segi3() adalah 15. Sedangkan variabel t yang nantinya digunakan sebagai variabel masukan untuk parameter tinggi dari function luas_segi3() adalah 20. Sehingga ketika argumen program utama yang memanggil function luas_segi3() pada baris ke-9 dijalankan, maka nilai dari parameter alas dan tinggi pada function luas_segi3() masing-masing adalah 15 dan 20. Dengan demikian function luas_segi3() akan bernilai 150.

Akan tetapi untuk argumen baris ke-12 pada program utama terlihat bahwa argumen tersebut tidak menyertakan variabel t, tetapi hanya menyertakan variabel a. Sehingga ketika argumen tersebut dijalankan maka hanya parameter alas pada function luas_segi3() saja yang menerima masukan dari argumen tersebut. Sedangkan untuk parameter tinggi tidak memperoleh masukan. Akan tetapi karena parameter tinggi mempunyai nilai default 10, maka nilai variabel tinggi yang digunakan untuk menghitung luas segitiga adalah 10. Dengan demikian nilai dari function luas_segi3() adalah 75.

6.11. Overloading

Overloading adalah function-function yang ada dalam suatu program dimana function-function tersebut mempunyai nama yang sama tetapi parameter formal-nya berbeda-beda antara yang satu dengan yang lainnya. Ada tiga jenis overloading, yaitu:

1. Function tersebut mempunyai jumlah parameter formal yang berbeda tetapi tipe data-nya sama.
2. Function tersebut mempunyai jumlah parameter formal yang sama tetapi tipe data yang berbeda.
3. Function-function tersebut mempunyai jumlah parameter formal yang berbeda dan tipe data dari parameter formal tersebut juga berbeda.

Berikut ini diberikan contoh untuk function-function yang mempunyai jumlah parameter formal yang berbeda tetapi tipe data-nya sama.

Untuk bahasa C++

```
1.  #include <iostream>
2.  using namespace std;
3.  void luas_segi3(int alas){
4.     int tinggi=10;
5.     cout << "Luas segitiga 1 = "
6.     cout << (0.5 * alas * tinggi) << endl;
7.  }
8.  void luas_segi3(int alas,int tinggi){
9.     cout << "Luas segitiga 2 = "
10.    cout << (0.5 * alas * tinggi) << endl;
11. }
12. void luas_segi3(int alas, int tinggi,int bagi){
13.    cout << "Luas segitiga 2 bagi "
14.    cout << bagi << " = "
15.    cout << (0.5 * alas * tinggi)/bagi
16.    cout << endl;
17. }
18. void main() {
19.    luas_segi3(10);
20.    luas_segi3(10,15);
21.    luas_segi3(10,15,3);
22. }
```

Keluaran programnya adalah :

```
Luas segitiga 1          = 150
Luas segitiga 2          = 75
Luas segitiga 2 bagi  3  = 25
```

Program di atas mempunyai tiga buah function yang mempunyai nama sama yaitu luas_segi3(). Function yang pertama (baris ke-3) hanya mempunyai satu buah parameter formal yaitu alas dengan tipe data integer. Sedangkan function yang kedua (baris ke-8) mempunyai dua buah parameter formal alas dan tinggi dimana keduanya bertipe data integer. Function yang terakhir (baris ke-12) mempunyai tiga buah parameter formal yaitu alas, tinggi, dan bagi yang semuanya bertipe data yang sama, yaitu integer.

Pada program utama, terdapat tiga buah argumen dimana argumen pertama (baris ke-19) hanya mempunyai satu variabel masukan untuk function, argumen kedua (baris ke-20) mempunyai dua buah variabel masukan, dan argumen terakhir (baris ke-21) mempunyai tiga buah variabel masukan.

C++ secara otomatis akan mengarahkan argumen pertama kepada function yang pertama, argumen kedua kepada function yang kedua, dan argumen ketiga kepada function yang ketiga, sesuai dengan variabel masukan yang dipunyai oleh masing-masing argumen. Sedangkan untuk bahasa Java:

```
1.  class hitung {
2.      public void luas_segi3(int alas){
3.          int tinggi=10;
4.          double luas=0.5*alas*tinggi;
5.          System.out.println("Luas segitiga 1 = " + luas);
6.      }
7.      public void luas_segi3(int alas,int tinggi){
8.          double luas=0.5*alas*tinggi;
9.          System.out.println("Luas segitiga 2 = " + luas);
10.     }
11.     public void luas_segi3(int alas,      int tinggi,int bagi){
12.         double luas=(0.5*alas*tinggi)/bagi;
13.         System.out.println("Luas segitiga 2 dibagi " + bagi + " = " + luas);
14.     }
15. }
16. class overload1 {
17.     public static void main(String[ ] args) {
18.         hitung sekutu = new hitung();
19.         sekutu.luas_segi3(10);
20.         sekutu.luas_segi3(10,15);
21.         sekutu.luas_segi3(10,15,3);
22.     }
23. }
```

Keluaran programnya adalah :

```
Luas segitiga 1          = 150
Luas segitiga 2          = 75
Luas segitiga 2 bagi  3  = 25.0
```

Sedangkan contoh untuk function-function yang mempunyai jumlah parameter formal yang sama tetapi tipe data-nya berbeda adalah seperti berikut. Untuk bahasa C++:

```
1.  #include <iostream>
2.  using namespace std;
3.  void luas_segi3(int alas){
4.     int tinggi=10;
5.     cout << "Luas segitiga 1 = " << (0.5 * alas * tinggi) << endl;
6.  }
7.  void luas_segi3(char* alas){
8.     cout << alas << endl;
9.  }
10. void main() {
11.    luas_segi3(10);
12.    luas_segi3("Belajar pemrograman");
13. }
```

Keluaran programnya adalah :

```
Luas segitiga 1 = 150
Belajar pemrograman
```

Program di atas mempunyai dua buah function yaitu luas_segi3() dimana setiap function mempunyai satu buah parameter formal tetapi tipe datanya berbeda. Function yang pertama parameter formalnya mempunyai tipe data integer, dan function kedua mempunyai parameter formal dengan tipe data pointer char.

Sama seperti sebelumnya, C++ juga secara otomatis akan mengarahkan argumen yang memanggil function pada program tersebut kepada function yang berkesesuaian. Argumen pertama (baris ke-11) akan diarahkan kepada function pertama pula (baris ke-3) karena tipe data dari variabel masukan sama dengan parameter formal dari function yang pertama, yaitu integer.

Sedangkan untuk argumen kedua (baris ke-12) akan diarahkan kepada function kedua (baris ke-7) karena tipe data dari variabel masukan sama dengan parameter formal dari function yang kedua, yaitu char. Untuk bahasa Java dengan permasalahan yang sama adalah sebagai berikut:

```
1.  class hitung {
2.     public void luas_segi3(int alas){
3.       int tinggi=10;
4.       double luas=0.5*alas*tinggi;
5.       System.out.println("Luas segitiga 1 = " + luas);
6.     }
7.     public void luas_segi3(String alas){
8.       System.out.println(alas);
9.     }
10. }
11. class overload2 {
12.    public static void main(String[ ] args) {
13.      hitung sekutu = new hitung();
14.      sekutu.luas_segi3(10);
15.      sekutu.luas_segi3("Belajar pemrograman");
16.    }
17. }
```

Keluaran programnya adalah :

```
Luas segitiga 1 = 50.0
Belajar pemrograman
```

Contoh untuk function-function yang mempunyai jumlah parameter formal yang berbeda dan tipe data yang berbeda pula adalah seperti berikut:

Untuk bahasa C++ :

```
1.  #include <iostream>
2.  using namespace std;
3.  void luas_segi3(int alas){
4.     int tinggi=10;
5.     cout << "Luas segitiga 1 = "
6.     cout << (0.5 * alas * tinggi) << endl;
7.  }
8.  void luas_segi3(char* alas){
9.     cout << alas << endl;
10. }
11. void luas_segi3(char* alas,int data){
12.    cout << alas << " : " << endl;
13.    cout << "Pangkat dua dari "
14.    cout << data << " adalah "
15.    cout << (data*data) << endl;
16. }
17. void main() {
18.    luas_segi3(10);
```

```
19.    luas_segi3("Belajar pemrograman");
20.    luas_segi3("Belajar pemrograman lagi",3);
21. }
```

Keluaran programnya adalah :

```
Luas segitiga 1   = 150
Belajar pemrograman
Belajar pemrograman lagi :
Pangkat dua dari 3 adalah 9
```

Program di atas mempunyai tiga buah function yang mempunyai nama yang sama, yaitu luas_segi3(). Masing-masing function mempunyai jumlah parameter formal yang berbeda dan tipe data dari parameter formal tersebut juga berbeda.

Sama dengan sebelumnya, C++ secara otomatis juga akan mengarahkan argumen yang memanggil function-function tersebut sesuai dengan jumlah variabel masukan dan tipe datanya. Untuk bahasa Java dengan permasalahan yang sama adalah sebagai berikut:

```
1.  class hitung {
2.      public void luas_segi3(int alas){
3.          int tinggi=10;
4.          double luas=0.5*alas*tinggi;
5.          System.out.println("Luas segitiga 1 = " + luas);
6.      }
7.      public void luas_segi3(String alas){
8.          System.out.println(alas);
9.      }
10.     public void luas_segi3(String alas,int data){
11.         System.out.println(alas + " : ");
12.         System.out.println("Pangkat dua dari " + data + " adalah " +(data*data));
13.     }
14. }
15. class overload3 {
16.     public static void main(String[ ] args) {
17.         hitung sekutu = new hitung();
18.         sekutu.luas_segi3(10);
19.         sekutu.luas_segi3("Belajar      pemrograman");
20.         sekutu.luas_segi3("Belajar      pemrograman",3);
21.     }
22. }
```

Keluaran programnya adalah :

```
Luas segitiga 1 = 50.0
Belajar pemrograman
Belajar Pemrograman :
Pangkat dua dari 3 adalah 9
```

6.12. Soal Latihan

Jawablah soal latihan dibawah ini dengan baik dan benar.

1. Apa yang dimaksud dengan prosedur dan
2. Sebutkan perbedaan prosedur dengan fungsi
3. Apa sajakah parameter-parameter yang dimiliki oleh prosedur
4. Mengapa dalam program perlu ditulis dalam bentuk subrutin
5. Apa yang dimaksud dengan funtion call function
6. Apa yang dimaksud dengan call by value dan call by reference
7. Apa yang dimaksud dengan overloading

BAB 7
FUNGSI

7.1. Pendahuluan

Fungsi adalah kumpulan pernyataan yang melakukan tugas tertentu. Sejauh ini Anda telah menggunakan fungsi dalam dua cara: (1) Anda telah membuat sebuah fungsi bernama utama dalam setiap program yang telah ditulis, dan (2) Anda memiliki fungsi library disebut seperti sqrt dan pow. Dalam bab ini akan mempelajari cara membuat fungsi yang dapat digunakan seperti fungsi library pada C++.

Salah satu alasan mengapa menggunakan fungsi adalah untuk memecah program ke dalam sebuah program yang lebih kecil sehingga mudah dikelola. Setiap unit modul, diprogram sebagai fungsi terpisah. Misalnya pada sebuah buku yang memiliki seribu halaman, tetapi tidak dibagi ke dalam bab atau bagian. Jika ingin mencoba untuk menemukan satu topik dalam buku ini akan sangat sulit. Real-world program dapat dengan mudah ada ribuan baris kode, dan kecuali mereka modularized, mereka bisa jadi sangat sulit untuk mengubah dan memelihara.

Alasan lain untuk menggunakan fungsi adalah untuk bahwa fungsi menyederhanakan program. Jika tugas tertentu dilakukan di beberapa tempat di sebuah program, sebuah fungsi dapat ditulis sekali saja untuk melakukan tugas itu, dan kemudian

akan dijalankan kapan saja dibutuhkan.

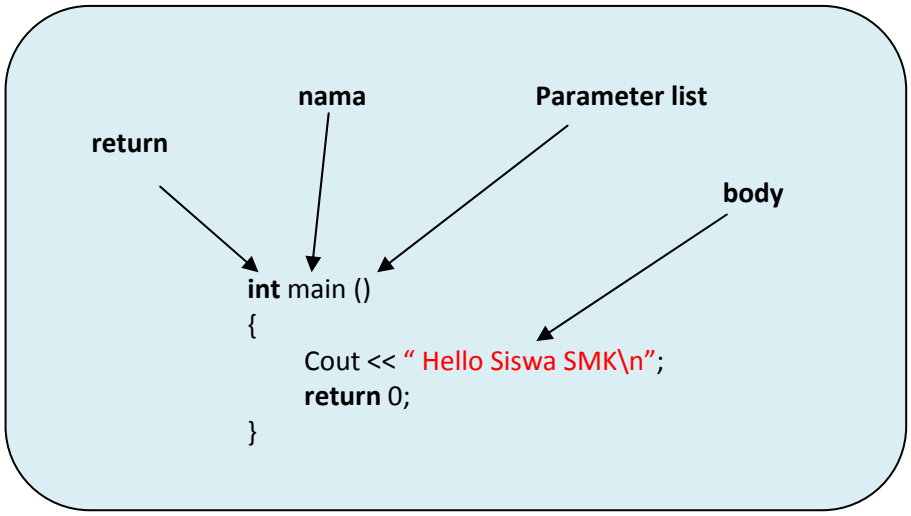
Ketika membuat sebuah fungsi, yang harus ditulis adalah definisi. Semua definisi mempunyai bagian-bagian dibawah ini:

Name
Setiap fungsi harus memiliki nama. Secara umum, peraturan yang sama berlaku untuk nama variabel juga berlaku untuk nama fungsi.

Parameter list
Program modul fungsi panggilan yang dapat mengirim data ke. Daftar parameter adalah daftar variabel yang memegang nilai-nilai yang disampaikan ke fungsi.

Body
Badan fungsi adalah serangkaian pernyataan yang melaksanakan tugas melakukan fungsi. Pernyataan ini akan ditutupi dalam satu set braces.

Return type:
fungsi dapat mengirim kembali ke nilai program modul yang memanggilnya. Return type merupakan tipe data dari nilai yang dikirim kembali.
Gambar dibawah menunjukkan definisi dari fungsi yang sederhana dengan berbagai bagian berlabel. Perhatikan bahwa fungsi dari jenis kembali sebenarnya tercantum pertama.



Gambar 7.1. function header

Contoh program diatas merupakan baris pendefinisian int main () yang disebut dengan *function header*.

7.2. Fungsi Void

Yang sudah tahu pada fungsi adalah bahwa fungsi dapat mengembalikan nilai. Fungsi utama dalam semua program yang telah dilihat dalam buku ini dinyatakan untuk kembali ke nilai int sebuah sistem operasi. Return 0; merupakan sebuah pernyataan yang menyebabkan nilai 0 bila fungsi utama telah selesai melaksanakan tugasnya. Hal tersebut tidak semua fungsi diperlukan untuk kembali ke nilai tersebut. Beberapa fungsi hanya melakukan satu atau lebih pernyataan dan kemudian mengakhiri. Ini disebut void fungsi. Fungsi TampilPesan seperti ditampilkan di bawah ini adalah sebagai contoh:

```
void TampilPesan()
{
    cout << "Hello from the function TampilPesan.\n";
}
```

Nama fungsi tersebut diatas TampilPesan. Nama ini jelas, sebagai nama fungsi harus. Ia memberi indikasi tentang apa yang tidak berfungsi: ini menampilkan pesan. Perhatikan kembali fungsi dari jenis void. Ini berarti fungsi tidak mengembalikan nilai ke bagian program yang dijalankan tersebut. Pemberitahuan juga tidak memiliki fungsi pernyataan Kembali. Ini hanya menampilkan pesan pada layar dan keluar.

7.3. Pemanggilan Fungsi

fungsi dijalankan ketika dipanggil. Fungsi utama akan dipanggil secara otomatis saat program dimulai, namun semua fungsi harus dijalankan oleh pernyataan *function call*. Ketika sebuah fungsi dipanggil, program untuk melakukan percabangan menuju dan melaksanakan semua pernyataan yang ada didalam tubuh fungsi tersebut. Untuk lebih jelasnya mari kita lihat Program dibawah, yang berisi dua fungsi yaitu: TampilPesan dan utama.

Program 7.1.

```
#include <conio.h>
#include <iostream>

using namespace std;

void TampilPesan()
{
cout << " >> Salam Hello dari fungsi TampilPesan.\n";
```

```
}

int main()
{

   cout << "Hello dari program Utama.\n";
   cout << "------------------------\n";
   TampilPesan();                    // memanggil TampilPesan
   cout << "------------------------\n";
   cout << "kembli ke Fungsi Utama lagi.\n";
   getch();
   return 0;
}
```

Keluaran programnya adalah:

```
Hello dari program Utama.
------------------------
 >> Salam Hello dari fungsi TampilPesan.
------------------------
kembli ke Fungsi Utama lagi.
```

Fungsi TampilPesan disebut oleh baris utama adalah: TampilPesan(); Baris ini digunakan untuk fungsi panggilan. Dimana hal ini merupakan nama fungsi yang diikuti oleh tanda kurung dan titik koma. Coba kita bandingkan dengan fungsi header seperti berikut:

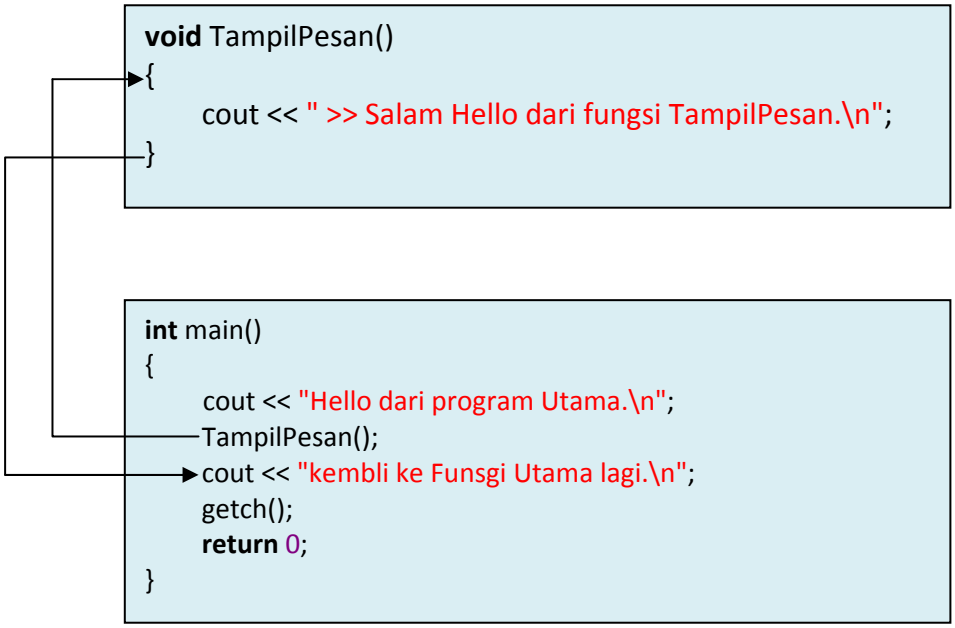
```
Function Header   →   void  TampilPesan()
Function Call     →   TampilPesan();
```

Fungsi header adalah bagian dari definisi fungsi. Ia menyatakan kembali sebuah fungsi dari jenis, nama, dan daftar parameter. Hal ini tidak diakhiri dengan titik koma karena definisi dari tubuh fungsi yang mengikutinya. Fungsi panggilan adalah pernyataan yang melaksanakan fungsi tersebut, sehingga diakhiri dengan titik koma seperti pernyataan C++ lainnya. Fungsi panggilan tidak ada dalam daftar, dan jika program ini tidak ke dalam fungsi, tanda kurung yang akan dikosongkan.

Walaupun program mulai dijalankan pada program utama (main), fungsi TampilPesan didefinisikan terlebih dahulu. Hal ini karena compiler harus mengetahui fungsi dari jenis return, jumlah parameter, dan setiap jenis parameter sebelumnya yang dipanggil. Salah satu cara agar compiler mengetahui informasi ini adalah dengan menempatkan fungsi

definisi sebelum semua panggilan ke fungsi.

Perhatikan bagaimana program diatas mengalir. Dalam memulai, tentu saja, didalam fungsi utama. Bila panggilan ke TampilPesan yang dihadapi, program untuk cabang yang berfungsi dan melakukan pernyataan-nya. Setelah selesai melaksanakan TampilPesan, program cabang kembali ke fungsi utama dan kembali dengan baris yang mengikuti fungsi panggilan. Hal ini diilustrasikan pada gambar dibawah ini.



Gambar 7.2. Ilustrasi Fungsi

Dalam menyatakan pemanggilan Fungsi dapat menggunakan struktur kendali seperti loop, pernyataan IF, dan pernyataan switch. Program dibawah merupakan program yang melakukan pemanggilan fungsi TampiPesan kedalam satu looping

Program 7.2.

```
#include <conio.h>
#include <iostream>

using namespace std;

void TampilPesan()
{
```

```
cout << " >> Salam Hello dari fungsi TampilPesan.\n";
}

int main()
{
        cout << "Hello dari program Utama.\n";
        cout << "------------------------\n";
        for (int count = 0; count < 5; count++)
        TampilPesan();                          // memanggil Fungsi TampilPesan
        cout << "------------------------\n";
        cout << "kembli ke Funsgi Utama lagi.\n";
        getch();
        return 0;
}
```

Keluaran Programnya adalah:

```
Hello dari program Utama.
------------------------
 >> Salam Hello dari fungsi TampilPesan.
 >> Salam Hello dari fungsi TampilPesan.
 >> Salam Hello dari fungsi TampilPesan.
 >> Salam Hello dari fungsi TampilPesan.
 >> Salam Hello dari fungsi TampilPesan.
------------------------
kembli ke Funsgi Utama lagi.
```

Setiap program dapat dimungkinkan untuk memiliki banyak fungsi atau sebuah fungsi dalam sebuah program. Program dibawah 7.3. dibawah ini memiliki tiga fungsi: yaitu utama, fungsi pertama dan fungsi kedua. Untuk lebih jelas perhatikan program dibawah ini:

Program 7.3

```
#include <conio.h>
#include <iostream>

using namespace std;

void pertama()
{
   cout << " >>Saya sekarang sedang di Fungsi Pertama.\n";
}

void kedua()
```

```
{
    cout << " >> Saya sekarang sedang di Fungsi Kedua.\n";
}

int main()
{
   cout << "Saya sedang di program Utama.\n";
   pertama();                    // pemanggilan fungsi pertama
   kedua();                      // pemanggilan fungsi kedua
   cout << "Saya kembali ke Program Utama.\n";
   getch();
   return 0;
}
```

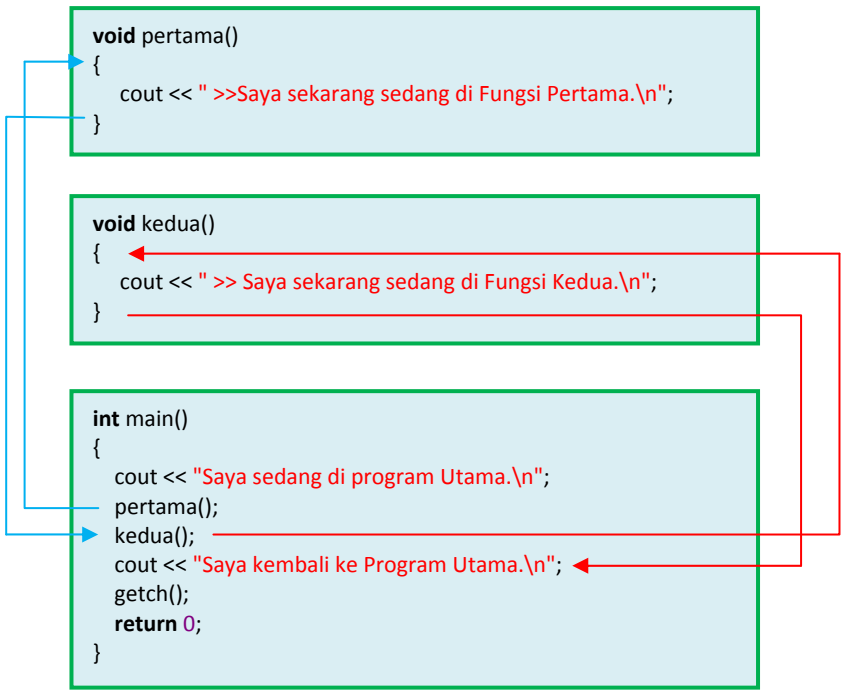
Keluaran programnya adalah sebagai berikut:

```
Saya sedang di program Utama.
 >> Saya sekarang sedang di Fungsi Pertama.
 >> Saya sekarang sedang di Fungsi Kedua.
Saya kembali ke Program Utama.
```

Dalam program diatas fungsi utama terdiri dari pemanggilan fungsi pertama dan kedua dengan instruksi sebagai berikut:

```
pertama();
kedua();
```

Setiap panggilan pernyataan menyebabkan program untuk cabang ke salah satu fungsi dan kemudian kembali ke utama bila fungsi selesai. Gambar dibawah menggambarkan jalan yang diambil oleh program



Gambar 7.3. Pemanggilan lebih dari satu fungsi

Sebuah fungsi sering juga dipanggil dalam dalam sebuah hirarki, atau berlapis. Program dibawah ini merupakan sebuah program yang memiliki tiga buah fungsi dimana fungsi tersebut adalah utama, Agakdalam dan PalingDalam. Kerja program tersebut adalah fungsi palingdalam dipanggil oleh fungsi agakdalam dan kemudian fungsi AgakDalam dipanggil oleh fungsi Utama. Untuk lebih jelasnya perhatikan program dibawah ini:

Program 7.4.

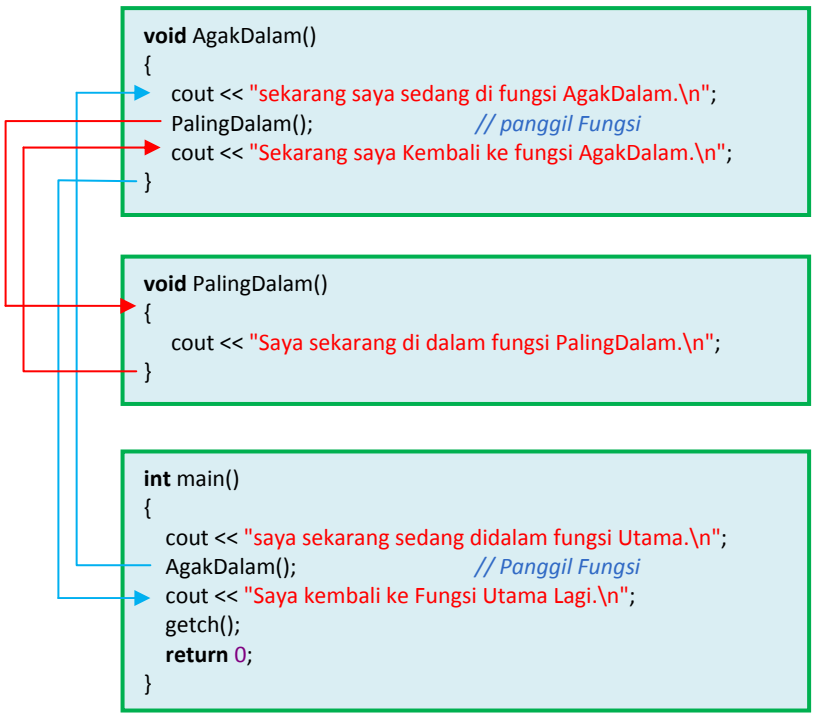
```
#include <conio.h>
#include <iostream>

using namespace std;

void PalingDalam()
{
cout << "Saya sekarang sedang di dalam fungsi PalingDalam.\n";
}

void AgakDalam()
```

```
{
    cout << "sekarang saya sedang di fungsi AgakDalam.\n";
    PalingDalam();                    // panggil Fungsi
    cout << "Sekarang saya Kembali ke fungsi AgakDalam.\n";
}

int main()
{
   cout << "saya sekarang sedang didalam fungsi Utama.\n";
   AgakDalam();                       // Panggil Fungsi
   cout << "Saya kembali ke Fungsi Utama Lagi.\n";
   getch();
   return 0;
}
```

Keluaran Program adalah sebagai berikut:

```
saya sekarang sedang didalam fungsi Utama.
sekarang saya sedang di fungsi AgakDalam.
Saya sekarang sedang di dalam fungsi PalingDalam.
Sekarang saya Kembali ke fungsi AgakDalam.
Saya kembali ke Fungsi Utama Lagi.
```

Dalam Program 7.4 diatasfungsi utama hanya memanggil fungsi agakDalam. Sedangkan fungsi AgakDalam memanggil Fungsi PalingDalam. Program diatas adalah memanggil fungsi dan kemudian fungsi yang dipanggil tersebut memnggil fungsi yang lain. Untuk lebih jelasnya perhatikan gambar dibawah ini:



Gambar 7.4. Pemanggilan fungsi didalam fungsi

7.4. Prototipe Fungsi

Sebelum compiler menemukan panggilan ke fungsi tertentu, kita harus sudah mengetahui beberapa hal mengenai fungsi. Secara khusus, kita harus mengetahui jumlah parameter yang digunakan oleh fungsi, jenis parameter, dan cara atau jenis kembalinya fungsi. Sebuah parameter membolehkan informasi untuk dikirim ke salah satu fungsi.

Untuk saat ini, fungsi yang akan digunakan berupa fugsi yang tidak memiliki parameter dan, kecuali pada fungsi utama, dan cara kembalinya menggunakan jenis void. Salah satu cara untuk memastikan bahwa compiler ini memiliki informasi yang diperlukan adalah dengan menempatkan fungsi yang telah didefinisikan sebelum melakukan panggilan ke fungsi tersebut. Ini merupakan pendekatan yang diambil seperti halnya dalam program 7.1 sampai program 7.4 diatas. Cara lainnya adalah adalah dengan menyatakan fungsi dengan fungsi prototipe (a *function prototype*). Berikut ini adalah prototipe untuk fungsi TampilPesan seperti dalam Program 7.1 adalah sebagai berikut:

```
void TampilPesan();
```

Prototipe tersebut diatas tampak mirip dengan fungsi header, kecuali ada titik koma dibagian akhir. Sebuah pernyataan fungsi akan

memberitahukan ke compiler bahwa fungsi TampilPesan memiliki tipe kembali (return) void, berarti fungsi tersebut tidak mengembalikan nilai serta tidak menggunakan parameter.

Prototipe fungsi biasanya ditempatkan pada bagian atas pada program sehingga compiler akan melakukan eksekusi terlebih dahulu sebelum fungsi tersebut dipanggil.

Program 7.5 dibawah ini adalah modifikasi dari program 7.3 diatas. Definisi fungsi pertama dan kedua telah ditempatkan setelah program utama, dan fungsi prototip telah ditempatkan diatas program utama secara langsung setelah pernyataan **namespace std**. Perhatikan program dibawah ini:

Program 7.5.

```
#include <conio.h>
#include <iostream>

using namespace std;

//Prototipe Fungsi
void pertama();
void kedua();

int main()
{
   cout << "Saya sedang di program utama.\n";
   pertama();                    // memanggil fungsi pertama
   kedua();                      // memanggil fungsi kedua
   cout << "Kembali dalam program utama lagi.\n";
   getch();
   return 0;
}

void pertama()
{
   cout << "saya sekarang didalam fungsi pertama.\n";
}

void kedua()
{
   cout << "saya sekarang didalam fungsi kedua.\n";
}
```

Keluaran Program adalah sebagai berikut:

```
Saya sedang di program utama.
```

```
saya sekarang didalam fungsi pertama.
saya sekarang didalam fungsi kedua.
Kembali dalam program utama lagi.
```

Ketika compiler sedang membaca program utama diatas, kemudian program utama tersebut melakukan panggilan ke fungsi pertama dan kedua sebelum compiler membaca definisi dari fungsi tersebut. Karena hal tersebut sebagai fungsi prototip, compiler sudah mengetahui jenis parameter kembali pada informasi fungsi pertama dan kedua. Sehingga harus ada prototipe pada setiap fungsinya dalam sebuah program kecuali program utama. Sebuah prototipe program ini tidak pernah diperlukan oleh program utama karena telah dilakukan pada awal program.

7.5. Pengiriman data pada Fungsi

Nilai-nilai yang akan dikirim pada sebuah fungsi disebut dengan argument (*arguments*). Programmer yang ahli biasanya sudah akrab dengan cara menggunakan argumen dalam pemanggilan fungsi. Pernyataan berikut ini merupakan fungsi pow dengan dua argumen yang sedang melakukan panggilan 2 dan 4.

```
result = pow(2, 4);
```

Sebuah parameter merupakan variabel khusus yang menangani nilai yang dilewatkan sebagai argumen menuju sebuah fungsi. Dengan menggunakan parameter, kita dapat merancang sendiri fungsi yang dapat menerima data. Program berikut merupakan definisi sebuah fungsi yang menggunakan parameter:

```
void TampilNilai(int num)
{
        cout << "Nilainya adalah " << num << endl;
}
```

Perhatikan integer num yng digunakan untuk mendefinisikan variabel yang berada dalam kurung (int num). Variabel num adalah parameter. Integer Ini digunakan untuk membuat fungsi TampilNilai supaya menerima nilai integer sebagai argumen. Program 7.6 dibawah merupakan contoh yang menggunakan fungsi TampilNilai. Perhatikan program dibawah ini:

Program 7.6.

```
#include <conio.h>
#include <iostream>

using namespace std;

//Prototipe Fungsi
void TampilNilai(int);

int main()
{
   cout << "Saya sedang memasukan 5 ke fungsi TampilNilai.\n";
   TampilNilai(5);              // Call TampilNilai dengan argument 5
   cout << "Sekarang saya sudah kembali ke program utama.\n";
   getch();
   return 0;
}

void TampilNilai(int num)
{
    cout << "Besar nilainya adalah: " << num << endl;
}
```

Keluaran programnya adalah sebagai berikut:

```
Saya sedang memasukan 5 ke fungsi TampilNilai.
Besar nilainya adalah: 5
Sekarang saya sudah kembali ke program utama.
```

Dalam prototype fungsi tersebut diatas yang perlu diperhatikan adalah pada TampilNilai:

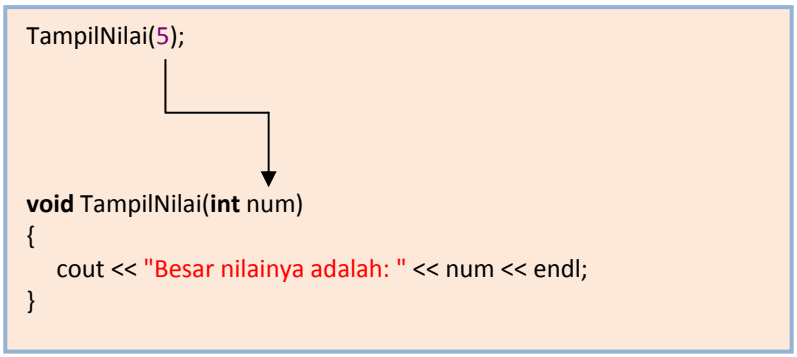
```
void TampilNilai(int);     // function prototype
```

Dalam fungsi tersebut kita tidak perlu memasukan daftar parameter yang merupakan variabel dalam tanda kurung, dan hanya dengan memasukan type data yang diperlukan saja. Fungsi ini dapat prototype ini juga dapat ditulis sebagai berikut:

```
void TampilNilai(int num);
```

Fungsi tersebut diatas sangat mudah digunakan walaupun, compiler mengabaikan nama variabel pada parameter fungsi prototype tersebut. Dalam program utama, fungsi TampiNilai disebut dengan argumen 5, dimana hal tersebut berada dalam tanda kurung. Nomor 5

dimasukan kedalam variabel num, yang mana num merupakan parameter TampilNilai. Untuk lebih jelasnya mengenai hal tersebut, perhatikan gambar dibawah ini:



Gambar 7.5. Fungsi prototipe

Dengan melihat gambar tersebut, apapun argumen yang tertulis dalam tanda kurung pada panggilan fungsi akan dicopy kedalam variable parameter fungsi. Pada dasarnya, parameter adalah variabel yang diinisialisasi menjadi nilai sesuai argument yang berhubungan. Program 7.7 dibawah menunjukkan bahwa fungsi \tampilNilai dapat dipanggil beberapa kali dengan alasan sesuai masing-masing kegunaannya.

Program 7.7.

```
#include <conio.h>
#include <iostream>

using namespace std;

//Prototipe Fungsi
void TampilNilai(int);

int main()
{
   cout << "Saya sedang mengisikan beberapa nilai pada TampilNilai.\n";
   TampilNilai(5);     // Panggil TampilNilai dengan argument 5
   TampilNilai(10);    // Panggil TampilNilai dengan argument 10
   TampilNilai(2);     // Panggil TampilNilai dengan argument 2
   TampilNilai(16);    // Panggil TampilNilai dengan argument 16
   cout << "sekarang saya kembali.\n";
```

```
   getch();
return 0;
}

void TampilNilai(int num)
{
    cout << "Nilainya adalah " << num << endl;
}
```

Keluaran programnya Adalah sebagai berikut;

```
Saya sedang mengisikan beberapa nilai pada TampilNilai.
Nilainya adalah 5
Nilainya adalah 10
Nilainya adalah 2
Nilainya adalah 16
sekarang saya kembali.
```

Setiap kali sebuah fungsi dalam program diatas tersebut dipanggil, variabel num akan mengambil nilai yang berbeda. Setiap ekspresi fungsi, nilai akan diberikan kedalam num yang digunakan sebagai argumen. Sebagai contoh, panggilan fungsi berikut digunakan untuk memasukan nilai 8 kedalam variabel num:

```
TampilNilai(3 + 5);
```

Jika kita memasukan jenis argumen yang tidak sama dengan jenis parameter, argumen tersebut akan digunakan secara otomatis. Misalnya, jika TampilNilai dari sebuah parameter adalah bertipe integer, argumen pada pemanggilan fungsi akan dipotong, sehingga nilai 4 akan dimasukan ke varibel num:

```
TampilNilai (4.7);
```

Seringkali hal tersebut berguna untuk digunakan oleh beberapa argumen dalam sebuah fungsi. Program dibawah ini menunjukkan sebuah fungsi dari fungsi yang memiliki tiga buah parameter.

Program 7.8.

```
#include <conio.h>
#include <iostream>

using namespace std;
```

```
//Prototipe Fungsi
void Tambahan(int, int, int);

int main()
{
   int value1, value2, value3;

   cout << "Masukan Tiga bilangan Integers dan saya akan menampilkan ";
   cout << "Penjumlahan: ";
   cin >> value1 >> value2 >> value3;
   Tambahan(nilai1, nilai2, nilai3);  // Call Tambahan dengan 3 arguments.
   getch();
   return 0;
}

void Tambahan(int num1, int num2, int num3)
{
    cout << (num1 + num2 + num3) << endl;
}
```

Keluaran Programnya adalah sebagai berikut;

```
Masukan Tiga bilangan Integers dan saya akan menampilkan Penjumlahan: 4 5 6
15
```

Dalam header fungsi Tambahan, daftar parameter berisi tiga definisi variabel yang dipisahkan oleh koma adalah sebagai berikut:

```
void Tambahan(int num1, int num2, int num3)
```

Hal yang perlu diperhatikan pada setiap variabel yaitu harus memiliki tipe data yang namanya sebelum telah tertulis. Error pada compiler akan terjadi jika daftar parameter yang dinyatakan sebagai int num1, num2, num3 ternyata bukan int num1, int num2, int num3.

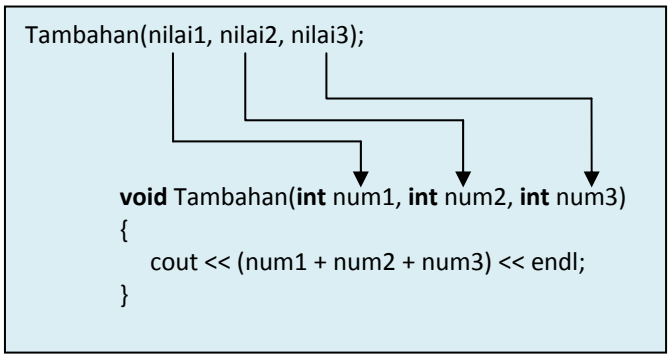
Dalam pemanggilan fungsi, sebuah variabel nilai1, nilai2 dan nilai 3 dapat digunakan sebagai argument:

```
Tambahan(nilai1, nilai2, nilai3);
```

Potongan program diatas jika diperhatikan ada perbedaan antara header fungsi dan panggilan fungsi saat melewati variabel sebagai argumen menjadi parameter. Dalam panggilan fungsi, kita tidak dapat menyertakan variabel jenis data didalam tanda kurung. Perhatikan potongan program dibawah ini:

```
Tambahan(int nilai1, int nilai2, int nilai3);
```

Ketika sebuah fungsi dengan beberapa parameter disebut, argumen akan dimasukan kedalam parameter yang di pesan. Hal tersebut dapat digambarkan seperti gambar dibawah ini.



Gambar 7.6. Fungsi dengan beberapa Parameter

Fungsi dibawah ini akan menyebabkan nilai 5 dimasukan kedalam parameter num1, nilai 10 dimasukan ke dalam num2, dan 15 dimasukan kedalam num3. Untuk lebih jelasnya perhatikan instruksi dibawah ini:

```
showSum(5, 10, 15);
```

meskipun pemanggilan fungsi berikut ini akan menyebabkan 15 yang dimasukan kedalam parameter num1, 5 dimasukan ke dalam num2, dan nilai 10 akan dimasukan kedalam num3:

```
Tambahan(15, 5, 10);
```

7.6. Passing Data by Value

Parameter khusus merupakan variabel tujuan yang ditentukan didalam tanda kurung dari definisi fungsi. Tujuan parameter tersebut adalah untuk menyampaikan informasi yng dilakukan oleh argumen dan tercantum dalam tanda kurung pada panggilan fungsi. Secara umum, ketika informasi disampaikan kedalam fungsi disebut

passed by value. Hal ini berarti parameter menerima copy dari nilai yang disampaikan kepadanya.

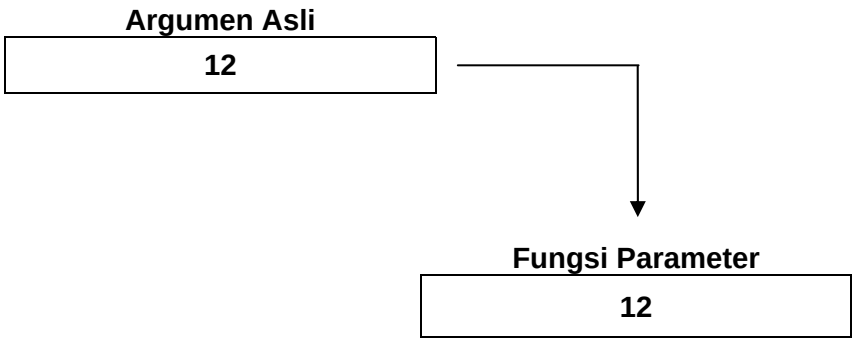
Jika sebuah parameter nilai berubah didalam fungsi maka hal tersebut tidak mempengaruhi pada argumen yang asli. Perhatikan program dibawah ini yang menunjukkan konsep mengenai *passed by value*.

Program 7.9.

```
include <iostream>

using namespace std;

void changeThem(int, double);

int main()
{
	int whole = 12;
	double real = 3.5;
	cout << "dalam nilai main adaIah" << whole << endl;
	cout << "dan nilai real adalah" << real << endl << endl;

	changeThem(whole, real);	// memanggil changeThem dengan 2 arguments

	cout << "sekarang kembali dalam main lagi, nilainya adalah ";
	cout << "semuanya masih " << whole << endl;
	cout << "dan nilai real adalah masih" << real << endl;
	return 0;
}

void changeThem(int i, double d)
{
	i = 100;
	d = 27.5;

	cout << "dalam fungsi changeThem nilainya diubah menjadi ";
	cout << i << endl;
	cout << "dan nilainya diubah menjadi " << d << endl << endl;
}
```

Pada program diatas walaupun parameter i dan f berubah dalam fungsi changeThem, kenyataannya seluruh argumen tidak diubah. Parameter i dan f hanya berisi salinan keseluruhan dan nyata. fungsi ChangeThem tidak memiliki akses ke argumen sebelumnya (original). Gambar dibawah menunjukan bahwa variabel

parameter disimpan dalam lokasi memori yang terpisah dari argumen yang asli. Perhatikan gambar berikut ini:



Gambar 7.7. Variabel parameter yang disimpan dalam memori

7.7. Pernyataan Kembali

Bila pernyataan terakhir dalam fungsi telah selesai eksekusi, fungsi akan diakhiri. Program akan kembali ke modul yang dipanggil dan meneruskan eksekusi dari titik dimana panggilan fungsi dilakukan sebelumnya. Hal sangat dimungkinkan, meskipun salah satu fungsi memaksa untuk kembali ketempat dimana pernyataan terakhir dari program sebelumnya yang dieksekusi. Ketika menemui pernyataan kembali, fungsi dengan segera diakhiri dan menuju program semula. Hal ini seperti ditunjukkan dalam Program dibawah. fungsi bagi digunakan untuk menunjukkan hasil pembagian dari arg1 dibagi arg2. Jika arg2 diatur ke nol, fungsi akan kembali tanpa melakukan pembagian, karena operasi pembagian tidak mungkin dilakukan.

Program 7.10.

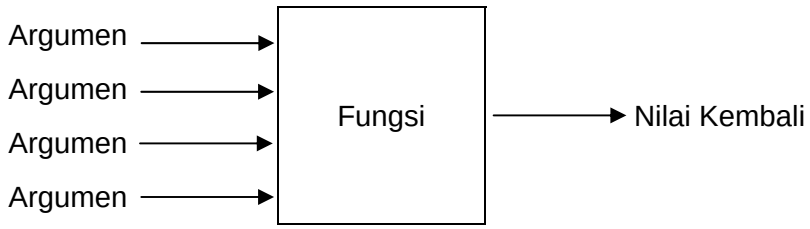
```
#include <iostream>

using namespace std;

void divide(double, double);

int main()
{
    double num1, num2;
    cout << "masukan dua angka yang akan dibagi \n";
    cout << "angka yang kedua: ";
```

```
    cin >> num1 >> num2;
    divide(num1, num2);
    return 0;
}

void divide(double arg1, double arg2)
{
    if (arg2 == 0.0)
    {
        cout << "maaf, tidak bisa dibagi dengan bilangan nol.\n";
        return;
    }
cout << "jawabanya adalah: " << (arg1 / arg2) << endl;
}
```

7.8. Mengembalikan Nilai dari Fungsi

Dari program diatas kita telah melihat bahwa informasi dapat dilewatkan menjadi salah satu fungsi oleh sebuah parameter. Informasi juga dapat dikembalikan dari fungsi, kembali menuju program yang memanggilnya. Walaupun beberapa argumen dapat dimasukan menjadi salah satu fungsi, hanya ada satu nilai yang dapat dikembalikannya. Sesuatu dari fungsi memiliki beberapa saluran komunikasi untuk menerima data (parameter), tetapi hanya satu saluran untuk mengirim data (return value).

Hal tersebut dapat digambarkan seperti dibawah ini:



Gambar 7.8. Fungsi dengan saluran beberapa data (parameter)

Dalam rangka untuk mengembalikan nilai-nilai dari beberapa fungsi, mereka harus "dibuat paket" sehingga mereka dianggap sebagai satu nilai. Tipe data nilai kembali mengawali fungsi dengan nama header fungsi dan prototipe. Berikut ini menyatakan prototipe fungsi dengan nama persegi yang menerima sebuah

argumen integer dan mengembalikannya menjadi integer:

```
int persegi(int);
```

instruksi diatas dapat didefinisikan dalam fungsi sebagai berikut:

```
int persegi(int number)
{
return number * number;
}
```

Fungsi tersebut hanya memiliki satu baris yang merupakan pernyataan kembali. Bila nilai fungsi kembali, maka hal tersebut harus memiliki pernyataan kembali. Ungkapan yang mengikuti kata kunci akan dievaluasi, dikonversikan ke tipe data fungsi kembali, dan dikirim kembali program yang memanggil fungsi tersebut. Untuk lebih jelasnya perhatikan Program dibawah ini:

Program 7.11

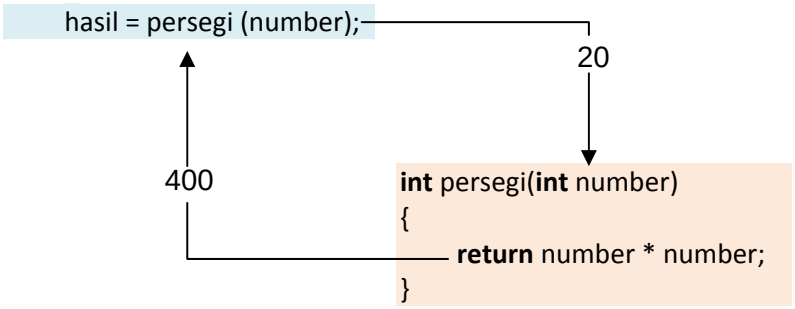
```
#include <iostream>

using namespace std;

int persegi(int);

int main()
{
	int number, result;

	cout << "Masukan angka yang akan diakarkan: ";
	cin >> number;
	hasil = persegi (number);
	cout << number << " Hasilnya adalah " << hasil << endl;
	return 0;
}

int persegi(int number)
{
	return number * number;
}
```

Dibawah ini merupakan sebuah baris memanggil fungsi persegi:

```
hasil = persegi (number);
```

Sebuah ekspresi adalah sesuatu yang memiliki nilai. Jika suatu fungsi mengembalikan nilai, panggilan ke fungsi merupakan sebuah ekspresi. Pernyataan diatas memberikan nilai kembali dari persegi ke variabel hasil.

Jadi, ketika nilai 20 dinyatakan sebagai argumen persegi, 20 kali 20, atau 400, dan dikembalikan ke hasil yang ditugaskan. Gambar dibawah ini menggambarkan bagaimana sebuah informasi disampaikan ke dan kembali dari fungsi.



Gambar 7.9. Penyampaian informasi dari fungsi

Sebenarnya dalam program diatas, variable hasil adalah variabel yang tidak penting. Adanya nilai pada fungsi persegi telah ditampilkan menggunakan cout objek, seperti ditunjukkan berikut dibawah ini:

```
cout << number << " persegi adalah " << persegi(number) << endl;
```

Dari program diatas kita sudah mengetahui bagaimana nilai dikembalikan oleh fungsi dan ditempatkan ke variabel atau dicetak. Program tersebut diatas Juga memungkinkan untuk menggunakan nilai yang dikembalikan oleh fungsi sebagai penghubung dalam pengujian sebuah ekspresi aritmatika. Perhatikan contoh berikut yang menggunakan dua pernyataan yang sempurna:

```
if (persegi(number) > 100)
    cout << "persegi besar\n";
    sum = 1000 + persegi (number);
```

Program dibawah menunjukkan versi dari fungsi persegi yang kembali senbagai duble. Fungsi tersebut digunakan dalam pernyataan matematis untuk menghitung daerah lingkaran. Program dibawah juga menggunakan fungsi getRadius untuk mendapatkan radius lingkaran dari pengguna dan kembali ke nilai utama.

Program 7.12

```
#include <iostream>
#include <iomanip>
```

```
using namespace std;

double getRadius();
double square(double);

int main()
{
	const double PI = 3.14159;
	double rad;

	cout << fixed << showpoint << setprecision(2);
	cout << "Program ini untuk menghitung luas lingkaran.\n";
	rad = getRadius();
	cout << "Luasnya adalah: " << PI * square(rad) << endl;
	return 0;
}

double getRadius()
{
double radius;
	cout << "masukan jari-jari lingkaran: ";
	cin >> radius;
	return radius;
}

double square(double number)
{
	return number * number;
}
```

Fungsi Persegi dalam Program 7.11 merupakan sebuah integer, sedangkan pada program 7.12 adalah fungsi ganda. Cara kembali jenis fungsi harus merupakan jenis data yang diinginkan untuk kembali dari fungsi. Jika fungsi kembali ke nilai ganda yang sedang ditugaskan ke variabel, maka variabel tersebut harus juga menjadi dua kali lipat. Jika dua nilai dikembalikan oleh fungsi persegi seperti pada program 7.12 maka akan ditugaskan ke salah satu variabel integer, sehingga nilai akan dipotong. Hal ini diilustrasikan dalam contoh program berikut:

```
int result;
result = square(2.7);
```

7.9. Pengembalian Nilai Boolean

Biasanya ada sesuatu yang dibutuhkan fungsi untuk menguji argumen dan akan mengembalikan nilai benar atau salah serta mengindikasikan kondisi yang ada. Misalnya, dalam sebuah program yang membutuhkan sesuatu untuk mengetahui apakah nomor genap atau ganjil, sebuah fungsi dapat ditulis kembali dengan benar jika argument genap dan false jika argumen yang ganjil. Untuk lebih jelasnya perhatikan program dibawah ini:

Program 7.13

```
#include <iostream>

using namespace std;

bool isEven(int);
int main()
{
      int val;

      cout << "masukan bilangan integer :";
      cout << "jika ganjil atau genap : ";
      cin >> val;

      if (isEven(val))
      cout << val << "adalah genap.\n";
      else
      cout << val << " adalah ganjil.\n";
      return 0;
}

bool isEven(int number)
{
      if (number % 2)
      return false;         // Bilangan adalah ganjil
      else
      return true;          // bilangan adalah genap.
}
```

Keluaran Program adalah sebagai berikut:

```
masukan bilangan integer : 5[Enter]
5 adalah ganjil.
```

Sebuah fungsi isEven disebut dalam pernyataan dibawah ini:

```
if (isEven(val))
```

ketika pernyataan IF dieksekusi, isEven disebut dengan val sebagai argument. Jika val genap, isEven adalah benar dan sebaliknya akan salah.

7.10. Menggunakan Fungsi dalam program menu

Dalam sebelumnya kita melihat menu-driven program yang menghitung biaya kesehatan untuk anggota klub. Program 7.14 dibawah adalah untuk meningkatkan modular versi program. Perhatikan bagaimana setiap fungsi, atau modul, yang dirancang untuk melakukan tugas tertentu.

Program 7.14

```
#include <iostream>
#include <iomanip>
#include <string>

using namespace std;

// Function prototypes
void displayMenu();
int getChoice();
void computeFees(string, double, int);

const double ADULT_RATE = 40.00,
SENIOR_RATE = 30.00,
CHILD_RATE = 20.00;

int main()
{
      int choice,
      months;

      cout << fixed << showpoint << setprecision(2);

      do
      { displayMenu();
          choice = getChoice();

          if (choice != 4)
          {
```

```
            cout << "berapa jumlah banyaknya bulan:? ";
            cin >> months;

            switch (choice)
            {
                case 1: computeFees("Dewasa", ADULT_RATE, months);
                break;
                case 2: computeFees("Anak", CHILD_RATE, months);
                break;
                case 3: computeFees("Senior", SENIOR_RATE, months);
            }
        }
    } while (choice != 4);
    return 0;
}

void displayMenu()
{
    cout << "\nmenu keanggotaan klub kesehatan \n\n";
    cout << "1. Keanggotaan dewasa \n";
    cout << "2. Keangotaan anak\n";
    cout << "3. senior\n";
    cout << "4. Keluar\n\n";
}

int getChoice()
{
    int choice;
    cin >> choice;
    while (choice < 1 || choice > 4)
    {
        cout << "pilihan hanya no 1 – 4 tekan enter. ";
        cin >> choice;
    }
    return choice;
}

void computeFees(string memberType, double rate, int months)
{
    cout << endl
    << "tipe keanggotaan: " << memberType << " "
    << "bayaran perbulan Rp" << rate << endl
    << "Jumlah bulan:" << months << endl
```

```
    << "Total:Rp."<< (rate * months)
    << endl << endl;
}
```

Yang perlu diperhatikan adalah fleksibilitas dari fungsi computeFees, yang disebut pada tiga tempat yang berbeda-beda dengan pernyataan switch. Hal ini disampaikan tiga argumen: string memegang jenis keanggotaan, double memegang biaya bulanan untuk jenis keanggotaan, dan integer memegang jumlah tagihan per bulan.

Tanpa argumen, kita akan menetapkan sesuatu yang diperlukan keseluruhan fungsi: satu untuk menghitung biaya keanggotaan dewasa, anak yang lain untuk menghitung biaya keanggotaan, dan ketiga untuk menghitung biaya keanggotaan senior. Karena kita dapat informasi yang berbeda lulus sebagai argumen ke fungsi tersebut, namun, kami mampu membuat satu tujuan umum-fungsi yang bekerja untuk semua tiga kasus.

7.11. Variabel Lokal dan Global

Seperti yang telah ditetapkan bahwa variabel didalam fungsi utama, Anda juga dapat menetapkan fungsi di dalam fungsi-fungsi lainnya. Variabel yang ditetapkan dalam fungsi lokal ke fungsi. Mereka tersembunyi dari pernyataan dalam fungsi lainnya, yang biasanya tidak bisa mengaksesnya. Program 7.15 menunjukkan bahwa variabel yang ditetapkan dalam suatu fungsi yang tersembunyi, dimana fungsi lainnya mungkin telah terpisah, berbeda variabel dengan nama yang sama.

Program 7.15

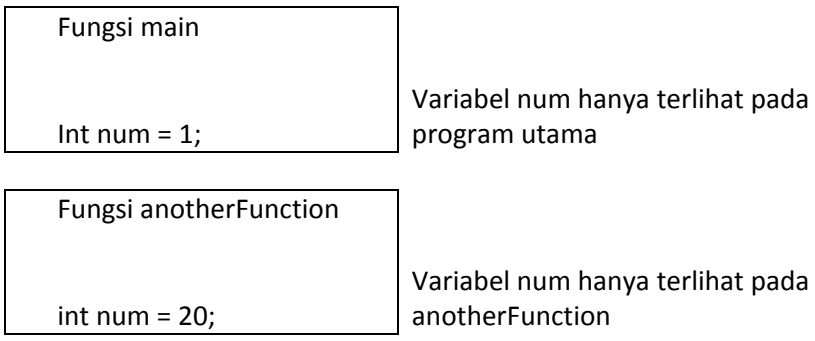
```
#include <iostream>

using namespace std;

void anotherFunction();

int main()
{
        int num = 1;
        cout << "Dalam program utatma,num adalah: " << num << endl;
        anotherFunction();
        cout << "Kembali dalam program utama, num masih" << num << endl;
        return 0;
}
```

```
void anotherFunction()
{
      int num = 20;        // variabel Local
      cout << "Dalam anotherFunction, num adalah: " << num << endl;
}
```

Keluaran Program adalah sebagai berikut:

```
Dalam program utatma,num adalah: 1
Dalam anotherFunction, num adalah: 20
Kembali dalam program utama, num masih 1
```

Meskipun terdapat dua variabel num, program ini hanya bisa "melihat" salah satu dari mereka setiap saat. Ketika program dijalankan dalam program utama, yang didefinisikan pada variabel num akan terlihat pada program utama. Ketika anotherFunction disebut sebagai variabel yang ditetapkan didalamnya akan terlihat, sehingga dalam variabel num program utama tersembunyi. Perhatikan program yang mengilustrasikan sifat tertutup dari dua fungsi, dimana bentuk kotak mewakili lingkup variabel



Gambar 7.10. Dua Fungsi yang mempunyai sifat tertutup,

Sebuah Lokal variabel dapat dikatakan sebagai variable yang aman dan tersembunyi dari fungsi lain, tetapi variable ini tidak menyediakan cara yang mudah untuk berbagi data. Jika sejumlah besar data harus dapat diakses oleh semua fungsi dalam program, variabel global dapat digunakan sebagai alternatif mudah. Sebuah variabel global merupakan variabel yang ditetapkan diluar oleh semua fungsi dalam sebuah program. Program 7.16 menunjukkan dua buah fungsi utama dan anotherFunction, dimana menggunakan akses yang sama yaitu variabel global num.

Program 7.16

```
#include <iostream>

using namespace std;

void anotherFunction();          // Fungsi  prototipe
int num = 2;                     // variabel Global

int main()
{
      cout << "dalam program utama, num adalah" << num << endl;
      anotherFunction();
      cout << "kembali dari program utama, num adalah" << num << endl;
      return 0;
}

void anotherFunction()
{
      cout << "dalam fungsi anotherFunction, num adalah " << num << endl;
      num = 50;
      cout << "tetapi, sekarang telah diubah menjadi " << num << endl;
}
```

Dalam Program 7.16 diatas, variabel num didefinisikan diluar semua fungsi. Karena muncul sebelum definisi pada program utama dan anotherFunction, maka keduanya memiliki akses ke fungsi tersebut.

Program 7.17

```
#include <iostream>

using namespace std;

int globalNum;                          // variabel Global. Otomatis diatur  ke null.
int main()
{
      cout << "globalNum adalah " << globalNum << endl;
      return 0;
}
```

Jika sebuah fungsi memiliki variabel lokal dengan nama yang sama sebagai variabel global, variabel lokal hanya dapat dilihat oleh fungsi. Hal ini ditunjukkan oleh Program dibawah ini:

Program 7.18

```
#include <iostream>

using namespace std;

// Funsi prototipe
void texas();
void arkansas();

int cows = 10;                    //variabel  global

int main()
{
      cout << "Disana banyak " << cows << " cows di program utama.\n";
      texas();
      arkansas();
      cout << "Kembali dalam program utama, disana banyak" << cows << " cows.\n";
      return 0;
}

void texas()
{ int cows = 100;                 //variabel  Local
      cout << "disana " << cows << " cows di texas.\n";
}

void arkansas()
{ int cows = 50;                  // variabel Local

      cout << "There are " << cows << " cows in arkansas.\n";
}
```

Ketika program diatas sedang menjalankan fungsi utama, variabel global cows akan terlihat. Dalam fungsi texas dan Arkansas, meskipun ada variabel lokal dengan nama cows. Variabel global tidak terlihat ketika program sedang menjalankan fungsi mereka. program dibawah merupakan program untuk mendaftar dan program ini menggunakan variabel global dan lokal. Fungsi ringUpSale menghitung dan menampilkan harga, pajak penjualan, dan subtotal untuk setiap item yang dibeli. Memiliki variabel lokal, pajak, yang memiliki nama yang sama

seperti variabel global. Pajak pada ringUpSale variabel yang digunakan untuk menghitung pajak penjualan pada item, sementara global pajak variabel yang digunakan oleh program utama untuk menghitung total pajak penjualan pada pembelian. Supaya lebih jelas, perhatikan program dibawah ini:

Program 7.19

```
#include <iostream>
#include <iomanip>

using namespace std;

                                        // Fungsi prototipe
void ringUpSale();

                                        // variabel Global
const double TAX_RATE = 0.06;
double tax, sale, total;

int main()
{
      char again;
      cout << fixed << showpoint << setprecision(2);
      do
      {
          ringUpSale();
          cout << "Apakah disana ada ada sesuatu yang dapat dijual? ";
          cin >> again;
      }
      while (again == 'y' || again == 'Y');
      tax = sale * TAX_RATE;
      total = sale + tax;
      cout << "\nPajak penjualannya adalah " << tax << endl;
      cout << "Total Pembayaran adalah " << total << endl;
      return 0;
}

void ringUpSale()
{                                       //Variabel Local
      int qty;
      double unitPrice, tax, thisSale, subTotal;

      cout << "\nJumlah: ";
```

```
    cin >> qty;
    cout << "Harga per satuan: ";
    cin >> unitPrice;

    thisSale = qty * unitPrice;   // total harga unit
    sale += thisSale;             // Update variabel global penjualan
    tax = thisSale * TAX_RATE;    // pembayaran pajak untuk item
    subTotal = thisSale + tax;    // subtotal untuk tiap item
    cout << "harga untuk tiap item:" << thisSale << endl;
    cout << "pajak untuk tiap item: " << tax << endl;
    cout << "SubTotal untuk tiap item: " << subTotal << endl;
}
```

7.12. Variabel Static Local

Jika suatu fungsi dipanggil lebih dari satu kali dalam sebuah program, maka nilai yang disimpan dalam fungsi sebagai variabel lokal tidak akan kuat menahan panggilan fungsi. Hal ini disebabkan karena variabel-variabel yang hancur ketika fungsi diakhiri dan kemudian kembali dibuat bila fungsi sudah dimulai lagi. Hal ini ditunjukkan dalam Program 7.20 dibawah ini:

.

Program 7.20

```
#include <iostream>

using namespace std;

// Fungsi prototipe
void showLocal();

int main()
{
    showLocal();
    showLocal();
    return 0;
}

void showLocal()
{
    int localNum = 5; //variabel Local
    cout << "localNum adalaj " << localNum << endl;
    localNum = 99;
```

```
}
```

Keluaran Program adalah sebagai berikut:

```
localNum is 5
localNum is 5
```

Walaupun pernyataan terakhir dalam fungsi showLocal took 99 pada localNum, variabel yang hancur bila fungsi kembali. Pada saat fungsi tersebut dipanggil, localNum akan recreated dan diinisialisasi ke 5 lagi.

Kadang-kadang keinginan untuk sebuah program untuk "mengingat" adalah nilai yang disimpan dalam variabel lokal antara fungsi panggilan. Hal ini dapat dicapai dengan membuat variabel statis. variabel Static adalah variable lokal yang tidak hancur bila fungsi kembali. Mereka ada untuk seluruh masa program, walaupun mereka hanya lingkup fungsi mereka yang ditetapkan. Program 7.21 menunjukkan beberapa karakteristik statik lokal variabel.

Program 7.21

```
#include <iostream>

using namespace std;

// Fungsi prototipe
void showStatic();

int main()
{
    for (int count = 0; count < 5; count++)
    showStatic();
    return 0;
}

void showStatic()
{
    static int statNum; // Static local variable
    cout << "statNum adalah" << statNum << endl;
    statNum++;
}
```

Program diatas terdapat variabel statNum adalah di tambahkan dengan satu (incremented) pada fungsi showStatic, dan tetap dengan nilai antara setiap panggilan fungsi. Perhatikan bahwa walaupun tidak

secara eksplisit statNum diinisialisasi, dimulai pada nol. Seperti variabel global adalah variabel lokal statik diinisialisasi ke nol secara default. Jika Anda memberikan nilai initialization statis variabel lokal, hanya dilakukan sekali initialization. Hal ini biasanya terjadi karena variabel inisialisasi dibuat, dan variabel lokal statik hanya dibuat sekali saat menjalankan sebuah program. program dibawah hanya sedikit modifikasi.

Program 7.22

```
#include <iostream>

using namespace std;

void showStatic();

int main()
{
      for (int count = 0; count < 5; count++)
      showStatic();
      return 0;
}

void showStatic(void)
{
      static int statNum = 5;
      cout << "statNum adalah " << statNum << endl;
      statNum++;
}
```

Keluaran program diatas adalah sebagai berikut:

```
statNum adalah 5
statNum adalah 6
statNum adalah 7
statNum adalah 8
statNum adalah 9
```

Walaupun pernyataan yang digunakan utuk mendefinisikan inisialisasi statNum ke 5, initialization tidak terjadi pada setiap kali fungsi dipanggil. Jika tidak, variabel y tidak dapat mempertahankan nilai-nya antara fungsi panggilan.

7.12. Soal Latihan

Jawablah soal latihan dibawah ini dengan baik dan benar.

1. Apa yang dimaksud dengan fungsi pada pemrograman komputer
2. Apakah alasanya menggunakan fungsi
3. Jelaskan cara pemanggilan fungsi didalam sebuah fungsi
4. Apakah yang dimaksud dengan argumen
5. Apakah perbedaan antara fungsi dengan prosedur
6. Buatlah program sederhana menggunakan fungsi untuk mencari bilangan genap antara 1 sampai dengan 10

BAB 8
OPERASI STRING

8.1. String pada bahasa C

String merupakan sebuah bentuk data yang sering dipakai dalam bahasa pemrograman untuk keperluan menampung dan memanipulasi data teks. Sebagai contoh fungsi string misalnya digunakan untuk menampung atau menyimpan sebuah kalimat dan lain sebagainya.

Pada bahasa C, string bukanlah merupakan tipe data berdiri tersendiri, melainkan hanyalah kumpulan dari nilai-nilai karakter yang berurutan dalam bentuk array berdimensi satu.

Dalam bahasa C++, string merupakan suatu nilai karakter yang berurutan dan disimpan dalam sebuah lokasi memori yang salalu diakhiri dengan karakter null. Sehingga untuk memanggil string akan selalu dalam bentuk karakter ASCII. Dalam sebuah program operasi string, karakter null biasanya ditulis ‘\0’, dimana hal tersebut biasanya secara umum berupa bilangan integer 0 atau nilai karakter

null konstanta. Kemudian pernyataan tersebut disimpan dalam bentuk karakter variabel seperti dibawah ini:

```
char ch1, ch2, ch3;
ch1 = '\0';
ch2 = 0;
ch3 = NULL;
```

Array merupakan sebuah lokasi memori yang berurutan untuk menyimpan nilai tipe data yang sama, tetapi pada string berupa array yang selalu diakhiri dengan karakter null. String dalam bahasa C++ akan muncul dalam program berupa salah satu dari tiga bentuk dibawah ini:

- "Hard-coded'' string literals
- Programmer mendefinisikan array dari karakter
- Pointers ke karakter

Tanpa memperhatikan dari ketiga bentuk string yang muncul dalam program tersebut, string selalu mempunyai ciri array yang diakhiri oleh karakter null dan direpresentasikan dalam program dalam bentuk pointer pada karakter pertama dalam sebuah array. Dengan kata lain tipe string pada bahasa C++ adalah:

```
char *
```

dimana hal tersebut diatas adalah tipe char string menggunakan sebuah pointer

8.1.1. Konstanta String

Penulisan String Secara harafiah sebenarnya dapat disebut juga dengan konstanta string yang dapat ditulis secara langsung dalam program secara berurutan dengan menambah tanda ketik dua. Perhatikan string dibawah ini:

```
"siapakah nama anda?"
"gatotkaca"
```

Kedua penulisan tersebut diatas merupakan cara penulisan string. Perhatikan contoh program dibawah ini:

Program 8.1

```
#include <iostream>

using namespace std;

int main()
{
        cout << "gatotkaca";
        return 0;
}
```

Pada program diatas dapat diketahui bahwa compiler menangani string yang datang dalam bentuk string literal. Ketika compiler menemui sebuah string "gatotkaca", maka hal tersebut akan dialokasikan dala sebuah array dengan panjang sepuluh karakter dalam bentuk data dalam program.

Compiler kemudian akan menyimpan sembilan karakter dalam bentuk string "gatotkaca". Dalam penyimpanan pertama dimasukan karakter kesembilan kedalam array dan diakhiri dengan karakter null pada penyimpanan akhir. Langkah terakhir compiler melewatkan nilai tipe char * dan alamat array dari karakter pertama menjadi obyek cout.

g	a	t	o	t	k	a	c	a	\0

Gambar 8.1. Contoh sebuah string

Pada gambar diatas setiap karakter akan menempati memori sebesar 1 byte. Byte terakhir otomatis akan berisi karakter NULL (\0). Dengan mengetahui bahwa suatu string diakhiri nilai NULL, maka akhir dari nilai suatu string akan dapat dideteksi. Sebagai sebuah array karakter, karakter pertama dari nilai string mempunyai indeks ke-0, karakter kedua mempunyai indeks ke-1, dan seterusnya. Program dibawah ini mengilustrasikan sebuah string yang diperlakukan oleh compiler sebagai sebuah nilai dengan type data char* . Hal tersebut merupakan sebuah operasi pointer pada sebuah karakter.

Program 8.2

```
#include <iostream>

using namespace std;

int main()
{
        string storedAt = " disimpan pada ";
        char *p, *q;
        // memasukan string ke pointer sebagai char
        p = "Hello ";
        q = "Bailey";
        // pernyataan berikut sama dengan
        cout << p << q << endl;
        //cetak alamat dimana string C disimpan
        cout << p << storedAt <<int(p)<< endl;
        cout << q << storedAt <<int(q)<< endl <<int ("string lainnya");
        return 0;
}
```

Keluaran program diatas adalah sebagai berikut:

```
Hello Bailey
Hello disimpan pada 4309447
Bailey disimpan pada 4309454
4309461
```

Dua tugas pada program diatas adalah menampilkan string dalam bentuk pointer menjadi char dengan menggunakan variabe type char*. Pointer p dan q kemudian menahan sebagai alamat dari dua string. Dengan memilih sebuah pointer int maka akan mengetahui dimana

konstanta string disimpan dalam sebuah memori. Dalam kasus ini sebuah compiler telah menyimpan semua konstanta string pada sebuah program dalam lokasi memori yang berurutan.

8.1.2. Variabel String

Sebuah variabel string dapat ditahan oleh string pada bahasa C yang ditulis dalam bentuk kode program. Untuk memiliki suatu karakter string dari sebuah keyboard, atau dari sebuah file, maka harus dapat mendefinisikan sebuah array untuk menampung karakter string tersebut. Di dalam melakukan hal ini, yang perlu pastikan adalah pada saat mengalokasikan satu tambahan isi array untuk mengakhiri karakter null. Sebagai contoh, jika string akan lebih dari 19 karakter panjangnya, maka perlu mengalokasikan sebuah array dengan 20 karakter, seperti contoh penulisan dibawah ini:

```
char company[20];
```

instruksi tersebut diatas merupakan instruksi untuk mendeklarasikan variabel string dengan panjang maksimal 20 karakter (termasuk karakter NULL). Deklarasi tersebut sebenarnya tidak lain merupakan deklarasi array bertipe char.

Pada sebuah konstanta string, compiler string akan menunjukkan string melalui alamat pertama karakter string, dalam hal ini, array identifier. Pada satu array dentifier dengan tanpa tanda kurung diinterprestasikan oleh compiler sebagai alamat pertama yang masuk pada array. Suatu string dalam bahasa c didefinisikan sebagai satu array dapat diberi nilai dengan menginisialisasi string tersebut. Selain memberi nilai dapat juga dilakukan dengan membaca karakter string melalui sebuah keyboard atau file, atau dapat juga dengan melakukan copy karakter ke dalam array satu karakter pada saat yang sama. Hal ini dapat juga dikatakan bahwa suatu variabel string dapat diinisialisasi seperti halnya array yang lain. Namun tentu saja elemen terakhirnya haruslah berupa karakter NULL.

```
char corporation[]
={'G','r','a','p','h','I','c','s','\0'};
```

yang menyatakan bahwa name adalah variabel string dengan nilai awal berupa string: “Graphics” Bentuk inisialisasi yang lebih singkat:

```
char corporation[ ] = "Graphics";
```

atau contoh lain bisa juga ditentukan jumlah karakternya:

```
char company[20] = "Robotic Systems,
inc.";
```

Pada bentuk penulisan diatas, karakter NULL tidak perlu ditulis. Secara implisit akan disisipkan oleh kompiler. Dalam penulisan yang perlu diperhatikan, bila corporation dideklarasikan sebagai string, penugasan (*assignment*) suatu string ke variabel string dan tidak diperkenankan menulis seperti dibawah ini:

```
corporation = “Graphics";
```

dalam melakukan inisialisasi array sebuah string caranya adalah, ukuran dari array di dalam sebuah array merupakan sebuah opsional. Jika hal tersebut tidak ditentukan, maka compiler itu akan mengatur ukuran string sesuai dengan banyaknya karakter yang ada didalam string tersebut seperti contoh diatas.

String pada bahasa C mendefinisikan sebagai aray atau larik yang dapat terbaca dan ditulis dengan berbagai object, operator, maupun keanggotaan fungsi masukan atau keluaran sebuah class. Suatu string pada bahasa C akan disimpan seorang programmer untuk mendefinisikan array yang dapat diproses dengan notasi subscript standar. Program dibawah merupakan sebuah contoh program string. Contoh tersebut akan mengeluarkan satu karakter string pada waktu yang sama, dan berhenti ketika itu menemukan akhiran null. Instruksi tersebut menggunakan fungsi keanggotaan getline yang dimiliki oleh bahasa C. Untuk lebih jelasnya perhatikan program dibawah ini:

Program 8.3

```
#include <iostream>

using namespace std;

int main()
{
	const int LENGTH = 80;
	char line[LENGTH];
	int count = 0;

	cout << "Masukan kalimat tidak lebih dari "
	<< LENGTH-1 << " characters:\n";
	cin.getline(line, LENGTH);
	cout << "Kalimat yang anda Masukan Adalah:\n";
	while (line[count] != '\0')
	{
		cout << line[count];
		count++;
	}
return 0;
}
```

Keluaran program adalah sebagai berikut:

```
Masukan kalimat tidak lebih dari 79 characters:
Bahasa C++ sangat menantang Ya ??[Enter]
Kalimat yang anda Masukan Adalah:
Bahasa C++ sangat menantang Ya ??
```

8.2. Pointer pada Operasi String

Seperti yang sudah kita ketahui bahwa string pada bahasa C dapat diwakili ketika konstanta string, atau array sebuah karakter. Keduanya metoda ini mengalokasikan array dan lalu menggunakan alamat array sebagai suatu pointer/penunjuk untuk char sebagai aktualisasi yang merepresentasikan string.

Perbedaan antara keduanya adalah bahwa dalam kasus yang pertama, array digunakan untuk menyimpan string yang dialokasikan secara implisit oleh compiler, sedangkan pada kasus kedua, array secara eksplisit dialokasikan oleh programmer.

Metoda yang ketiga adalah dengan merepresntasikan string bahasa C menggunakan pointer untuk operasi char untuk menunjuk string pada bahasa C. Dengan metode ini penyimpanan sudah dialokasikan oleh salah satu atau dua metoda yang lain. Di sini ada beberapa contoh penggunaan string bahasa C adalah sebagai berikut:

```
char name[20] = "Sekolah menengah";
char *p;
p = name;
cout << p << endl;        // cetak p
p = "sekolahku";
cout << p << endl;        // cetak p
```

Keuntungan utama dalam menggunakan sebuah variabel penunjuk pointer adalah untuk merepresentasikan string pada bahasa C yang mempunyai kemampuan untuk membuat penunjuk titik pointer yang menunjuk string yang berbeda.

Cara lain cara menggunakan penunjuk pointer char string pada bahasa C++ adalah dengan mendefinisikan penunjuk pointer dan kemudian menetapkannya untuk menunjuk secara dinamis dan mengalokasikan penyimpanan yang dikembalikan oleh operator yang baru. Hal ini dapat digambarkan pada contoh program dibawah ini:

Program 8.4

```
#include <iostream>

using namespace std;

int main()
{
    const int NAME_LENGTH = 50;
    char *pname;

    pname = new char[NAME_LENGTH];        // alokasi penyimpanan
    cout << "Masukan nama Anda: ";
    cin >> pname;                         // cetak nama
    cout << "Hello " << pname;
    return 0;
```

```
}
```

Keluaran program diatas setelah diberi masukan adalah:

```
Masukan nama Anda:  Suprapto [Enter]
Hello Suprapto
```

Suatu kesalahan umum yang sering terjadi dalam menggunakan penunjuk pointer char pada string bahasa C adalah ketika menggunakan penunjuk pointer tetapi tidak menunjuk dan mengalokasikan string dengan baik. Perhatikan contoh kode dibawah ini:

```
char *pname;
cout << "Masukan Nama Anda: ";
cin >> pname;
```

kesalahannya adalah karena program mencoba untuk membaca sebuah string ke dalam lokasi memori yang ditunjuk oleh **pname**, ketika **pname** belum diinisialisasi dengan baik.

8.3. Library String Bahasa C++

Library C++ menyediakan banyak fungsi untuk memanipulasi dan menguji string. Sebagai contoh, sebuah program segmen berikut menggunakan fungsi **strlen** untuk menentukan panjang sebuah string yang disimpan dengan nama **name**:

```
char name[50] = "Thomas Edison";
int length;
length = strlen(name);
```

Fungsi strlen untuk menerima string sebagai argumentasi. Hal tersebut digunakan untuk mengembalikan panjang string, dimana banyaknya karakter akan sampai tetapi tidak termasuk tanda null akhir string. Sebagai hasil dari panjang variabel string akan memiliki jumlah karakter 13 yang disimpan di dalam string. Panjang sebuah string harus tidak dikacaukan dengan ukuran dari array yang menahannya. Hal yang perlu diingat adalah bahwa satu-satunya informasi yang sedang diberikan kepada strlen adalah alamat permulaan dari string. Hal tersebut tidak mengetahui yang mana array sebagai tujuan akhir, sehingga hal tersebut akan mencari kode null yang menandai adanya ujung string.

Karena string pada bahasa C berupa penunjuk pointer untuk tipe data char, string menangani fungsi yang mengambil parameter berupa array dari sebuah char atau ekivalensinya, penunjuk pointer untuk menangani tipe data char. string pada C dapat juga dilewatkan pada fungsi di dalam tiga bentuk manapun dimana string dapat mengambil:

- Konstanta string
- Nama array yang menyimpan string
- Variabel Pointer yang menangani alamat C

Sebagai contoh pada operasi untuk menghitung panjang string, maka fungsi **strlen** dapat digunakan sebagai ditunjukkan dibawah ini:

```
length = strlen("Thomas Edison");
```

Contoh lain suatu fungsi penanganan string pada C adalah **strcat**. Fungsi **strcat** mengambil dua string ketika sebuah parameter menggabungkan mereka, mengembalikan suatu string yang terdiri dari semua karakter yang diikuti string pertama oleh karakter string kedua. Di bawah ini adalah satu contoh penggunaan pada program:

```
char string1[13] = "Hello ";
char string2[7] = "World!";
cout << string1 << endl;
cout << string2 << endl;
strcat(string1, string2);
cout << string1 << endl;
```

pernyatan-pernyataan tersebut akan menghasilkan keluaran sebagai berikut:

```
Hello
World!
Hello World!
```

Fungsi strcat untuk menyalin isi dari string2 hingga selesai dari string1. Di dalam contoh ini, string1 berisi string "Hello " sebelum memanggil ke strcat. Setelah panggilan dilakukan, maka nilainya akan berisi string " Hello World!". Gambar dibawah menunjukkan isi dari kedua array sebelum dan setelah fungsi tersebut dipanggil.

Sebelum memanggil strcat (string1,string2);

String1

H	**e**	**l**	**l**	**o**		**\0**						

String2

W	**o**	**r**	**l**	**d**	**!**	**\0**

Setelah memanggil strcat (string1, string2)

String1

H	**e**	**l**	**l**	**o**		**W**	**o**	**r**	**l**	**d**	**!**	**\0**

String2

W	**o**	**r**	**l**	**d**	**!**	**\0**

karakter yang terakhir pada string1 sebelum tanda null adalah sebagai ruang atau space. Fungsi strcat tidak untuk menyisipkan sebuah ruang, dengan demikian hal tersebut adalah tanggung jawab seorang programmer bahwa space telah dipastikan satu telah di sana, jika hal tersebut diperlukan. Hal tersebut merupakan tugas seorang programmer untuk memastikan bahwa array dapat menjaga string1 dan sudah cukup besar untuk menahan string1 dan string2 serta tanda akhiran null. Pada kasus ini, sebuah segmen program yang menggunakan operator sizeof untuk menguji ukuran array sebelum strcat dipanggil:

```
if (sizeof(string1) >= (strlen(string1)+ strlen(string2)+1))
        strcat(string1, string2);
else
        cout << "String1 tidak cukup besar untuk kedua string tersebut.\n";
```

Array tidak bisa ditugaskan selain dengan operator = . Masing-masing unsur individu harus mempunyai tugas, biasanya di dalam sebuah program pengulangan. Fungsi strcpy, dapat digunakan untuk menyalin sebuah string ke bentuk yang lain. Contoh penggunaannya dapat dilihat pada program dibawah ini:

```
char name[20];
strcpy(name, "Albert Einstein");
```

Alasan penggunaan fungsi **strcpy** adalah string pada bahasa C. String yang kedua dicopy pada alamat yang ditetapkan oleh argumen string yang pertama. Jika sesuatu sudah disimpan di dalam lokasi yang telah disesuaikan oleh argumen yang pertama, hal tersebut akan terjadi overwritten, seperti ditunjukkan dalam segmen program berikut ini:

```
char string1[10] = "Hello", string2[10] =
"World!";
cout << string1 << endl;
cout << string2 << endl;
strcpy(string1, string2);
cout << string1 << endl;
cout << string2 << endl;
```

sehingga keluaran program adalah sebagai berikut:

```
Hello
World!
World!
World!
```

8.4. Membandingkan string

Operator assignment dan operator relasional bekerja dengan class string pada bahasa C++ string karena mereka dapat terjadi overload yang bekerja bersama kelas tersebut. Meskipun demikian, sama seperti operator assignment tidak bisa digunakan untuk memberi pada string C, operator relasional <=, <, >, >=, !=, dan == tidak bisa digunakan untuk membandingkan pada string. Hal ini dikarenakan ketika digunakan dengan string, operator tersebut membandingkan alamat-alamat dimana strings tersebut disimpan daripada membandingkan karakter yang berurutan secara nyata yang menjadi anggota string tersebut.

Program dibawah menunjukkan hasil keluaran yang salah dimana program tersebut mencoba membandingkan string menggunakan persamaan operator diatas.

Program 8.5.

```
#include <iostream>

using namespace std;

int main()
{
    const int LENGTH = 40;
    char firstString[LENGTH], secondString[LENGTH];

    cout << "Masukan string: ";
    cin.getline(firstString, LENGTH);
    cout << "Masukan string lain: ";
    cin.getline(secondString, LENGTH);
```

```
    if (firstString == secondString)
        cout << "Yang anda masukan stringnya sama\n";
    else
        cout << "String Tidak Sama.\n";
    return 0;
}
```

Keluaran program diatas adalah:

```
Masukan string: Alfonso[Enter]
Masukan string lain: Alfonso[Enter]
String Tidak Sama.
```

Meskipun kedua string tersebut yang dimasukan serupa, Program diatas akan melaporkan bahwa mereka tidaklah sama. Hal ini dikarenakan adanya instruksi dibawah ini:

```
firstString == secondString
```

hal yang digunakan dalam dalam program diatas adalah statemen IF untuk membandingkan alamat memori dari dua aray tersebut, daripada membandingkan karakter string dan menyimpan alamat tersebut. Karena alamatnya berbeda, maka perbandingan menghasilkan suatu hasil yang salah. Sebenarnya, dalam bahasa C++, perbandingan dapat dilakukan seperti dibawah ini:

```
"abc" == "abc"
```

Biasanya instruksi diatas akan menghasilkan suatu hasil yang salah. Ini dikarenakan kebanyakan compiler tidak memeriksa atau melihat jika konstanta string sudah ditemui sebelumnya, dan akan menyimpan kedua string-string pada alamat memori yang berbeda. Compiler tersebut akan membandingkan kedua alamat-alamat yang berbeda, kemudian memberi suatu nilai hasil yang salah.

8.4.1. Fungsi strcmp

Untuk membandingkan string pada C dengan baik, maka perlu menggunakan fungsi library **strcmp**. Fungsi ini mengambil dua string sebagai parameter dan mengembalikan bilangan integer untuk menandai,bagaimana dua string dibandingkan antara satu sama lain. Perhatikan instruksi dibawah ini:

```
int strcmp(char *string1, char *string2);
```

menunjukkan bahwa fungsi pengambilan dua string pada C sebagai parameter dan mengembalikan hasil bilangan integar. Nilai dari hasil tersebut diatur menurut aturan sebagai berikut:

- Hasilnya kosong jika kedua string adalah sama pada karakter melalui karakter dasar
- Hasilnya negatif jika string1 datang sebelum string2 dalam urutan menurut abjad

- Hasilnya positif jika string1 mengikuti string2 dalam urutan menurut abjad

Perhatikan contoh program dengan menggunakan strcmp untuk menentukan kesamaan dua buah string:

```
if(strcmp(string1, string2) == 0)
    cout << "string adalah sama";
else
    cout << "String tidak sama";
```

Pada program diatas, yang hasilnya salah adalah untuk menguji dua buah string dengan operator relational, dapat ditulis ulang secara benar dengan fungsi strcmp, seperti yang ditunjukkan pada program dibawah ini:

Program 8.6

```
#include <iostream>
#include <cstring>

using namespace std;

int main()
{
    const int LENGTH = 40;
    char firstString[LENGTH], secondString[LENGTH];

    cout << "Masukan String: ";
    cin.getline(firstString, LENGTH);
    cout << "Masukan string lain: ";
    cin.getline(secondString, LENGTH);
    if (strcmp(firstString, secondString) == 0)
        cout << "Yang anda masukan stringnya sama\n";
    else
        cout << "String tidak sama.\n";
    return 0;
}
```

Keluaran program diatas adalah sebagai berikut:

```
Masukan String: Alfonso[Enter]
Masukan string lain: Alfonso[Enter]
Yang anda masukan stringnya sama
```

Fungsi strcmp bersifat case-sensitive ketika digunakan untuk membandingkan string. Jika pengguna memasukan "Anjing" dan "anjing" pada program diatas, maka akan melaporkan bahwa kedua string tidaklah sama karena ada perbedaan huruf kapital dan kecil pada kata

anjing. Kebanyakan compiler menyediakan **strcmp** versi tidak standar untuk melaksanakan perbandingan case-insensitive.

Sebagai contoh, Borland C++ mempunyai fungsi **stricmp**. Hal tersebut hampir sama dengan **strcmp** kecuali mengenai case dari karakter tersebut yang akan diabaikan.

Program dibawah ini adalah contoh yang lebih praktis bagaimana **strcmp** digunakan. Hal tersebut akan minta kepada user untuk memasukan sejumlah angka lebih dari satu sesuai dengan yang mereka beli. Bagian angka merupakan digit, huruf, dan tanda penghubung, sehingga dengan demikian hal tersebut akan disimpan sebagai suatu string. Begitu pengguna memasukan angka, program akan menampilkan harga tersebut.

Program 8.7

```
include<conio.h>
#include <iostream>
#include <cstring>
#include <iomanip>

using namespace std;

int main()
{
   const double A_PRICE = 100.00, B_PRICE = 150.00;
   const int PART_LENGTH = 8;
   char partNum[PART_LENGTH];

   cout << "kode nomer ganda adalah:\n";
   cout << "\tKotak besar, kode nomernya: S147-29A\n";
   cout << "\tRak Besar, kode nomernya: S147-29B\n";
   cout << "masukan kode nomer yang ada\n";
   cout << "ingin membeli: ";
   cin >> setw(9);
   cin >> partNum;
   cout << showpoint << fixed;
   cout << setprecision(2);
   if (strcmp(partNum, "S147-29A") == 0)
        cout << "harganya adalah $" << A_PRICE << endl;
   else if (strcmp(partNum, "S147-29B") == 0)
        cout << "harganya adalah $" << B_PRICE << endl;
   else
        cout << partNum << " nomer yang dimasukan tidak sah.\n";
   getch();
   return 0;
```

```
}
```

Keluaran program diatas adalah sebagai berikut:

```
kode nomer ganda adalah:
    Kotak besar, kode nomernya: S147-29A
    Rak Besar, kode nomernya: S147-29B
masukan kode nomer yang ada
ingin membeli: S147-29A[Enter]
harganya adalah $100.00
```

8.5. Operator logika NOT

Beberapa para programmer lebih suka menggunakan operator logika NOT dengan strcmp ketika menguji persamaan string-string. Karena logika 0 dipertimbangkan sebagai logika salah, operator ! akan mengkonversi nilai tersebut menjadi benar. Ekspresi **strcmp** (string1, string2) akan kembali benar jika kedua string adalah sama, dan salah ketika string berbeda. Pada kedua statemen berikut melaksanakan operasi yang sama:

```
if (strcmp(firstString, secondString) == 0)
if (!strcmp(firstString, secondString))
```

8.6. Pengurutan String

Program sering ditulis dengan dicetak berurutan menurut daftar abjad. Misalnya penggunaan sistem komputer pada sebuah toko serba ada untuk menyimpan nama dan alamat pelanggan dalam sebuah file. Nama-nama tersebut tidak akan muncul dalam file yang berurutan menurut abjad tetapi dalam order atau pesanan, dimana sesuai operator memasukkan mereka.

Jika suatu daftar tersebut dicetak dalam urutan pesanan tersebut, maka akan menjadi sangat sulit untuk menempatkan setiap nama tersebut. Daftar tersebut mau tidak mau harus diurutkan sebelum dicetak. Karena nilai yang dikembalikan oleh **strcmp** berdasarkan pada urutan alfabet relative pesanan dari kedua string yang sedang dibandingkan, maka hal tersebut dapat digunakan dalam program untuk jenis string pendek. Program dibawah akan meminta pemakai untuk memasukan dua nama, yang kemudian dicetak dalam secara urutan alfabet pemesan.

Program 8.8

```
#include<conio.h>
#include <iostream>
#include <cstring>

using namespace std;

int main()
```

```
{
  const int PANJANG_NAMA = 30;
  char name1[PANJANG_NAMA], name2[PANJANG_NAMA];

  cout << "Masukan Nama (last Name): ";
  cin.getline(name1, PANJANG_NAMA);
  cout << "Masukan Nama Lain: ";
  cin.getline(name2, PANJANG_NAMA);
  cout << "Disini nama akan di urutkan berdasarkan alphabet:\n";
  if (strcmp(name1, name2) < 0)
    cout << name1 << endl << name2 << endl;
  else if (strcmp(name1, name2) > 0)
    cout << name2 << endl << name1 << endl;
  else
    cout << "Anda memasukan nama yang SAMA!\n";
  getch();
  return 0;
}
```

Keluaran program diatas adalah sebagai berikut:

```
Masukan Nama (last Name): suprapto
Masukan Nama Lain: anwar
Disini nama akan di urutkan berdasarkan alphabet:
anwar
suprapto
```

setelah mempelajari beberapa program string diatas, dibawah ini merupakan tabel ringkasan sebuah penanganan fungsi string.

Tabel 8.1. Fungsi untuk menangani string

FUNGSI	PENJELASAN
strlen	Menerima string sebagai argumen. Pernyataan panjang string (tidak termasuk null terminator. Contoh Penggunaan: len = strlen (nama);
strcat	Menerima dua string sebagai argumen. Fungsi menambahkan isi string kedua untuk pertama string. (Yang pertama adalah string diubah, kedua string tersebut dibiarkan tidak berubah.) Contoh Penggunaan: strcat (string1, string2);
strcpy	Menerima dua string sebagai argumen. Fungsi salinan kedua string untuk pertama string. String miliknya kedua string dibiarkan tidak berubah. Contoh Penggunaan: strcpy (string1, string2);
strncpy	Menerima dua string dan argumen integer. Argumen yang ketiga, sebuah

	integer, menunjukkan berapa banyak karakter yang dicopy dari string kedua menuju string pertama. Jika kurang dari string2 sebanyak n karakter, string1 adalah ditambahkan dengan dengan karakter '\ 0'. Contoh Penggunaan: strncpy (string1, string2, n);
strcmp	Menerima dua string argumen. Jika string1 dan string2 sama, fungsi ini memberikan hasil 0. Jika string2 alfabetnya lebih besar dari string1, ia kembali angka negatif. jika String2 alfabetnya kurang dari string1, ia kembali angka positif. Contoh Penggunaan: if (strcmp (string1, string2))
strstr	Pencarian untuk pertama terjadinya string2 dalam string1. Jika terjadinya string2 ditemukan, fungsi akan mengembalikan pointer string pertama. Jika tidak, ia akan mengembalikan NULL pointer (alamat 0). Contoh Penggunaan: cout <<strstr (string1, string2);

Fungsi yang terakhir dalam tabel diatas adalah strstr, yang digunakan untuk mencari sebuah string dalam dari sebuah string yang penjang. Sebagai contoh, misalnya digunakan untuk mencari string "tujuh" didalam sebuah string kalimat yang lebih besar "Nilai Empat dan tujuh pada tahun yang lalu". Fungsi argumentasi yang pertama adalah string dicari, dan argumentasi yang kedua adalah string untuk mencari. Jika fungsi menemukan string dalam string yang kedua terlebih dulu hal tersebut kembalikan ke alamat kejadian string didalam string yang kedua. Sebaliknya jika hal tersebut tidak dikembalikan alamat 0, atau alamat NULL. Di bawah ini adalah contoh pengunaan fungsi tersebut:

```
char array[] = "Nilai empat pada tujuh tahun yang lalu";
char *strPtr;
cout << array << endl;
strPtr = strstr(array, "tujuh");
        // mencari string "tujuh"
cout << strPtr << endl;
```

Dalam segmen program yang sebelumnya, strstr akan menempatkan string "tujuh" di dalam string "Nilai Empat pada tujuh tahun yang lalu". Hal tersebut akan kembalikan kealamat dari karakter yang pertama didalam "tujuh", yang selanjutnya akan disimpan dalam variabel penunjuk pointer strPtr. Jika menjalankan bagian dari suatu program engkap, segmen tersebut akan menampilkan sebagai berikut:

```
Nilai Empat pada tujuh tahun yang lalu
tujuh pada tahun yang lalu
```

Fungsi strstr dapat bermanfaat dalam setiap program yang harus menempatkan informasi didalam string satu atau lebih. Program dibawah merupakan contoh, menyimpan suatu database sejumlah produk dan penjelasan dalam satu array pada string. Hal tersebut memperbolehkan pengguna untuk mengetahui penjelasan sebuah produk dengan memasukan semua atau bagian dari nomor produknya.

Program 8.9

```
#include<conio.h>
#include <iostream>
#include <string>

using namespace std;

int main()
{
   const int N_ITEMS = 5, S_LENGTH = 31;
   char prods[5][S_LENGTH] = {"TV311 televisi 31 inch",
   "CD111 CD Player",
   "MC123 Mesin Cuci",
   "TM456 tape Mobil",
   "PC955 Personal Computer"};
   char lookUp[S_LENGTH], *strPtr = NULL;
   int index;

   cout << "\tProduct Database\n\n";
   cout << "Masukan Kode angka product untuk mencari data: ";
   cin.getline(lookUp, S_LENGTH);

   for (index = 0; index < N_ITEMS; index++)
   {
     strPtr = strstr(prods[index], lookUp);
     if (strPtr != NULL)
      break;
   }
    if (strPtr == NULL)
       cout << "kode tidak sesuai dengan produk.\n";
    else
       cout << prods[index] << endl;
       getch();
       return 0;
}
```

Keluaran program diatas adalah:

```
Product Database
Masukan Kode angka product untuk mencari data: CD111[Enter]
CD111 CD Player


Product Database
```

```
Masukan Kode angka product untuk mencari data: GJ987[Enter]
kode tidak sesuai dengan produk.
```

Dalam program tersebut diatas, karena pengulangan tiap putaran melalui string dalam pemanggilan array maka pernyataannya adalah sebagai berikut:

```
strPtr = strstr(prods[index], lookUp);
```

Fungsi strstr mencari string yang sesuai oleh prods[index] karena nama yang dimasukkan oleh pengguna, yang mana disimpan dalam lookUp. Jika lookUp ditemukan di dalam prods[index], fungsi akan mengembalikan alamatnya. Dalam kasus tersebut, statemen if menyebabkan pengulangan akan berakhir.

```
if (strPtr != NULL)
break;
```

Di luar pengulangan tersebut, sesuai statemen IF-ELSE untuk menentukan jika string dimasukkan oleh pengguna ditemukan dalam array. Jika tidak, maka akan menginformasikan kepada pengguna bahwa tidak ada produk yang sesuai ditemukan. Sebaliknya, nomor produk dan deskripsi akan ditampilkan:

```
if (strPtr == NULL)
    cout << "kode tidak sesuai dengan
                produk.\n";
else
    cout << prods[index] << endl;
```

8.7. Konversi String/Numeric

Ada suatu perbedaan besar antara nomor yang disimpan sebagai string dan yang disimpan sebagai nilai klasifikasi. String "26792" sebenarnya tidak sebagai nomor, tetapi rangkaian yang diwakili kode-kode ASCII untuk setiap digit-digit dari nomor. Rangkaian tersebut menggunakan enam byte dari memori termasuk kode akhir string (null).

Karena itu bukan sebagai nomor nyata, maka tidak mungkin untuk melaksanakan operasi matematik dengannya, kecuali jika hal tersebut dikonversi menjadi nilai numerik. Beberapa fungsi yang ada dalam pustaka C++ untuk mengubah penyajian string angka ke dalam nilai numerik dan sebaliknya. Tabel dibawah menunjukan fungsi-fungsi tersebut:

Tabel 8.2. Fungsi Konversi String

FUNGSI	PENJELASAN
atoi	Dalam string sebagai argument. Fungsi untuk mengubah String menjadi integer dan mengembalikan nilai. Contoh Penggunaan: num = atoi("4569");
atol	Dalam bahasa c, string sebagai argument. Fungsi untuk mengubah String menjadi long integer dan mengembalikan nilai.

	Contoh Penggunaan*:* lnum = atol("500000");
atof	Menerima C-string sebagai argumen. Fungsi mengubah string ke dua yang kembali dan nilai. Gunakan fungsi ini mengkonversi string ke float atau double. Contoh Penggunaan: fnum = atof ("3,14159")
itoa	Mengkonversi sebuah integer ke string. Argumen pertama, nilai, adalah integer. Hasilnya akan disimpan di lokasi yang oleh kedua argumen, string. Argumen yang ketiga, basis, merupakan integer. It menentukan penomoran system yang dikonversi integer yang harus dinyatakan dalam (8 = oktal, desimal = 10, 16 = heksadesimal, dll). Contoh Penggunaan: itoa (nilai, string, basis);

Fungsi atoi mengkonversi string menjadi bilangan integer. Hal tersebut menerima argumentasi string dan kebalika nilai bilangan integer yang dikonversi. Di bawah ini adalah sebuah contoh dari bagai-mana cara melakukan konversi:

```
int num;
num = atoi("1000");
```

Dalam statemen ini, atoi mengkonversi string "1000" ke dalam bilangan integer 1000. Begitu variabel num diberikan nilai tersebut,maka dapat digunakan dalam operasi matematik atau setiap opersi yang memerlukan suatu nilai numerik.

Fungsi atol bekerja seperti halnya atoi, kecuali pada nilai dengan tipe data long integer. Perhatikan contoh dibawah ini:

```
long bigNum;
bigNum = atol("500000");
```

sebagaimana yang diinginkan, fungsi atof menerima sebuah argumentasi string dan melakukan konversi string menjadi double. Sebuah nilai numerik double akan dikembalikan, seperti ditunjukkan pada program dibawah ini:

```
double fnum;
fnum = atof("12.67");
```

Fungsi itoa adalah serupa dengan atoi, tetapi itoa bekerja kebalikannya. Itoa akan mengkonversi suatu bilangan integer ke dalam penyajian string yang merepresentasikan bilangan integer. Fungsi itoa menerima tiga argumentasi: yaitu nilai bilangan integer untuk dikonversi, suatu penunjuk pointer menjadi lokasi memori dimana string tersebut disimpan, dan sebuah nomor merepresentasikan dasar-dasar sebuah nilai yang dikonversi. Di bawah ini adalah satu satu contoh instruksinya:

```
char numArray[10];
itoa(1200, numArray, 10);
cout << numArray << endl;
```

Sebuah potongan program tersebut diatas akan mengkonversi bilangan integer 1200 menjadi string.

String tersebut kemudian disimpan di dalam sebuah array dengan nama numArray. Argumentasi yang ketiga, 10, berarti nomor harus ditulis dalam sistim desimal, atau notasi dasar 10. Keluaran dari statemen cout diatas adalah:

```
1200
```

Sekarang perhatikan program dibawah ini, yang menggunakan sebuah fungsi konversi string-to-number, atoi. Pada program meminta pengguna untuk memasukan sebuah rangkaian nilai-nilai, atau surat Q atau q untuk berhenti. Rata-rata dari angka-angka tersebut kemudian dihitung dan ditampilkan.

Program 8.10

```
#include<conio.h>
#include <iostream>
#include <cstring>        // untuk strcmp
#include <cstdlib>        // untuk atoi

using namespace std;

int main()
{
 const int LENGTH = 20;
 char input[LENGTH];
 int total = 0, count = 0;
 double average;

 cout << "Program ini akan mencari rata-rata bilangan.\n";
 cout << "Masukan bilangan pertama atau tekan Q untuk keluar: ";
 cin.getline(input, LENGTH);

 while ((strcmp(input, "Q") != 0)&&(strcmp(input, "q") != 0))
 {
   total += atoi(input);
   count++;
   cout << "Masukan bilangan selanjutnya atau tekan Q untuk keluar: ";
   cin.getline(input, LENGTH);
 }

 if (count != 0)
 {
   average = double(total)/count;
   cout << "Reratanya adalah: " << average << endl;
  }
```

```
 getch();
 return 0;
}
```

Keluaran program, diatas adalah:

```
Program ini akan mencari rata-rata bilangan.
Masukan bilangan pertama atau tekan Q untuk keluar: 45
Masukan bilangan selanjutnya atau tekan Q untuk keluar: 67
Masukan bilangan selanjutnya atau tekan Q untuk keluar: 65
Masukan bilangan selanjutnya atau tekan Q untuk keluar: 45
Masukan bilangan selanjutnya atau tekan Q untuk keluar: 43
Masukan bilangan selanjutnya atau tekan Q untuk keluar: 23
Masukan bilangan selanjutnya atau tekan Q untuk keluar: 23
Masukan bilangan selanjutnya atau tekan Q untuk keluar: 45
Masukan bilangan selanjutnya atau tekan Q untuk keluar: 67
Masukan bilangan selanjutnya atau tekan Q untuk keluar: q
Reratanya adalah: 47
```

Pemanggilan fungsi strcmp untuk membandingkan dua string. Jika kedua string serupa, maka nilai akan diisi dengan 0. sebaliknya jika tidak maka nilai yang diisikan bukan nol. Sebagaimna statemen while, strcmp digunakan untuk menentukan, jika string masukannya menggunakan salah satu "Q" atau "q".

```
while ((strcmp(input, "Q") != 0)&&(strcmp(input, "q") != 0))
```

Jika user tersebut belum memasukkan karakter "Q" atau "q" maka program akan menggunakan atoi untuk mengkonversi string kedalam pada masukan menjadi bilangan integer dan menambahkan nilai total secara keseluruhan sebagaimana dalam insruksi berikut ini:

```
total += atoi(input); // Keep a running total
```

user kemudian akan meminta nomor berikut. Ketika semua angka-angka telah dimasukkan, user mengakhiri pengulangan dengan memasukan karakter "Q" atau "q". Jika satu atau lebih angka-angka tersebut telah dimasukkan, maka perhitungan rata-rata angka-angka tersebut akan ditampilkan.

8.8. Menguji sebuah Karakter

Library bahasa C++ menyediakan beberapa fungsi yang memperbolehkan seorang programer untuk menguji nilai dari sebuah karakter. Fungsi tersebut menguji sebuah argumentasi int tunggal dan hasil pengujian tersebut berupa benar atau salah. Sebuah bilangan

integer adalah Kode ASCII dari suatu karakter. Sebagai contoh pada segmen program berikut dibawah menggunakan fungsi **isupper 86** untuk menentukan jika karakter lolos dari sebagai huruf besar sebagai argumentasi yang diinginkan. Jika hal tersebut sesuai maka fungsi akan benar. Jika tidak, maka hasilnya salah.

```
char letter = 'a';
if (isupper(letter))
      cout << " huruf uppercase.\n";
else
      cout << " huruf bukan uppercase.\n";
```

Pada contoh program diatas menggunakan variabel huruf yang berisi karakter huruf kecil, maka isupper akan salah. Sehingga statemen akan menyebabkan pesan yang muncul "huruf bukan uppercase". Tabel dibawah merupakan daftar beberapa fungsi untuk menguji karakter. Tiap-tiap fungsi diterapkan dalam file header cctype, maka file header harus dipastikan tertulis dalam program ketika menggunakan fungsi tersebut.

Tabel 8.3. Fungsi Pengujian Karakter

FUNGSI KARAKTER	PENJELASAN
isalpha	Returns betul (angka bukan angka nol) jika argumen adalah huruf alfabet. Returns salah jika argumen bukan huruf.
isalnum	Return betul (angka bukan angka nol) jika argumen berupa huruf atau angka. Sebaliknya return adalah salah.
isdigit	Returns betul (angka bukan angka nol) jika argumen adalah angka 0 sampai 9. Sebaliknya selain itu salah.
islower	Return betul (angka bukan angka nol) jika argumen berupa huruf kecil. Sebaliknya jika tidak maka salah.
isprint	Return betul (angka bukan angka nol) jika argumen berupa karakter yang dapat dicetak (termasuk spasi). jika tidak maka Return salah.
ispunct	Returns betul (angka bukan angka nol) jika argumen yang dicetak adalah karakter selain angka, huruf, atau ruang. Selain itu salah.
isupper	Returns betul (angka bukan angka nol) jika argumen adalah huruf besar. Selain itu salah
isspace	Return betul (angka bukan angka nol) jika argumen berupa karakter spasi, selain itu salah. Karakter spasi adalah salah satu dari karakter berikut ini: spasi ' ' vertikal tab '\v' line baru '\n' tab '\t'

8.9. Deskripsi Fungsi Karakter

Program dibawah menggunakan beberapa fungsi seperti pada tabel diatas. Hal tersebut akan minta pengguna untuk memasukan sebuah karakter dan selanjutnya tampilkan sebagai pesan, tergantung pada hasil dari tiap fungsi yang digunakan.

Program 8.11

```
#include<conio.h>
#include <iostream>
#include <cctype>

using namespace std;

int main()
{
    char input;

    cout << "Masukan karakter bebas: ";
    cin.get(input);
    cout << "karakter yang dimasukan adalah: " << input << endl;
    cout << "Kode ASCIInya adalah: " << int(input) << endl;
    if (isalpha(input))
      cout << "Itu adalah karater alfabet.\n";
    if (isdigit(input))
      cout << "Itu adalah digit numerik.\n";
    if (islower(input))
      cout << "Huruf yang anda masukan adalah huruf kecil.\n";
    if (isupper(input))
      cout << "Huruf yang anda masukan adalah huruf besar.\n";
    if (isspace(input))
      cout << "itu adalah karakter spasi.\n";
    getch();
    return 0;
}
```

Keluaran Program adalah sebagai berikut:

```
Masukan karakter bebas: g[Enter]
karakter yang dimasukan adalah: g
Kode ASCIInya adalah: 103
Itu adalah karater alfabet.
Huruf yang anda masukan adalah huruf kecil..
```

Program dibawah akan menampilkan sebuah aplikasi praktis mengenai fungsi uji karakter. Program tersebut menguji tujuh karakter nomor pelanggan untuk menentukan apakah sesuai dengan format atau bentuk yang tepat.

Program 8.12

```
#include <conio.h>
#include <iostream>
#include <cctype>

using namespace std;

bool testNum(char []);

const int NUM_LENGTH = 8;
const int ALPHA_LENGTH = 3;
int main()
{

 char customer[NUM_LENGTH];

 cout << "masukan nomer pelanggan dalam Form ";
 cout << "LLLNNNN\n";
 cout << "(LLL = huruf dan NNNN = angka): ";
 cin.getline(customer, NUM_LENGTH);
 if (testNum(customer))
   cout << "Nomer pelanggan yang dimasukan valid\n";
 else
  {
   cout << "hal tersebut tidak sesuai dengan format";
   cout << "angka pelanggan.\n contohnya adalah seperti dibawah ini:\n";
   cout << " ABC1234\n";
  }
 getch();
 return 0;
}

// Mendefinisikan Fungsi TestNumb.
bool testNum(char custNum[])
{
  for (int count = 0; count < ALPHA_LENGTH; count++)
   {
    if (!isalpha(custNum[count]))
```

```
    return false;
   }
  for (int count = ALPHA_LENGTH; count < NUM_LENGTH - 1; count++)
   {
    if (!isdigit(custNum[count]))
    return false;
   }
  return true;
}
```

Keluaran program diatas adalah sebagai berikut:

```
masukan nomer pelanggan dalam Form LLLNNNN
(LLL = huruf dan NNNN = angka): ABC1235
Nomer pelanggan yang dimasukan valid
```

Keluaran program diatas jika yang dimasukan salah adalah sebagai berikut:

```
masukan nomer pelanggan dalam Form LLLNNNN
(LLL = huruf dan NNNN = angka): sasfewfrw
hal tersebut tidak sesuai dengan formatangka pelanggan.
 contohnya adalah seperti dibawah ini:
 ABC1234
```

Dalam program diatas, nomor pelanggan diharapkan terdiri dari huruf alfabet yang diikuti oleh digit-digit. Kemudian fungsi **testNum** menerima sebuah array dan menguji karakter awal sesuai dengan pengulangan berikut ini:

```
for (count = 0; count < ALPHA_LEN;
count++)
{
        if (!isalpha(custNum[count]))
        return false;
}
```

Fungsi isalpha untuk mengembalikan kondisi benar jika argumentasi adalah satu karakter alfabet. operator ! digunakan didalamnya, jika statemen tersebut untuk menentukan apakah karakter yang diuji bukan berupa alfabet. Jika hal ini terjadi pada beberapa dari tiga yang karakter pertama, fungsi testNum akan salah. Demikian juga, karakter keempat berikutnya diuji dengan pengulangan berikut ini:

```
for (count = ALPHA_LEN; count <
NUM_LENGTH - 1; count++)
{
        if (!isdigit(custNum[count]))
        return false;
}
```

Fungsi isdigit akan bernilai benar jika argumentasi merepresentasikan karakter tentang semua digit 0 sampai 9. Operator ! digunakan untuk menentukan jika karakter yang diuji bukan sebuah digit. Ini terjadi biasanya pada empat karakter

terakhir, fungsi testNum akan bernilai salah.

Jika nomor pelanggan mengggunakan bentuk yang tepat, fungsi tersebut akan melalui siklus kedua pengulangan tanpa ada hasil yang nilai salah. Di kasus tersebut, garis terakhir dalam fungsi tersebut adalah statemen dengan hasil yang benar, dimana hal tersebut menandai bahwa nomor pelanggan adalah valid.

8.10. Konversi Karakter

Library Bahasa C++ menyediakan dua fungsi, toupper dan tolower. Untuk mengubah kasus dari sebuah karakter. Fungsi tersebut digambarkan pada Tabel dibawah ini. Fungsi ini merupakan prototype dalam file header cctype, maka pastikan untuk memasukkan dua fungsi didalamnya.

Tabel 8.4. Fungsi Pengubah Karakter

FUNGSI	DEFINISI
toupper	Mengembalikan huruf besar setara dengan argumen.
tolower	Mengembalikan lowercase setara dengan argumen

Keduanya dari fungsi pada tabel diatas akan menerima sebuah representasi bilangan integer kode ASCII dari suatu karakter yang dikonversi dan mengembalikan representasikan bilangan integer Kode ASCII dari huruf besar atau huruf kecil setara. Tiap fungsi hanya mengembalikan argumentasinya, jika konversi tidak bisa dibuat atau tak diperlukan. Sebagai contoh, jika argumentasi tersebut pada toupper bukan surat huruf kecil, lalu toupper hanya kembalikan argumentasi yang ada tanpa mengubahnya. Prototipe fungsi-fungsi tersebut adalah:

```
int toupper(int ch);
int tolower(int ch);
```

kenyataannya fungsi dua veriabel ini mengkembalikan pada sebuah bilangan integer pernyataan sebagai berikut:

```
cout << toupper('a');  // prints 65
```

pada potongan program diatas akan mencetak bilangan integer Kode ASCII dari 'A'. Untuk mendapatkan cetakan karakter, maka dapat memasukan hasil kenilai tipe data char, seperti dalam potongan program dibawah ini:

```
cout << static_cast<char>(toupper('a'));
// prints A
```

atau dapat juga dengan memberi hasil ke nilai sebuah variabel karakter pertama dan kemudian karakter tersebut dicetak:

```
char ch = toupper('a');
```

```
cout << ch; // prints A
```

Sebagaimana telah disebutkan diatas, fungsi toupper akan mengembalikanya tanpa perubahan, jika hal tersebut buka berupa huruf:

```
cout << static_cast<char>(toupper('&'));
// prints &
```

atau suatu huruf tetapi itu sudah merupakan huruf besar:

```
cout << static_cast<char>(toupper('B');
// prints B
```

Karena uraian mengenai toupper dan tolower dilewati oleh nilai, fungsi tersebut tidak akan mengubah parameter yang mereka miliki: Sebagai gantinya, mereka hanya mengembalikan ekivalensi huruf besar atau huruf kecil. Sebagai contoh, di dalam sebuah segmen program yang berikut, variabel huruf diatur menjadi nilai 'A'. Fungsi tolower untuk mengembalikan karakter 'a', ketika huruf masih berisi 'A'.

```
char letter = ‘A’;
char ch = tolower(letter);
cout << ch << endl;
cout << letter;
```

potongan program diatas kalau di compile menampilkan hasil sebagai berikut:

```
a
A
```

Program berikut dibawah untuk mendemontrasikan fungsi toupper dalam sebuah pengulangan sampai ada kemungkian user memasukan salah satu huruf Y atau N.

Program 8.13

```
#include <conio.h>
#include <iostream>
 #include <cctype>
 #include <iomanip>

 using namespace std;

 int main()
 {
  const double PI = 3.14159;
  double radius;
  char go;

  cout << "Program untuk menghitung luas lingkaran\n";
  cout << setprecision(2);
  cout << fixed;
  do
   {
```

```
    cout << "Masukan jari-jari lingkaran: ";
    cin >> radius;
    cout << "Luas lingkaran adalah " << (PI * radius * radius);
    cout << endl;
    do
     {
        cout << "Akan menghitung lagi? (Y or N) ";
        cin >> go;
     } while (toupper(go) != 'Y' && toupper(go) != 'N');
  } while (toupper(go) == 'Y');
getch();
return 0;
}
```

Program keluaran program diatas adalah berikut

```
Program untuk menghitung luas lingkaran
Masukan jari-jari lingkaran: 77
Luas lingkaran adalah 18626.49
Akan menghitung lagi? (Y or N) y

Masukan jari-jari lingkaran: 23
Luas lingkaran adalah 1661.90
Akan menghitung lagi? (Y or N) n
```

8.11. Menulis string

Setelah mampu melewati materi mengenai array, maka anda dapat menulis fungsi secara sendiri untuk memproses sebuah string. Sebagai contoh, program dibawah ini adalah menggunakan sebuah fungsi untuk melakukan copy atau menyalin string dari satu array ke bentuk yang lain.

Program 8.14

```
#include <conio.h>
#include <iostream>

using namespace std;

void stringCopy(char [], char []);

int main()
 {
   const int S_LENGTH = 30;
```

```
   char first[S_LENGTH], second[S_LENGTH];

   cout << "masukan string dengan tidak lebih dari "
   << S_LENGTH - 1 << " karakter:\n";
   cin.getline(first, S_LENGTH);
   stringCopy(first, second);
   cout << "String yang anda masukan adalah:\n" << second << endl;
   getch();
   return 0;
}

// Definisi Fungsi stringCopy *
void stringCopy(char string1[], char string2[])
{
    int index = 0;

    while (string1[index] != '\0')
    {
        string2[index] = string1[index];
        index++;
    }
    string2[index] = '\0';
}
```

Keluaran program diatas adala sebagai berikut:

```
masukan string dengan tidak lebih dari 29 karakter:
hello saya sedang belajar
String yang anda masukan adalah:
hello saya sedang belajar
```

Yang perlu diperhatikan pada program diatas adalah pada fungsi stringCopy yang tidak menerima satu bentuk array. Program tersebut hanya melakukan copy karakter-karakter dari string1 ke dalam string2 sampai menemui sebuah hal yang dapat mengakhiri atau tanda null dalam string1.

Ketika sebuah tanda yang mengakhiri tersebut ditemukan dan pengulangan sudah mencapai ujung dri string, statemen terakhir dalam fungsi memberi suatu tanda batal (karakter '\0') hingga selesai string2, dengan demikian hal tersebut dapat diakhiri dengan baik.

Program dibawah menggunakan handling function (fungsi untuk menangani) string: nameSlice. Program akan minta pengguna itu untuk memasukan nama awal dan akhir pengguna, yang dipisahkann oleh spasi. Kemudia fungsi mencari string untuk spasi dan mengganti

dengan suatu terminator akhir. Perhatikan program dibawah ini:

Program 8.15

```
#include<conio.h>
#include <iostream>

using namespace std;

void potogannama(char []);

int main()
{
   const int PANJANG_NAMA = 41;
   char name[PANJANG_NAMA];

   cout << "Masukan nama depan dan nama akhir Anda dengan dipisahkan";
   cout << "dengan spasi:\n";
   cin.getline(name, PANJANG_NAMA);
   potogannama(name);
   cout << "Nama pertama anda adalah: " << name << endl;
   getch();
   return 0;
}

// Definisi Fungsi potogan nama.
void potogannama(char userName[])
{
    int count = 0;
    while (userName[count] != ' ' && userName[count] != '\0')
       count++;
    if (userName[count] == ' ')
       userName[count] = '\0';
}
```

Keluaran program diatas adalah sebagai berikut:

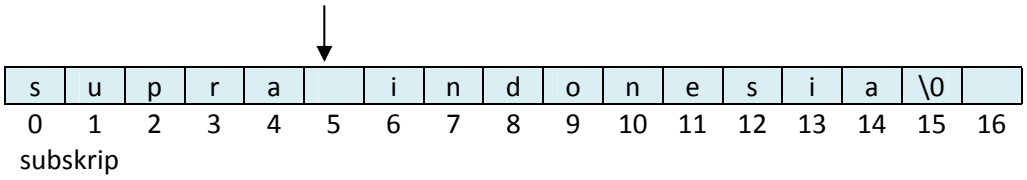
```
Masukan nama depan dan nama akhir Anda dengan dipisahkandengan spasi:
Supra indonesia
Nama depan anda adalah: supra
```

Pengulangan pada variabel potongannama dimulai saat karakter yang pertama dalam array dan melakukan scan string, kemudian mencari baik sebuah spasi atau terminator null:

```
while (userName[count] != ' ' &&
userName[count] != '\0')
count++;
```

Jika karakter dalam userName[count] bukanlah sebuah spasi atau terminator null, count akan ditambah dengan karakter berikutnya yang akan diuji. Dengan contoh memasukan string nama depan dan nama belakang supra indonesia", pada program pengulangan menemukan pemisah spasi "suprapto" dan "indonesia", pada userName[5]. Ketika pengulangan berhenti, count diatur sampai 5. Akan lebih jelasnya jika memperhatikan gambar dibawah ini:

Perulangan akan berhenti ketika menemukan tanda spasi atau userName[5]

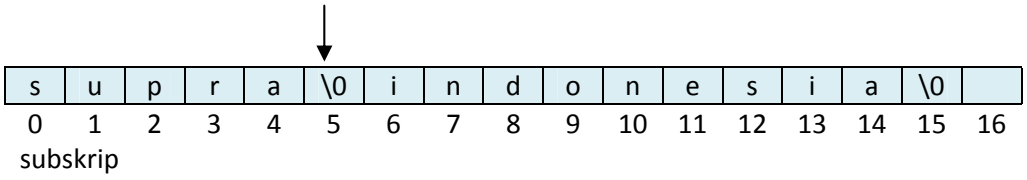


Begitu pengulangan sudah diselesaikan, userName[count] akan berisi salah satu spasi atau terminator null. Jika berisi spasi, dan diikuti tanda if statemen kemudian menggantikannya dengan terminator null:

```
if (userName[count] == ' ')
userName[count] = '\0';
```

sehingga dapat diilustrasikan seperti pada gambar dibawah ini:

Ruangan diberi \0 atau tanda sebagai string selesai



8.12. Pointer untuk menguraikan String

Penunjuk pointer sangat bermanfaat untuk menulis sebuah fungsi proses pada string. Jika alamat awal sebuah string berubah menjadi sebuah variabel parameter penunjuk pointer, maka akan dapat diasumsikan bahwa semua karakter dari alamat tertinggi akan menahan tanda null menjadi bagian dari string. Program dibawah menunjukkan sebuah fungsi, countChars, yang menggunakan penunjuk pointer untuk

menghitung jumlah karakter khusus muncul didalam string.

Program 8.16

```
#include<conio.h>
#include <iostream>

using namespace std;

// Fungsi prototipe
int countChars(char *, char);

int main()
{
        const int PANJANG_S = 51;
        char userString[PANJANG_S], letter;

        cout << "Masukan String (lebih dari " << PANJANG_S-1 << " karakter): ";
        cin.getline(userString, PANJANG_S);
        cout << "Masukan karakter dan saya akan menghitung berapa\n";
        cout << "kali yang muncul dalam string yang telah anda tulis: ";
        cin >> letter;
        cout << letter << " Muncul: ";
        cout << countChars(userString, letter) << " kali.\n";
        getch ();
        return 0;
}

// Definisi countChars.
int countChars(char *strPtr, char ch)
{
   int times = 0;
   while (*strPtr != '\0')
    {
      if (*strPtr == ch)
        times++;
      strPtr++;
    }
   return times;
}
```

Keluaran program diatas adalah sebagai berikut:

```
Masukan String (lebih dari 50 karakter): saya akan pergi berangkat ke kantor
Masukan karakter dan saya akan menghitung berapa
kali yang muncul dalam string yang telah anda tulis: a
a Muncul: 7 kali.
```

Di dalam fungsi countChars, strPtr poin pada string tersebut akan dicari dan ch berisi karakter untuk mencari. Pengulangan while akan di ulangi ketika panjang karakter strPtr poin pada akhir bukan terminator null:

```
while (*strPtr != '\0')
```

Dalam pengulangan, statemen if yang mengikutinya membandingkan karakter strPtr poin dengan karakter didalam ch:

```
if (*strPtr == ch)
```

Jika keduanya sama, variabel times akan diincrement. Statemen yang terakhir dalam pengulangan tersebut adalah:

```
strPtr++;
```

statemen increment menaikan alamat dalam strPtr. Ini menyebabkan strPtr akan menunjuk karakter berikutnya didalam sebuah string. Kemudian pengulangan memulai kembali. Ketika strPtr akhirnya menjangkau terminator null, pengulangan mengakhiri dan fungsi mengembalikan nilai dalam fungsi time.

8.13. Class String pada C++

Dari satu segi pandangan metode pemrograman yang mudah, library standar class string menawarkan beberapa keuntungan pada string. Ketika anda sudah mengetahui dalam bab ini, class string mempunyai beberapa anggota fungsi dan operator yang mempunyai muatan lebih. Hal ini jelas menyederhanakan tugas, seperti penempatan sebuah karakter atau string di dalam suatu string, yang jika dilakukan tanpa fungsi akan mengalami kesulitan dan membosankan untuk melakukan operasi string.

Setiap program class string harus menggunakan #include sebagai file header string. Object string dapat diciptakan dengan beberapa construktor. Program dibawah menunjukkan dua construktor class string: sebagai default construktor dan konversi konstruktor string ke menjdai obyek string.

Program 8.17

```
#include<conio.h>
 #include <iostream>
 #include <string>

 using namespace std;
```

```
int main()
{
   string ucapan;
   string nama("suprapto");

   ucapan = "Halo ";

   cout << ucapan << nama << endl;
   getch();
   return 0;
}
```

Keluaran program diatas adalah sebagai berikut:

```
Halo suprapto
```

Supaya dapat membuah program selain contoh program diatas, perhatikan tabel pemakaian konsruktor string yang diberikan dalam tabel dibawah ini:

Tabel 8.5. Pemakaian Konsruktor String

DEFINITION	DESCRIPTION
string alamat;	Definisi objek string yang langsung disebutkan
string nama("suprapto");	Mendefinisikan string benda bernama nama, diinisialisasi dengan "suprapto"
string person1(person2);	Mendefinisikan string benda bernama person1, yang merupakan salinan dari person2. person2 mungkin salah satu objek atau string karakter array.
string set1(set2, 5);	Mendefinisikan string bernama set1 objek yang diinisialisasi untuk pertama lima karakter dalam karakter array set2.
string lineFull('z', 10);	Mendefinisikan string benda bernama lineFull diinisialisasi dengan 10 'z' karakter.
string firstName(fullName, 0, 7);	Mendefinisikan string benda bernama firstname, diinisialisasi dengan substring dari string fullName. Substring adalah tujuh karakter, mulai pada posisi 0.

Hal yang perlu diketahi pada program dibawah, menggunakan operator = untuk memberi nilai obyek string. class string memberi muatan lebih pada beberapa operator, yang digambarkan pada Tabel dibawah.

Tabel 8.6. Operator String

OVERLOADED	PENJELASAN OPERATOR
>>	Ekstrak karakter stream dan masukkan karakter ke dalam string. Karakter akan dicopy sampai menemui spasi atau akhir kalimat dan sekaligus menhitung karakter masukan.
<<	Menyisipkan string ke dalam stream.
=	Memasukan string pada sebelah kanan ke obyek string pada sebelah kiri.
+=	Menambahkan copyan string pada string sebelah kanan ke obyek string pada sebelah kiri.
+	Mengembalikan string yang merupakan rangkaian dari dua operand string
[]	Melaksanakan tanda notasi array, seperti nama [x]. dimana hal tersebut merujuk pada karakter pada pengembalian posisi x.
Relational Operators	Masing-masing operator penghubung diimplementasikan: < > <= >= == !=

Program dibawah digunakan untuk mendemontrasikan sebagian dari operator string

Program 8.18

```
#include <iostream>
#include <string>

using namespace std;

int main()
{
        string str1, str2, str3;
        str1 = "ABC";
        str2 = "DEF";
        str3 = str1 + str2;

        cout << str1 << endl;
        cout << str2 << endl;
        cout << str3 << endl;
        str3 += "GHI";
        cout << str3 << endl;
        return 0;
}
```

Keluaran program tersebut diatas adalah sebagai berikut:

```
ABC
DEF
ABCDEF
ABCDEFGHI
```

Class string juga mempunyai beberapa fungsi anggota. Sebagai contoh, fungsi size mengembalikan panjang string. Hal tersebut didemontrasikan dalam pengulangan For pada program dibawah ini:

Program 8.19

```
#include<conio.h>
 #include <iostream>
 #include <string>

 using namespace std;

 int main()
 {
    string str1, str2, str3;
    str1 = "ABC";
    str2 = "DEF";
    str3 = str1 + str2;

    for (int x = 0; x < str3.size(); x++)
       cout << str3[x];
    cout << endl;

    if (str1 < str2)
       cout << "str1 adalah lebih kecil daripada str2\n";
    else
       cout << "str1 tidak lebih kecil daripada str2\n";
    getch();
    return 0;
 }
```

Keluaran program diatas adalah sebagai berikut:

```
ABCDEF
str1 adalah lebih kecil daripada str2
```

Tabel dibawah ini merupakan daftar dari banyak string yang merupakan anggota class string variasi yang dibebankan.

Tabel 8.7. Class String Keanggotaan Fungsi

ANGGOTA FUNGSI	PENJELASAN
theString.append(str);	Menambahkan str ke theString. str bisa menjadi objek atau string karakter array.
theString.append(str, x, n);	n jumlah karakter dari str, dimulai pada posisi x, akan ditambahkan ke theString. jika TheString terlalu kecil, fungsi akan melakukan copy sejumlah karakter yang mungkin bisa dicopy.

theString.append(str, n);	N karakter pertama dari array karakter str yang ditambahkan ke theString.
theString.append(n, 'z');	N menambahkan salinan 'z' untuk theString.
theString.assign(str);	Memberikan str ke theString. Parameter str dapat menjadi obyek string atau C-string.
theString.assign(str, x, n);	jumlah n karakter dari str, dimulai pada posisi x, ditugaskan ke theString. jika TheString terlalu kecil, fungsi akan menyalin banyak karakter mungkin.
theString.assign(str, n);	karakter pertama n dari array karakter str ditugaskan ke theString.
theString.assign(n, 'z');	Memberikan salinan n 'z' menuju theString.
theString.at(x);	Mengembalikan karakter pada posisi x dalam string.
theString.begin();	Kembali pada sebuah iterator yang menunjuk ke karakter pertama pada string.
theString.capacity();	Mengembalikan ukuran penyimpanan dialokasikan untuk string.
theString.clear();	Membersihkan string dengan menghapus semua karakter yang disimpan di dalamnya.
theString.compare(str);	Melakukan perbandingan seperti fungsi strcmp dengan nilai yang sama kembali. str bisa menjadi string objek atau karakter array.
theString.compare(x, n, str);	Membandingkan theString dan str, mulai dari posisi x, dan dilanjutkan terus untuk n karakter. pengembalian nilai seperti strcmp. str bisa menjadi objek string atau karakter array.
theString.copy(str, x, n);	menyalin karakter str ke array theString, mulai pada posisi x, untuk n karakter. TheString jika terlalu kecil, maka fungsi akan menyalin karakter semampu mungkin.
theString.c_str():	Mengembalikan nilai C-string string objek.
theString.data();	Kembali karakter array yang berisi null string dihentikan, karena disimpan dalam theString.
theString.empty();	Returns true jika theString kosong. Returns true if theString is empty.
theString.end();	Kembali ke sebuah iterator yang terakhir karakter dari string di theString. (Untuk informasi lebih lanjut tentang iterators, lihat Bab 15). Returns an iterator pointing to the last character of the string in theString. (For more information on iterators, see Chapter 15.)
theString.erase(x, n);	Erases n karakter dari theString, mulai di posisi x.

	Erases n characters from theString, beginning at position x.
theString.find(str, x);	Kembali posisi pertama di luar atau di mana posisi x string str ditemukan di theString. Parameter str mungkin salah satu objek string atau C-string. Str jika tidak ditemukan, posisi diluar akhir theString dikembalikan. Returns the first position at or beyond position x where the string str is found in theString. The parameter str may be either a string object or a C-string. If str is not found, a position beyond the end of theString is returned.
theString.find('z', x);	Kembali posisi pertama di luar atau di mana posisi x 'z' yang ditemukan di theString. Returns the first position at or beyond position x where 'z' is found in theString.
theString.insert(x, str);	Menyisipkan salinan str ke theString, mulai di posisi x. str mungkin salah satu objek atau string karakter array. Inserts a copy of str into theString, beginning at position x. str may be either a string object or a character array.
theString.insert(x, n, 'z');	Menyisipkan 'z' n kali menuju theString di posisi x.
theString.length();	Pernyataan panjang string di theString. Returns the length of the string in theString.
theString.replace(x, n, str);	Menggantikan n karakter di awal theString di posisi x dengan karakter dalam string str objek. Replaces the n characters in theString beginning at position x with the characters in string object str.
theString.resize(n, 'z');	Perubahan besarnya alokasi di theString ke n. N kurang jika dibandingkan dengan ukuran string, string yang dipotong untuk n karakter. Jika n lebih besar, string yang diperluas dan 'z' yang ditambahkan pada akhir waktu yang cukup untuk mengisi ruang-ruang baru. Changes the size of the allocation in theString to n. If n

	is less than the current size of the string, the string is truncated to n characters. If n is greater, the string is expanded and 'z' is appended at the end enough times to fill the new spaces.
theString.size();	Pernyataan panjang string pada theString.
theString.substr(x, n);	Kembali melakukan salinan suatu substring. The substring adalah n karakter dan mulai pada posisi x dari theString.
theString.swap(str);	Swaps isi dengan theString str.

8.14. Membuat Class String Sendiri

Class string pada bahasa C++ secara otomatis menangani banyak tugas-tugas yang terlibat dalam penggunaan string, seperti alokasi memori dinamis dan memeriksa margin dan lain sebagainya. Ia juga merupakann operator overload seperti + dan =, dan juga memiliki banyak anggota fungsi yang memudahkan dalam bekerja dengan string. Dalam beberapa bagian, kita harus membuat sebuah data tipe string dengan banyak kelas fungsi C++.

Dalam proses, kita lihat misalnya pada contoh copy constructor keberatan dan operator penuh aksi, serta contoh-contoh teknik pemrograman yang berguna sebagai solusi dari berbaai masalah. Kelas MyString yang ditetapkan dalam bagian ini adalah jenis data abstrak untuk menangani string. Memiliki banyak keunggulan yang dimiliki oleh kelas C++ string yang diberikan oleh Standard Template Library:

- Memori yang dteralokasi secara dinamis untuk setiap string yang tersimpan dalam MyString objek. Programmer yang menggunakan kelas ini tidak perlu khawatir dengan seberapa besar untuk membuat array.
- Strings mungkin akan ditugaskan ke objek dengan MyString = operator. Programmer yang menggunakan kelas ini tidak perlu memanggil fungsi strcpy.
- Satu string mungkin concatenated lain dengan + = operator. Ini menghilangkan kebutuhan untuk fungsi strcat.
- Strings mungkin akan diuji untuk kesetaraan dengan == operator. Programmer yang menggunakan kelas ini tidak perlu memanggil fungsi strcmp.

Pada sebuah Kelas MyString memiliki pointer sebagai anggota dan secara dinamis mengalokasikan memori untuk menyimpan nilai string, konstruksi salinan disediakan. Fungsi ini akan menyebabkan objek diatur dengan benar datanya ketika diinisialisasi dengan obyek MyString lain.

Kelas MyString memiliki 2 overloaded = operator. Yang pertama adalah untuk menempatkan satu MyString ke objek lain. fungsi Operator ini dipanggil ketika operand di sebelah kanan dari tanda = adalah

MyString objek, seperti yang terlihat pada segmen kode berikut:

```
MyString first("Hello"), second;
second = first;
```

Kedua versi MyString = dari operator untuk menempatkan tradisional string ke MyString objek. Operator fungsi ini dipanggil ketika operand di kanan = adalah sebuah string konstan atau pointer ke sebuah string (seperti nama sebuah char array). Hal ini ditunjukkan pada segmen program berikut:

```
MyString name;
char who[] = "Jimmy";
name = who;
```

Operator + = dirancang untuk menggabungkan string pada MyString kanan ke kiri pada objek. Seperti operator =, MyString memiliki dua versi =. + Versi pertama ini dirancang untuk bekerja pada saat yang tepat operand adalah MyString obyek lain, seperti yang ditunjukkan dalam program ini segmen:

```
MyString first("Hello "), second("world");
first += second;
```

Kedua versi dari operator + = akan dipanggil ketika operand sebelah kanan adalah string literal atau pointer ke karakter :

```
MyString first("Hello ");
first += "World";
```

Objek MyString yang memiliki kelebihan beban versi dari operator == untuk melakukan ujian kesetaraan. Seperti operator lainnya, versi yang pertama dirancang untuk bekerja dengan objek MyString lainnya dan yang kedua dirancang untuk bekerja dengan string C++ konvensional. == Fungsi akan kembali jika string berada di sebelah kanan operand sesuai dengan anggota str panggilan objek. Jika kedua operands string tidak cocok, fungsi return false. fungsi Operator ini memungkinkan programmer menggunakan kelas ini untuk membangun penghubung ekspresi seperti ini:

```
MyString name1("Supra"),
name2("Supra");
if (name1 == name2)
  cout << "nama adalah sama.\n";
else
   cout << "nama adalah berbeda\n";

MyString name1("Suprapto");
if (name1 == "Supra")
  cout << "nama adalah sama.\n";
else
   cout << "nama adalah berbeda.\n";
```

Objek MyString memiliki dua versi yan berkelebihan beban > lebih besar daripada operator untuk melakukan tes, dan < operator untuk melakukan kurang dari tes. Versi pertama dari masing-masing dirancang untuk bekerja dengan objek lain MyString dan yang kedua ini dirancang untuk bekerja dengan C + + tradisional string. (fungsi perpustakaan menggunakan fungsi strcmp untuk menentukan apakah sebuah lebih daripada atau kurang daripada hubungan yang ada.)

Sebuah fungsi > return benar jika str anggota panggilan objek lebih besar daripada string dalam operand

kanan. Jika tidak, fungsi return false. Yang <fungsi return true jika str anggota panggilan objek kurang dari string dalam operand kanan. Jika tidak, mereka kembali palsu. Operator fungsi ini memungkinkan programmer menggunakan kelas ini untuk membangun penghubung ekspresi seperti yang ditampilkan dalam program ini segmen:

```
MyString name1("Suprapto"),
name2("Supra");
if (name1 > name2)
   cout << "Suprapto lebih besar
            daripada Supra\n";
else
   cout << "Suprapto tidal lebih besar
            daripada Supra.\n";

MyString name3("Suprapto");

if (name3 < "Supra")
    cout << "Suprapto lebih kecil
             daripada Supra.\n";
else
   cout << "Suprapto tidak lebih kecil
            daripada\n";
```

Objek MyString memiliki dua versi > = operator untuk melakukan operasi lebih besar daripada atau tes sama dengan, dan <= operator untuk melakukan kurang dari atau tes kesamaan. Versi pertama dari masing-masing dirancang untuk bekerja dengan objek lain MyString dan yang kedua ini dirancang untuk bekerja dengan C++ tradisional string.

Pada fungsi > = kembali benar, jika str anggota panggilan objek yang lebih besar dari atau sama dengan string yang ada di sebelah kanan operand. Jika tidak, fungsi return salah. Fungsi <= return benar jika str anggota panggilan objek kurang dari atau sama dengan string yang ada di sebelah kanan operand. Jika tidak, mereka kembali palsu. Operator fungsi ini memungkinkan programmer menggunakan kelas ini untuk membangun penghubung ekspresi seperti yang ditampilkan dalam segmen program ini:

```
MyString name1("Suprapto"),
name2("Supra");
if (name1 >= name2)
     cout << "Suprapto adalah lebih besar
              atau sama dengan Supra.\n";
 else
     cout << "Suprapto lebih kecil
              daripada Supra.\n";

MyString name3("Suprapto");
if (name3 <= "Supra")
    cout << "Suprapto lebih kecil atau
             sama dengan Supra.\n";
 else
    cout << "Suprapto lebih besar sama
             dengan Supra.\n";
```

Program dibawah menunjukkan bagaimana program MyString memperlihatkan rangkaian stringnya. Selain itu, kode utama program mendemonstrasikan bagaimana MyString memungkinkan programmer untuk memperlakukan string lebih banyak yang lain seperti mengikutkan tipa data didalamnya.

Program 8.20

```
#include <iostream>
#include "mystring.h"

using namespace std;

int main()
{
   MyString object1("Ini"), object2("adalah");
   MyString object3("menguji.");
   MyString object4 = object1;              // panggil copy constructor.
   MyString object5("adalah hanya menguji.");
   char string1[] = "sebuah pengujian.";

   cout << "Object1: " << object1 << endl;
   cout << "Object2: " << object2 << endl;
   cout << "Object3: " << object3 << endl;
   cout << "Object4: " << object4 << endl;
   cout << "Object5: " << object5 << endl;
   cout << "String1: " << string1 << endl;
   object1 += " ";
   object1 += object2;
   object1 += " ";
   object1 += object3;
   object1 += " ";
   object1 += object4;
   object1 += " ";
   object1 += object5;
   cout << "object1: " << object1 << endl;
   return 0;
}
```

Keluaran Program diatas adalah sebagai berikut

```
Object1: Ini
Object2: adalah
Object3: menguji.
Object4: Ini
Object5: adalah hanya menguji.
String1: sebuah pengujian.
object1: Ini adalah menguji. Ini hanya menguji sebuah pengujian.
```

8.15. Studi Kasus

Sebagai programmer untuk programmer lanjut pada Software Enterprises, kita biasanya akan diminta untuk mengembangkan satu kelas yang bernama mata penyisipan koma dan tanda dolar ($) di lokasi yang sesuai dalam sebuah string yang berisi jumlah dolar tidak diformat. Kelas konstruktor harus menerima string objek atau pointer ke string bahasa C yang berisi nilai seperti 1084567,89. Kelas harus menyediakan fungsi anggota yang mengembalikan obek string menjadi sejumlah formatted-dollar, seperti $ 1.084.567,89.

Tabel 8.8. Keanggotaan Variabel

MEMBER VARIABEL	PENJELASAN
Original	Obyek string menahan string asli yang bersifat unformatted.
formatted	Obyek string menahan string yang bersifat formatted.

Tabel 8.9. Fungsi Keanggotaan

FUNGSI KEANGGOTAAN	PENJELASAN
Constructor	Disertakan dalam obyek string sebagai argumen. Obyek disalin ke original member, dan memanggil fungsi keanggotaan dolarrFormat .
dollarFormat	Menyalin keanggotaan original menjadi format member. Tanda koma dan dolar disertakan dalam memasukkan ke lokasi bentuk keanggotaan.
getOriginal	Mengambalikan keanggotaan original
getFormatted	Mengembalikan bentuk keanggotaan

Program 8.21. Penggunaan class

```
#include <iostream>
#include <string>
#include "currency.h"

using namespace std;

int main()
{
        string input;
        cout << "Masukan jumlah dollar ke rekening dengan form nnnnn.nn : ";
        cin >> input;
        Currency dollars(input);
        cout << "disini rekening diatur:\n";
```

```
        cout << dollars.getFormatted() << endl;
        return 0;
}
```

Keluaran program diatas adalah sebagai berikut**:**

```
Masukan jumlah dollar ke rekening dengan form nnnnn.nn :  : 1084567.89[Enter]
disini rekening diatur:
$1,084,567.89
```

8.16. Soal Latihan

Jawablah soal latihan dibawah ini dengan baik dan benar.

1. Apa yang dimaksud dengan string
2. Tulislah program sederhana dengan string untuk menampilkan tulisan "Hallo saya siswa smk"
3. Buatlah program untuk menggabungkan dua string "hello" dan "apa khabar"
4. Apa perbedaan anatara string dengan pointer
5. Tulislah program yang menggunakan pointer untuk operasi string
6. Sebutkan beberapa fungsi string pada bahasa c++
7. Tulislah program yang digunakan untuk membandingkan dua string
8. Sebutkan fungsi untuk menangani string pada bahasa c++
9. Buatlah program untuk mengkonversi string
10. Apa yang dimaksud dengan fungsi topupper dan tolower
11. Sebutkan beberapa operator string pada bahasa c++

BAB 9
ARRAY

9.1. Pengertian Array

Tipe data array menjelaskan jangkauan nilai yang dapat ditampung pada sebuah variabel dan kumpulan operasi yang dapat dilakukan terhadap variabel tersebut. Dengan pengecualian pada string karakter, semua tipe yang telah dipelajari sampai saat ini hanya dapat menampung sebuah nilai.

Pada saat tertentu program yang dibuat dibutuhkan untuk menyelesaikan suatu permasalahan yang dikehendaki suatu variabel yang dapat menampung banyak nilai. Sebagai contoh variabel skor barangkali mencatat skor ujian dari 100 siswa. Demikian juga variabel gaji mungkin menampung gaji yang berlainan dari masing-masing pegawai suatu perusahaan.

Array adalah struktur data yang dapat menyimpan sejumlah nilai bertipe sama, sebagai contoh kita dapat menciptakan sebuah array yang dapat menampung 100 nilai bertipe int dan array kedua yang dapat menampung 25 nilai bertipe float. Setiap nilai yang ditugaskan ke array harus bertipe sama dengan array tersebut

Array adalah variabel yang mampu menyimpan sejumlah nilai yang bertipe sama. Untuk mendeklarasikan sebuah array, harus disebutkan tipe dari array yang dibuat misalnya int, float atau double dan juga ukuran array. Untuk menentukan ukuran array,perlu ditempatkan jumlah nilai yang dapat disimpan array dalam sebuah tanda kurung kurawal siku yang terletak sesuah nama array.

Deklarasi berikut sebagai contoh untuk menciptakan array bernama skor yang mampu menyimpan 100 skor nilai yang bertipe int.

```
int skor[100];
```

Pada saat dideklarasikan sebuah array, compiler C mengalokasikan memori yang cukup untuk

menampung semua elemen sesuai dengan yang dideklarasikan. Masukan pertama berada pada lokasi 0. Sebagai contoh berdasarkan array skor, pernyataan berikut menugaskan nilai 80 pada elemen pertama dari array

```
skor[0] = 80;
```

Karena elemen pertama dari array dimulai dengan offset 0, maka elemen terakhir dari array adalah satu lokasi sebelum ukuran array. Berdasarkan array skor diatas, pernyataan berikut menugaskan nilai ke elemen terakhir dari array.

```
skor[99]  = 75;
```

Untuk inisialisasi array ada beberapa macam yang sering dilakukan sebagai contoh perhatikan potongan program berikut ini :

```
char title[] = "Dasar Pemrograman";
char section [64] = "Arrays";
```

Pada kasus yang pertama, compiler C akan mengalokasikan 17 byte untuk menampung string. Pada kasus yang kedua, compiler akan mengalokasikan array 64 byte, menginisialisasi enam karakter yang yang pertama dengan huruf-huruf “Arrays” dan karakter NULL. Kebanyakan compiler juga akan menginisialisasi lokasi byte-byte tersisa dengan NULL. Pada saat dideklarasikan array dengan tipe lain, kita juga dapat menginisialisasinya dengan cara yang sama. Sebagai contoh pernyataan berikut menginisialisasi array integer scores dengan nilai 80, 70, 90, 85 dan 80.

```
int scores[5] = {80, 70, 90, 85, 80} ;
```

Pada saat kita menugaskan nilai awal ke suatu array , kita harus melingkupi nilai-nilai tersebut dengan tanda kurung kurawal{}. Dalam hal ini, ukuran array sesuai dengan jumlah nilai yang ditugaskan ke array. Akan tetapi pernyataan berikut akan menugaskan empat nilai titik mengambang ke array yang dapat menyimpan 64 nilai.

```
float salaries[64] ={2500000.0, 3200000.0, 4400000.0, 5500000.0};
```

Tergantung dari compiler yang digunakan, nilai-nilai yang ditugaskan ke elemen-elemen yang tidak secara eksplisit ditugaskan mungkin diisi dengan 0. Sebagai pegangan kita harus tidak menganggap bahwa compiler akan menginisialisasi elemen-elemen yang lain. Jika kita tidak menyebutkan ukuran array, kompiler akan mengalokasikan memori yang hanya cukup untuk menampung nilai-nilai yang disebutkan. Deklarasi array berikut menciptakan array yang cukup menampung tiga buah nilai yang bertipe long.

```
long planets[] = {1234L, 5678L, 12347L};
```

Array adalah suatu variabel yang terdiri dari sekumpulan data dimana data-data tersebut mempunyai tipe data yang sama. Setiap data disimpan dalam alamat memori yang berbeda-beda dan disebut dengan elemen array. Setiap elemen mempunyai nilai indek sesuai dengan urutannya. Melalui indek inilah kita dapat mengakses data-data tersebut.

Indek dari elemen array ini, baik dalam bahasa C++ maupun Java dimulai dari 0, bukan 1 seperti dalam bahasa Pascal.

Dalam beberapa literatur, array sering disebut (diterjemahkan) sebagai larik. Array adalah kumpulan dari nilai-nilai data bertipe sama dalam urutan tertentu yang menggunakan sebuah nama yang sama. Nilai-nilai data di suatu array disebut dengan elemen-elemen array. Letak urutan dari elemen-elemen array ditunjukkan oleh suatu *subscript* atau indeks. Suatu array berdimensi satu dideklarasikan dalam bentuk umum berupa:

```
tipe_data nama_var[ukuran];
```

dimana :

- tipe_data : untuk menyatakan tipe dari elemen array, misalnya *int, char, float*.
- nama_var : nama variabel array
- ukuran : untuk menyatakan jumlah maksimal elemen array.

Contoh pendeklarasian array :

```
float nilai_tes[5];
```

pada program diatas menyatakan bahwa array **nilai_tes** mengandung 5 elemen bertipe *float*. Pada bahasa C, data array akan disimpan dalam memori yang berurutan. Elemen pertama mempunyai indeks bernilai 0. Jika variabel **nilai_tes** dideklarasikan sebagai array dengan 5 elemen, maka elemen pertama memiliki indeks sama dengan 0, dan elemen terakhir memiliki indeks 4. Bentuk umum pengaksesan array adalah:

```
nama_var[indeks]
```

sehingga, untuk array **nilai_tes**, maka:

```
nilai_tes[0] → elemen pertama dari nilai_tes
nilai_tes[4] → elemen ke-5 dari nilai_tes\
```

perhatikan contoh berikut ini:

```
nilai_tes[0] = 70;
scanf("%f", &nilai_tes[2]);
```

Contoh pertama merupakan pemberian nilai 70 ke **nilai_tes[0]**. Sedangkan contoh 2 merupakan perintah untuk membaca data bilangan dari keyboard dan diberikan ke **nilai_tes[2]**. Pada contoh 2 ini

```
&nilai_tes[2]
```

berarti "alamat dari nilai_tes[2]". Perlu diingat bahwa *scanf()* memerlukan argument berupa alamat dari variabel yang digunakan untuk menyimpan nilai masukan. Selengkapnya perhatikan contoh program di bawah ini.

Sebuah array dapat diinisialisasi sekaligus pada saat dideklarasikan. Untuk mendeklarasikan array, nilai-nilai yang diinisialisasikan dituliskan di antara kurung kurawal ({}) yang dipisahkan dengan koma.

```
int jum[12] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
```

Ada beberapa variasi cara mendeklarasikan sebuah array (dalam hal ini yang berdimensi satu), di antaranya adalah sebagai berikut :

- int numbers[10];
- int numbers[10] = {34, 27, 16};
- int numbers[ ] = {2, -3, 45, 79, -14, 5, 9, 28, -1, 0};
- char text[ ] = "Welcome to New Zealand.";
- float radix[12] = {134.362, 1913.248};
- double radians[1000];

Array dideklarasikan dengan tanda [ ] (bracket), baik dalam bahasa C++ dan Java. Bentuk umum dari tipe data array adalah :

```
tipe_data nama_array[jumlah_elemen]
```

Jika ingin mendeklarasikan sebuah array dengan tipe data integer dengan nama a dan jumlah elemen array-nya 10 maka kodenya adalah :

```
int a[10];
```

Dalam bahasa Java pendeklarasian array lebih variarif. Selain dengan kode seperti di atas, Java juga dapat mendeklarasikan array dalam bentuk :

```
int[ ] a;
```

Kemudian setelah melakukan deklarasi array, baik dengan kode yang pertama maupun yang kedua, Java harus menciptakan (membuat) objek terlebih dahulu sebelum array dapat digunakan karena dalam Java array merupakan sebuah Class. Cara menciptakan objek array dalam Java adalah :

```
a = new int[10];
```

Program 9.1

Dalam Java pendeklarasian array dan pembuatan objek array dapat dilakukan alam satu sintak, yaitu :

```
int[ ] a = new int[10];
```

atau

```
int a[ ] = new int[10];
```

Baik C++ maupun Java, untuk mengakses elemen array, misalnya elemen ke-10 dari array dan kemudian menampung nilainya dalam sebuah variabel x, maka sintaknya adalah :

```
x=a[9];
```

Untuk memasukkan data ke dalam array, sintak yang digunakan adalah :

```
a[nomor_elemen] = data;
```

```
a[0] = 5;
a[1] = 6;
a[2] = 7;
dan seterusnya.
```

Agar lebih efisien dan efektif, maka pemasukan data dalam array dapat menggunakan perulangan seperti berikut ini :

```
for (i=0; i<jumlah_data; i++) {
      cout << “a[“ << i << “] = “;
      cin >> a[i];
}
```

Berikut contoh program lengkap dalam bahasa C++ adalah :

```
#include <iostream>

using namespace std;

int a[10],jumlah=10;
bool cari(int cariData,int nElemen) {
        int i;
        for(i=0; i<nElemen; i++) {
                if(a[i] == cariData)
                break;
        }
        if(i == nElemen) return false;
        else return true;
}
void input(int data,int i) {
a[i] = data;
}
void hapus(int data,int nElemen) {
        int i;
        for(i=0; i<nElemen; i++) {
                if( data == a[i] )
                break;
        }
        if(i==nElemen)  cout << "Data"<< data << "tidak terhapus (tidak ada)" << endl;
        else {
                for(int j=i; j<nElemen-1; j++) {
                        a[j] = a[j+1];
                }
        cout << "Data " << data << " dihapus" << endl;
        }
}
void tampil(int nElemen) {
        for(int i=0; i<nElemen; i++)
                cout << a[i] << " ";
                cout << endl;
}
void main() {
        int data;
        for(int i=0; i<jumlah; i++){
        cout << "a[" << i << "] = ";
        cin >> data;
        input(data,i);
        }
```

```
        tampil(jumlah);
        int cariData = 12;
        if (cari(cariData,jumlah)==false)
                cout << "Data " << cariData << " tidak ditemukan" << endl;
        else
                cout << "Data " << cariData << " ditemukan" << endl;
        hapus(89,jumlah);
        jumlah--;
        hapus(0,jumlah);
        jumlah--;
        tampil(jumlah);
}
```

Program di atas terdiri dari empat function yaitu function cari() yang digunakan untuk mencari data dalam array, function input() digunakan untuk memasukkan data dalam array, hapus() untuk menghapus data dalam array, dan function tampil() untuk menampilkan data dalam array.

Keempat function tersebut kemudian dipanggil satu per satu oleh program utama adalah kode untuk memasukkan data dalam array dengan argumen pemanggilan function input() sebanyak sepuluh kali dengan menggunakan perulangan.

Setelah memasukkan data selesai, maka data yang telah dimasukan dalam array kemudian ditampilkan dengan menggunakan argument. Function tampil() mengakses data array satu persatu dan kemudian menampilkannya.

Pencarian data dilakukan oleh program utama. Function pencarian data ini (cari()) dilakukan dengan mengunjungi atau mengakses data array satu persatu dan kemudian membandingkan data pada setiap elemen indek dengan data yang dicari. Jika nilai datanya sama dengan nilai data yang dicari, maka proses pencarian data dihentikan. Jika tidak, maka pencarian terus dilakukan sampai semua data array diakses. Jika proses pencarian yang dilakukan sama dengan jumlah elemen array, maka data yang dicari tidak ditemukan. Proses pencarian seperti ini merupakan metode linier atau sekuensial (Linear / Sequential Search).

Program ini melakukan proses penghapusan data sebanyak dua kali dimana setiap kali melakukan penghapusan data maka jumlah elemen array dikurangi satu.

Keluaran dari program tersebut adalah:

```
a[0] = 18
a[1] = 12
a[2] = 34
a[3] = 44
a[4] = 89
a[5] = 34
```

```
a[6] = 63
a[7] = 24
a[8] = 67
a[9] = 3
18 12 34 44 89 34 63 24 67 3
Data 12 ditemukan
Data 89 dihapus
Data 0 tidak terhapus <tidak ada>
2 34 44 34 63 24 67
```

9.2. Deklarasi Array

Array adalah variabel yang mampu menyimpan sejumlah nilai yang bertipe sama. Untuk mendeklarasikan sebuah array, harus disebutkan tipe dari array yang dibuat misalnya int, float atau double dan juga ukuran array. Untuk menentukan ukuran array,perlu ditempatkan jumlah nilai yang dapat disimpan array dalam sebuah tanda kurung kurawal siku yang terletak sesuah nama array. Deklarasi berikut sebagai contoh untuk menciptakan array bernama skor yang mampu menyimpan 100 skor nilai yang bertipe int.

```
int skor[100];
```

Pada saat dideklarasikan sebuah array, compiler C mengalokasikan memori yang cukup untuk menampung semua elemen sesuai dengan yang dideklarasikan. Masukan pertama berada pada lokasi 0. Sebagai contoh berdasarkan array skor, pernyataan berikut menugaskan nilai 80 pada elemen pertama dari array

```
skor[0] = 80;
```

Karena elemen pertama dari array dimulai dengan offset 0, maka elemen terakhir dari array adalah satu lokasi sebelum ukuran array. Berdasarkan array skor diatas, pernyataan berikut menugaskan nilai ke elemen terakhir dari array.

```
skor[99]  = 75;
```

Inisialisasi array adalah pemberian nilai default pada array. Pemberian nilai default ini dilakukan pada saat array dideklarasikan.

Bentuk umum dari inisialisasi array ini adalah sebagai berikut :

```
tipe_data nama_array[jumlah_elemen] = {nilai1,nilai2,…,nilaiN}
```

Nilai didalam kurung kurawal disebut dengan Initialization List.

```
int a[10] = { 0, 3, 6, 9, 12, 15, 18, 21, 24, 27 };
```

Untuk lebih jelasnya, perhatikan contoh berikut ini :

Program 9.2.

```
#include <iostream>

using namespace std;

int a[5]={2,4,6,8,10},jumlah=5;
void input(int data,int i) {
        a[i] = data;
}
void tampil(int nElemen) {
        for(int i=0; i<nElemen; i++)
                cout << a[i] << " ";
                cout << endl;
}
void main() {
        int data;
        cout << "Belum ada perubahan data array" << endl;
        tampil(jumlah);
        for(int i=0; i<jumlah; i++){
                cout << "a[" << i << "] = ";
                cin >> data;
                input(data,i);
        }
        cout << "Setelah ada perubahan data array" << endl;
        tampil(jumlah);
}
```

Keluaran dari program di atas adalah :

```
Belum ada perubahan data array
2 4 6 8 10
a[0] = 12
a[1] = 43
a[2] = 67
a[3] = 34
a[4] = 48
setelah asa perubahan data array
12 43 67 34 48
```

Program di atas mempunyai tipe data array yang berisi lima elemen dimana nilai default dari kelima elemen array tersebut adalah 2, 4, 6, 8, 10 sehingga ketika data array dipanggil atau ditampilkan maka array sudah mempunyai data, tidak kosong seperti halnya pada kode berikut ini.

Program 9.3.

```
#include <iostream>

using namespace std

int a[5],jumlah=5;
void input(int data,int i) {
        a[i] = data;
}
void tampil(int nElemen) {
        for(int i=0; i<nElemen; i++)
                cout << a[i] << " ";
                cout << endl;
        }
void main() {
        int data;
        cout << "Belum ada perubahan data array" << endl;
        tampil(jumlah);
        for(int i=0; i<jumlah; i++){
                cout << "a[" << i << "] = ";
                cin >> data;
                input(data,i);
        }
        cout << "Setelah ada perubahan data array" << endl;
        tampil(jumlah);
}
```

Keluaran program diatas adalah sebagai berikut:

```
Belum ada perubahan data array
0 0 0 0 0
a[0] = 12
a[1] = 13
a[2] = 14
a[3] = 15
a[4] = 16
setelah ada perubahan data array
12 13 14 15 16
```

Kedua program di atas sama. Bedanya program utama tipe data array-nya diinisialisasi, sedangkan program yang kedua tidak diinisialisasi sehingga pada waktu dipanggil pertama kali nilainya masih kosong. Baik pada program pertama maupun yang kedua, merupakan kode untuk memasukkan data baru pada array yang artinya data atau nilai default dari array ditimpa dengan data yang baru.

9.3. Inisialisasi Array

Untuk inisialisasi array ada beberapa macam yang sering dilakukan sebagai contoh perhatikan potongan program berikut ini :

```
char title[] = “Dasar Pemrograman”;
char section [64] = “Arrays”;
```

Pada kasus yang pertama, compiler C akan mengalokasikan 17 byte untuk menampung string. Pada kasus yang kedua, compiler akan mengalokasikan array 64 byte, menginisialisasi enam karakter yang yang pertama dengan huruf-huruf “Arrays” dan karakter NULL. Kebanyakan compiler juga akan menginisialisasi lokasi byte-byte tersisa dengan NULL.

Pada saat dideklarasikan array dengan tipe lain, kita juga dapat menginisialisasinya dengan cara yang sama. Sebagai contoh pernyataan berikut menginisialisasi array integer scores dengan nilai 80, 70, 90, 85 dan 80.

```
int scores[5] = {80, 70, 90, 85, 80} ;
```

Pada saat kita menugaskan nilai awal ke suatu array , kita harus melingkupi nilai-nilai tersebut dengan tanda kurung kurawal{}. Dalam hal ini, ukuran array sesuai dengan jumlah nilai yang ditugaskan ke array. Akan tetapi pernyataan berikut akan menugaskan empat nilai titik mengambang ke array yang dapat menyimpan 64 nilai.

```
float salaries[64] = {2500000.0, 3200000.0, 4400000.0, 5500000.0};
```

Tergantung dari compiler yang digunakan, nilai-nilai yang ditugaskan ke elemen-elemen yang tidak secara eksplisit ditugaskan mungkin diisi dengan 0. Sebagai pegangan kita harus tidak menganggap bahwa compiler akan menginisialisasi elemen-elemen yang lain. Jika kita tidak menyebutkan ukuran array, kompiler akan mengalokasikan memori yang hanya cukup untuk menampung nilai-nilai yang disebutkan. Deklarasi array berikut menciptakan array yang cukup menampung tiga buah nilai yang bertipe long.

```
long planets[] = {1234L, 5678L, 12347L};
```

9.4. Array Multi Dimensi

Array multi dimensi adalah suatu array yang mempunyai lebih dari satu subskrip. Array multi dimensi ini aplikasinya antara lain untuk matrik. Adapun deklarasi dari array multi dimensi adalah :

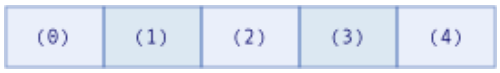
```
tipe_data
```

```
nama_array[jumlah_elemen_baris]
[jumlah_elemen_kolom];
```

9.4.1. Array Satu Dimensi

Banyak array yang hanya memiliki satu dimensi, seperti sejumlah orang dari setiap umur.

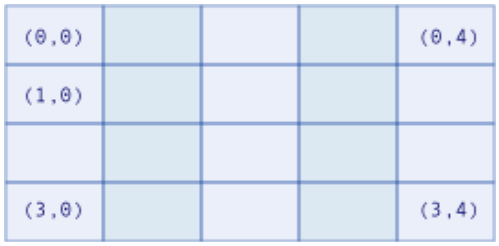
Satu-satunya persyaratan untuk menentukan elemen adalah usia yang berpendapat bahwa unsur hitungan. Karena itu, seperti array hanya menggunakan satu indeks saja. Gambar berikut menyatakan variabel array satu-dimensi.



Gambar 9.1. Array Satu Dimensi

9.4.2. Array Dua Dimensi

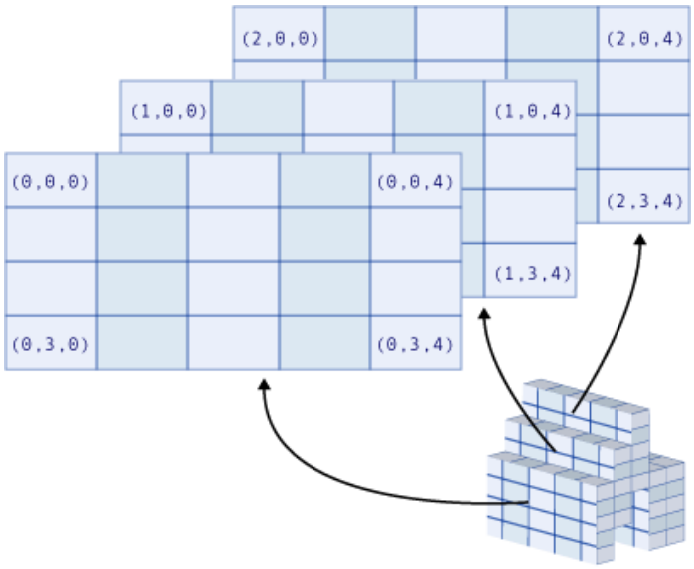
Ada beberapa array dua dimensi, seperti jumlah kantor-kantor di setiap lantai di setiap gedung kampus. Spesifikasi yang memerlukan kedua elemen bangunan dan jumlah lantai, dan setiap elemen yang berpendapat bahwa untuk menghitung kombinasi bangunan dan lantai. Karena itu, seperti array menggunakan dua indeks. Gambar berikut menyatakan variabel array dua dimensi



Gambar 9.2. Array Dua Dimensi

9.4.3. Tiga Dimensi

Beberapa array memiliki tiga dimensi, seperti nilai dalam tiga dimensi ruang. Seperti array menggunakan tiga indeks, yang dalam hal ini mewakili x, y, z dan koordinat dari ruang fisik. gambar berikut menyatakan variabel untuk mengadakan array tiga dimensi.



Gambar 9.3. Array 3 Dimensi

Untuk lebih jelasnya mengenai array multi dimensi dibawah ini diberikan beberapa contoh array. Arrai dibawah ini merupakan matrik 2 X 2 yang menggunakan array.

```
int matrik[2][2];
```

Berikut adalah contoh program lengkapnya:

Program 9.4.

```
#include <iostream>

using namespace std;

int main(void) {
        int t, i, matrik[3][4];
        for(t=0; t<3; t++)
        for(i=0; i<4; i++)
                        matrik[t][i] = (t*4)+i+1;
        for(t=0; t<3; t++) {
                for(i=0; i<4; i++)
                        cout << matrik[t][i] << "  ";
                cout << endl;
        }
        return 0;
}
```

Keluaran dari program di atas adalah :

```
1 2 3 4
5 6 7 8
9 10 11 12
```

t

t	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12

Baris ke-5 sampai ke-7 adalah pemasukan data pada matrik. Sedangkan baris ke-8 sampai ke-12 adalah menampilkan data matrik. Pada kedua proses tersebut, pemasukan data dan menampilkan data terdapat dua buah perulangan dimana perulangan pertama adalah untuk mengidentifikasi baris dari matrik dan perulangan kedua untuk mengidentifikasi kolom dari matrik.

Berikut ini adalah contoh untuk penjumlahan dua buah matrik.

Program 9.5.

```
#include <iostream>

using namespace std;

int main(void) {
	int t, i, A[3][4],B[3][4],C[3][4];
	for(t=0; t<3; t++)
	for(i=0; i<4; i++)
				A[t][i] = (t*4)+i+1;
	cout << "MATRIK A" << endl;
	for(t=0; t<3; t++) {
		for(i=0; i<4; i++)
				cout <<A[t][i] << "  ";
				cout << endl;
	}
	for(t=0; t<3; t++)
		for(i=0; i<4; i++)
				B[t][i] = (t*4)+i+1;
	cout << endl;
	cout << "MATRIK B" << endl;
	for(t=0; t<3; t++) {
		for(i=0; i<4; i++)
				cout <<B[t][i] << "  ";
		cout << endl;
	}
	for(t=0; t<3; t++)
		for(i=0; i<4; i++)
				C[t][i] = A[t][i]+B[t][i];
	cout << endl;
	cout << "MATRIK C = A + B" << endl;
	for(t=0; t<3; t++) {
		for(i=0; i<4; i++)
				cout <<C[t][i] << "  ";
		cout << endl;
	}
	return 0;
}
```

Keluaran program :

```
MATRIK A
```

```
1 2 3 4
5 6 7 8
9 10 11 12 13

MATRIK B
1 2 3 4
5 6 7 8
9 10 11 12

MATRIK C = A +B
2 4 6 8
10 12 1416
18 20 22 24
```

9.5. Mengurutkan Elemen Array

Ada banyak sekali metode untuk mengurutkan data, antara lain Exchange Sort, Selection Sort, Insertion Sort, Bubble Sort, Quick Sort, Shell Sort, Binary Insertion Sort, dan masih banyak lagi.

Dari sekian banyak metode, hanya metode Exchange Sort, Bubble Sort, dan Insertion Sort saja yang akan kita bahas karena ketiga metode ini merupakan metode yang paling mudah dan banyak digunakan.

Untuk metode Selection Sort langkah-langkahnya adalah sebagai berikut :

1. Proses 1: variabel indek diberi nilai 1 (data ke-1) kemudian data indek dibandingkan dengan data ke-2. Jika data indek lebih besar maka nilai indeknya diganti dengan 2 (data ke-2), jika tidak nilai indeknya tetap.
   Kemudian data indek dibandingkan lagi dengan data ke-3, lebih besar? Nilai indek ditukar! Demikian seterusnya.
   Setelah selesai, nilai indek diperiksa apakah nilai indek berubah atau tidak. Jika nilai indek mengalami perubahan maka data ke-1 ditukar dengan data indek.
2. Pada proses 2: variabel indek diberi nilai 2 (data ke-2) kemudian data indek dibandingkan dengan data ke-3. Jika data indek lebih besar maka nilai indeknya diganti dengan 3 (data ke-3).
   Kemudian data indek dibandingkan lagi dengan data ke-4, lebih besar? Nilai indek ditukar! Demikian seterusnya.
   Setelah selesai, nilai indek diperiksa apakah nilai indek berubah atau tidak. Jika nilai indek mengalami perubahan maka data ke-2 ditukar dengan data indek.
3. Demikian untuk proses selanjutnya dimana banyak prosesnya adalah jumlah elemen array dikurangi satu.

Berikut program lengkap metode Selection Sort dalam bahasa C++ adalah:

Program 9.6

```
#include <iostream>

using namespace std;

int a[10];
void input(int data,int i) {
        a[i] = data;
}
void tampil() {
        for(int i=0; i<10; i++)
                cout << a[i] << " ";
        cout << endl;
 }
void tukar(int data1, int data2) {
        long temp = a[data1];    a[data1] = a[data2];
        a[data2] = temp;
}
void selectionSort() {
        int i,j,indek;
        for(i=0; i<5-1; i++) {
                indek=i;
                for(j=i+1; j<5; j++)
                        if(a[indek] > a[j] )
                                indek=j;
                tukar(i,indek);
        }
}
void main() {
        input(57,0);
        input(89,1);
        input(49,2);
        input(51,3);
        input(12,4);
        input(90,5);
        input(1,6);
        input(0,7);
        input(63,8);
        input(25,9);
        tampil();
        selectionSort();
```

```
        tampil();
}
```

Keluaran program diatas adalah sebagai berikut :

```
57 89 49 51 12 90 1 0 63 25
12 49 51 57 89 90 1 0 63 25
```

Metode Bubble Sort mempunyai langkah-langkah sebagai berikut :

1. Membandingkan data ke-1 dengan data ke-2, jika data ke-1 lebih besar, maka kedua data ditukar.
2. Kemudian membandingkan data ke-2 dengan data ke-3, jika data ke-2 lebih besar, kedua data ditukar lagi.
3. Demikian seterusnya sampai data terakhir, sehingga data kedudukannya akan bergeser-geser.
4. Untuk proses 2, pembandingan (pergeseran data) hanya sampai pada data terakhir dikurangi satu.

Kode program lengkapnya dalam bahasa C++ adalah sebagai berikut :

Program 9.7

```
#include <iostream>

using namespace std;

int a[10];
void input(int data,int i) {
        a[i] = data;
}
void tampil() {
        for(int i=0; i<10; i++)
                cout << a[i] << " ";
        cout << endl;
}
void tukar(int data1, int data2) {
        long temp = a[data1];
        a[data1] = a[data2];
        a[data2] = temp;
}
void bubSort() {
        int i, j;
        for(i=10-1; i>1; i--)
                for(j=0; j<i; j++)
                        if( a[j] > a[j+1] ) tukar(j, j+1);
}
```

```
void main() {
        input(57,0);
        input(89,1);
        input(49,2);
        input(51,3);
        input(12,4);
        input(90,5);
        input(1,6);
        input(0,7);
        input(63,8);
        input(25,9);
        tampil();
        bubSort();
        tampil();
}
```

Keluaran program diatas adalah sebagai berikut:

```
57 89 49 51 12 90 1 0 63 25
0 1 12 25 49 51 57 63 89 90
```

Metode Insertion Sort mirip dengan cara orang mengurutkan kartu selembar demi selembar, kartu diambil dan disisipkan (insert) ke tempat yang seharusnya. Adapun langkah-langkahnya adalah sebagai berikut :

- Pengurutan dimulai dari data ke-2 sampai dengan data terakhir.
- Jika ditemukan data yang lebih kecil atau lebih besar, maka akan ditempatkan (diinsert) diposisi yang seharusnya

Program selengkapnya dalam bahasa C++ adalah :

Program 9.8.

```
#include <iostream>

using namespace std;

int a[10];
void input(int data,int i) {
        a[i] = data;
}
void tampil() {
        for(int i=0; i<10; i++)
                cout << a[i] << " ";
        cout << endl;
}
void insertionSort() {
```

```
        int i,j;
        for(i=1; i<10; i++) {
                long temp = a[i];
                j=i;
                while(j>0 && a[j-1] >= temp) {
                        a[j] = a[j-1];
                        --j;
                }
                a[j] = temp;
        }
}
void main() {
        input(57,0);
        input(89,1);
        input(49,2);
        input(51,3);
        input(12,4);
        input(90,5);
        input(1,6);
        input(0,7);
        input(63,8);
        input(25,9);
        tampil();
        insertionSort();
        tampil();
}
```

Keluaran programnya adalah sebagai berikut:

```
57 89 49 51 12 90 1 0 63 25
0 1 12 25 49 51 57 63 89 90
```

9.6. Contoh Program Array

Dibawah ini merupakan beberapa contoh program yang bekerja menggunakan array. Array merupakan variabel yang mampu menyimpan sejumlah nilai yang bertipe sama. Pada contoh dibawah ini menggunakan array yang bekerja satu dimensi maupun multidimensi (dimensi dua atau tiga). Untuk lebih jelasnya perhatikan beberapa contoh dibawah ini:

Program 9.9.
Program di bawah ini untuk membaca data kemudian menampilkannya.

```
#include<iostream.h>
#include<conio.h>
```

```
using namespace std;

void main()
{
    int data[10];           // array dengan 10 elemen bertipe integer
    int elemen;

    // entri 10 data
    for (elemen=0;elemen <= 9;elemen++)
    {
          cout << "Data ke - " << elemen << ": ";
          cin >> data[elemen];
          }
             // tampilkan data setelah entri
             for (elemen=0;elemen <= 9;elemen++)
          {
          cout << "Data ke - " << elemen << ": " << data[elemen];
    }
}
```

Catatan: Dalam C/C++ elemen array dimulai dari 0.

Program 9.10.
Program untuk menampilkan data array dari hasil inisialisasi:

```
include<iostream.h>
#include<conio.h>

using namespace std;

void main()
{
    int data[5] = {4, 1, 0, -9, 8};
    int elemen;

    // tampilkan data
    for (elemen=0;elemen <= 4;elemen++)
    {
          cout << "Data ke - " << elemen << ": " << data[elemen];
    }
}
```

Program 9.11.

Program untuk mencari data dari array, dan menampilkan nomor elemennya.

```
#include<iostream.h>
#include<conio.h>

void main()
{
      int x;
      int data[10] = {4, 1, 0, -9, 8, 5, -1, 2, 3, -7};
      int elemen, ketemu;
      cout << "Data yang dicari : ";
      cin >> x;
      ketemu = 0;
      for(elemen=0; elemen<= 9; elemen++)
      {
          if (data[elemen] == x)
          { ketemu =!
              ketemu;
              break;
          }
      }
      if (ketemu == 0) cout << "Data tidak ditemukan ";
      else cout << "Data ada di elemen : " << elemen;
}
```

Program 9.12.
Program untuk menampilkan data terbesar (maks) dari suatu array.

```
#include<iostream.h>
#include<conio.h>

using namespace std;

void main()
{
    int data[10] = {4, 1, 0, -9, 8, 5, -1, 2, 3, -7};
    int elemen, max;
    max = data[0];
    for(elemen=0; elemen<= 9; elemen++)
    {
        if (data[elemen]>max) max = data[elemen];
        else max = max;
    }
    cout << "Nilai maksimum adalah : " << max;
}
```

Array di atas adalah array dimensi satu. Bagaimana dengan array dimensi dua? Berikut ini contoh penggunaan array dua dimensi:

Program 9.13.

```
#include<iostream.h>
#include<conio.h>

using namespace std;

void main()
{
    int j, k;
    int data[5][3] ={ {3, 4, -1}, {2, 3, 0}, {1, 1, 2}, {5, 9, -4}, {6, 6, 2} };
    for (j = 0; j<=4; j++)
    {
        for (k = 0; k<=2; k++)
        cout << "data[" << j << "]["<< k << "] = " << data[j][k] << endl;
    }
}
```

9.7. Soal Latihan

Jawablah soal latihan dibawah ini dengan baik dan benar.

1. Apa yang dimaksud dengan array
2. Apa kelebihan dan kekurangan program menggunakan array
3. Bagaimana cara mendeklarasikan array
4. Bagaimana cara menginisialisasi array
5. Buatlah program penjumlahan matrik dengan menggunakan array 2 dimensi
6. Buatlah salah satu program pengurutan dengan array

BAB. 10
REKURSIF

10.1. Pengertian Rekursif

Rekursi adalah konsep pengulangan yang penting dalam ilmu komputer. Konsep ini dapat digunakan untuk merumuskan solusi sederhana dalam sebuah permasalahan yang sulit untuk diselesaikan secara iteratif dengan menggunakan loop for, while do.

Pada saat tertentu konsep ini dapat digunakan untuk mendefinisikan permasalahan dengan konsisten dan sederhana. Pada saat yang lain, rekursi dapat membantu untuk mengekspresikan algoritma dalam sebuah rumusan yang menjadikan tampilan algoritma tersebut mudah untuk dianalisa.

Rekursif adalah salah satu metode dalam dunia matematika dimana definisi sebuah fungsi mengandung fungsi itu sendiri. Dalam dunia pemrograman, rekursi diimplementasikan dalam sebuah fungsi yang memanggil dirinya sendiri. Atau Rekursif merupakan satu teknik pemrograman dengan cara memanggil sebuah fungsi dari dirinya sendiri, baik itu secara langsung maupun tidak langsung. Pemanggilan fungsi rekursif secara langsung berarti dalam fungsi tersebut terdapat *statement* untuk memanggil dirinya sendiri sedangkan secara tidak langsung berarti fungsi rekursif tersebut memanggil 1 atau lebih fungsi lain sebelum memanggil dirinya sendiri.

Fungsi rekursif langsung merupakan rekursif, jika ekspresi yang merealisasikan fungsi tersebut mengandung aplikasi terhadap fungsi tersebut.

```
Realisasi
F (<list-param>)
   depend on
         <kondisi-basis> : <ekspresi-1>
```

```
        <kondisi-rekurensi> : F
<ekspresi-2>
```

Fungsi rekursif tidak langsung merupakan realisasi fungsi yang dapat *cross-recursif* yaitu jika realisasi fungsi f mengandung fungsi g yang realisasinya adalah aplikasi terhadap f.

```
Realisasi
G (<list-param>): F (<ekspresi-1>)
F (<list-param>):
   depend on
        <kondisi-basis> : <ekspresi-1>
        <kondisi-rekurensi> : G
<ekspresi-2>
```

Fungsi merupakan sub program yang sangat bermanfaat dalam pemrograman, terutama untuk program atau proyek yang besar. Manfaat penggunaan subprogram antara lain adalah :

- Meningkatkan *readibility*, yaitu mempermudah pembacaan program
- Meningkatkan *modularity*, yaitu memecah sesuatu yang besar menjadi modul-modul atau bagian-bagian yang lebih kecil sesuai dengan fungsinya, sehingga mempermudah pengecekan, testing dan lokalisasi kesalahan.
- Meningkatkan *reusability*, yaitu suatu sub program dapat dipakai berulang kali dengan hanya memanggil sub program tersebut tanpa menuliskan perintah-perintah yang semestinya diulang-ulang.

Sub Program Rekursif adalah subprogram yang memanggil dirinya sendiri selama kondisi pemanggilan dipenuhi. Dengan melihat sifat sub program rekursif di atas maka sub program rekursif harus memiliki :

- kondisi yang menyebabkan pemanggilan dirinya berhenti (disebut kondisi khusus atau special condition)
- pemanggilan diri sub program (yaitu bila kondisi khusus tidak dipenuhi)

Secara umum bentuk dari sub program rekursif memiliki statemen kondisional :

```
if kondisi khusus tak dipenuhi
then panggil diri-sendiri dengan parameter yang sesuai
else lakukan instruksi yang akan dieksekusi bila kondisi khusus dipenuhi
```

Sub program rekursif umumnya dipakai untuk permasalahan yang memiliki langkah penyelesaian yang terpola atau langkah-langkah yang teratur. Bila kita memiliki suatu permasalahan dan kita mengetahui algoritma penyelesaiannya, kadang-kadang sub program rekursif menjadi pilihan kita bila memang memungkinkan untuk dipergunakan. Secara algoritmis (dari segi algoritma, yaitu bila kita mempertimbangkan penggunaan memori, waktu eksekusi sub program) sub program rekursif sering bersifat tidak efisien . Dengan demikian sub program rekursif umumnya memiliki efisiensi dalam penulisan perintah, tetapi kadang tidak efisien secara algoritmis. Meskipun demikian banyak pula permasalahan-permasalahan yang lebih sesuai diselesaikan dengan cara rekursif (misalnya dalam pencarian/ *searching*).

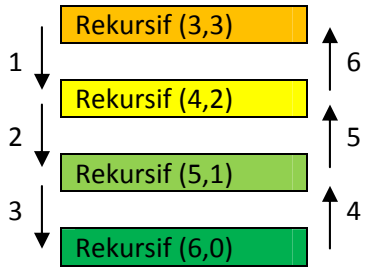
Untuk dapat memahami proses yang terjadi dalam sebuah fungsi rekursif , perhatikan contoh fungsi rekursif berikut :

```
void rekursi (int a, int b)
{
        If (b==0) return;
        a++;
        b--;
        cout<<"Masuk  \ n"<<a <<b;
        rekursi(a,b);
        cout<<"Keluar \ n"<<a <<b;
}
```

Misalkan Fungsi tersebut dipanggil dengan nilai a = 3 dan b = 3 maka pertama tama di cek apakah b = 0 (if (b == 0) return), jika sama maka keluar. Ternyata nilai b tidak sama dengan 0 maka tambahkan a dengan 1 dan kurangi b dengan 1. Maka keadaan sekarang menjadi a = 4 dan b = 2 . Baris berikutnya menampilkan nilai a dan b ke layar (printf cout<<"Masuk \ n"<<a <<b;). Kemudian panggil fungsi rekursi dengan nilai a = 4 dan b = 2 . Langkah – langkah tersebut diulang terus sampai pemanggilan fungsi rekursi dengan nilai a = 6 dan b = 0. Pada saat ini kondisi if bernilai benar sehingga fungsi akan keluar (return) dan melanjutkan perintah setelah pemanggilan fungsi rekursi dengan a = 6 dan b = 0. Yaitu mencetak nilai a dan b (cout<<"Keluar \n"<<a << b;).

Setelah mencetak nilai a dan b maka fungsi rekursif akan keluar lagi , dan melanjutkan perintah setelah pemanggilan fungsi rekursif sebelumnya dimana nilai a = 5 dan b = 1 .

Demikian seterusnya sampai nilai a = 4 dan nilai b = 2. yang tidak lain pemanggilan fungsi rekurif yang pertama. Proses pemanggilan fungsi rekursif dapat diilustrasikan:



Langkah ke:	
1	a=4 ; b=2 . Cetak : masuk → a=4 || b=2
2	a=5 ; b=1 . Cetak : masuk → a=5 || b=1
3	a=6 ; b=0 . Cetak : masuk → a=6 || b=0
4	a=6 ; b=0 . Cetak : keluar → a=6 || b=0
5	a=5 ; b=1 . Cetak : keluar → a=5 || b=1
6	a=4 ; b=2 . Cetak : keluar → a=4 || b=2

Gambar 10.1. Proses Pemanggilan Fungsi Rekursif

Penggunaan fungsi rekursif misalnya pada fungsi pangkat, faktorial, dan barisan fibonacci. Mari kita lihat satu demi satu. Dalam fungsi pangkat xy , kita tahu bahwa semua bilangan selain 0, jika dipangkatkan dengan 0 nilainya sama dengan 1. Jika x dipangkatkan

dengan y, dengan y lebih dari 0, maka hasilnya sama dengan x dikalikan dengan x dipangkatkan y – 1. Jika dituliskan dalam notasi matematika definisinya adalah sebagai berikut:

$$x^y = 1, jika\ y = 0$$

$$x^y = x * x^{y-1}, jika\ y > 0$$

Kita lihat di atas pada definisi y > 0, bentuk pemangkatan muncul kembali di sisi kanan. Itulah yang disebut rekursif. Definisi rekursif selalu dimulai dengan kasus penyetop, penghenti, atau kasus dasar dari suatu permasalahan, dalam hal ini terjadi ketika nilai y = 0. Definisi rekursif yang lebih kompleks mengandung inti dari permasalahan yang akan dipecahkan, namun lebih sederhana. Dalam hal ini yang tadinya x dipangkatkan dengan y, kini bentuk pemangkatan menjadi lebih sederhana, yaitu y – 1. Hal ini dimaksudkan untuk "menggiring" masalah kompleks ke kasus dasar atau penyetop rekursinya. Untuk x = 10 dan y = 0, hasil dari xy adalah 1. Untuk x = 10 dan y = 3 hasilnya dapat digambarkan sebagai berikut:

Konsep rekursifitas banyak ditemui di dunia nyata. Istilah ini muncul pertama kali di kajian bidang matematika. Dalam kasus tertentu, konsep ini memudahkan perumusan formula. Konsep ini pun difasilitasi dalam pemrograman.

Dalam pemrograman, ada 2 terminologi yang bisa didefinisikan dengan rekursif, yaitu prosedur dan fungsi. Seperti halnya dalam bidang matematika, penggunaan konsep ini juga untuk memudahkan pendefinisian dua terminologi tersebut. Bahkan terdapat suatu masalah yang hanya bisa diselesaikan dengan rekursifitas dan sangat sulit untuk diselesaikan tanpa rekursifitas.

Definisi rekursif harus memuat komponen **basis** dan komponen **rekursif**. Dalam pemrograman, komponen ini dapat dipisahkan dengan menggunakan perintah analisa kasus. Misalkan **e1** adalah ekspresi kondisi untuk basis dan **e2** adalah ekspresi kondisi untuk bagian rekursif, definisi rekursif dapat dituliskan:

```
if (e1) then
  {bagian basis}
else
  {bagian rekursif}
```

Salah satu contoh dari kasus rekursif adalah barisan bilangan fibonacci. Barisan bilangan fibonacci adalah 1, 1, 2, 3, 5, 8, 13, 21, ... Definisi barisan bilangan fibonacci adalah sebagai berikut. Misalkan fibonacci(i) menyatakan bilangan fibonacci yang ke-i, maka:

$$fibonacci(i) = \begin{cases} fibonacci(i-1) + fibobacci(i-2) & ,i > 2 \\ 1 & ,i = 1,2 \end{cases}$$

Program 10.1. Penerapan bilangan Fibonaci dengan rekursi

```
#include <iostream>
#include<conio.h>

using namespace std;

int Fibonaci(int);

int main()
{

        int n=7;
        for (int i=1; i<=n; i++)
   cout<<" \ndata "<< Fibonaci(i);
        getch();
   return 0;
}

int Fibonaci(int n)
{
        if (n<=2)
        return n;
        else
        return(Fibonaci(n-2)+Fibonaci(n-1));
}
```

Keluaran program diatas adalah:

```
data 1
data 2
data 3
data 5
data 8
data 13
data 21
```

Rekursif adalah suatu *method* yang memanggil dirinya sendiri secara langsung maupun tidak lansung. Rekursif merupakan teknik pemrograman yang sangat berguna. Dalam beberapa kasus, menggunakan cara rekursi memudahkan pengembangan program secara natural, langsung, dan simple dalam memecahkan problem yang susah dipecahkan. Untuk lebih memahami rekursif perhatikan contoh berikut: Contoh yang umum untuk memahami teknik rekursif adalah masalah factorial.

```
4! = 4 x 3 x 2 x 1
3! = 3 x 2 x 1
2! = 2 x 1
1! = 1
```

Dari contoh di atas bisa diubah cara pandangnya menjadi:

```
4! = 4 x 3!
3! = 3 x 2!
2! = 2 x 1!
1! = 1
```

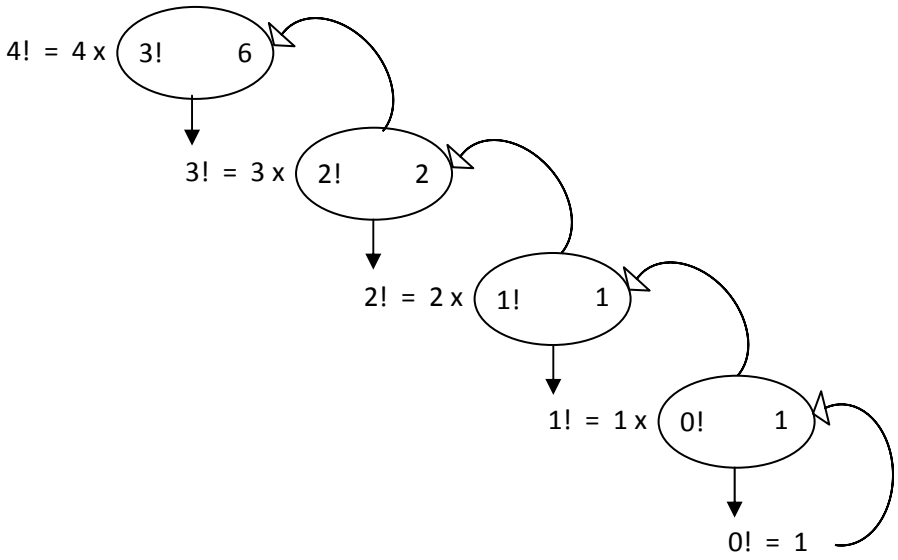
Dengan catatan bahwa 0! = 1 maka cara pandang yang kedua bisa dlanjutkan menjadi:

```
4! = 4 x 3!
3! = 3 x 2!
2! = 2 x 1!
1! = 1
0! = 1
```

Cara pandang yang kedua inilah yang disebut rekursif, karena dalam proses perhitungan selalu memanggil dirinya sendiri. Yaitu factorial memanggil factorial. Akan tetapi perlu dicatat bahwa teknik rekursif ini harus ada terminasinya atau ada saat berhentinya. Pada contoh factorial ini adalah 0!. Yaitu sama dengan 1 bukan 0. Dalam istilah matematika 0! ini disebut dengan *base step*. Sedangkan bagian yang lain disebut *recursive step*. Secara umum rumus factorial dalam bentuk rekursif menjadi sebagai berikut:

```
0! = 1;    base step
n! = n x (n – 1)!; n > 0   recursive step
```

Jadi proses rekursif itu akan selalu memanggil dirinya sendiri dengan variable yang berbeda, dan terus dilakukan sampai mencapai *base step* sebagai terminasinya. Setelah terminasi maka akan mendapatkan hasil dan selanjutnya hasil tersebut dikembalikan ke pemanggilnya untuk dilakukan perhitungan. Hal itu dilanjutkan sampai ke pemanggil yang pertama sehingga mendapatkan hasil sebagaimana mestinya. Ilustrasi berikut memberi gambaran tentang proses rekursif.



Gambar 10.2. Ilustrasi Tentang Proses Rekursif.

10.2. Pengertian Teknik Iteratif

Teknik Iteratif adalah teknik perulangan atau menghitung secara berulang. Teknik ini memanfaatkan kelebihan komputer sebagai mesin hitung yang mampu melakukan perhitungan secara berulang dengan perintah yang sangat sederhana.

Cara ini mudah dilakukan akan tetapi memerlukan proses yang panjang untuk mendapatkan hasil, dan cara ini menggunakan memori langsung. Contoh factorial cara pandang yang pertama menggunakan teknik iteratif

```
4! = 4 x 3 x 2 x 1
3! = 3 x 2 x 1
2! = 2 x 1
1! = 1
```

Untuk menghitung n! berarti mengkalikan bilangan bulat positif dari 1 sampai dengan n. Pengertian penghitunganannya terlihat mudah, namun jika n bilangan yang besar jika dihitung tanpa komputer akan sangat sulit dan lama.

Kebanyakan pemrogram pemula menggunakan cara ini karena lebih jelas algoritmanya.

10.3. Perbandingan Teknik Rekursif dan Teknik Iteratif

Teknik Iteratif lebih mudah dimengerti karena jelas tinggal mengurutkan angka-angkanya dari 1 atau mungkin 0 sampai dengan n. Operasi yang terjadi bisa tambah(+), kurang(-), bagi(/), kali(*), kadangkala juga pangkat atau yang lainnya namun dapat dikembalikan ke empat operasi dasar pertama. Kerugian yang pertama teknik ini

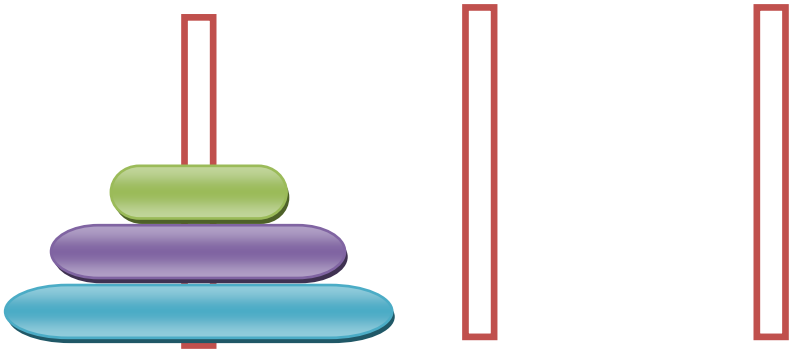
membutuhkan memori yang banyak terutama untuk bilangan yang besar. Yang kedua proses iterasi bisa membutuhkan waktu yang lama apabila bilangannya besar.

Sedang Teknik Rekursif harus tahu formulanya dahulu, dalam hal ini mana *base step*-nya, dan mana *rekursif step*-nya. Biasa dipakai oleh pemrogram yang sudah agak lanjut dan operasi matematikanya merupakan operator biasa. Namun memori yang digunakan bukan memori langsung penampung variable namun merupakan *stack*. Jadi banyak sedikitnya penghitungan tergantung dari besarnya kapasitas *stack* yang dipunyai komputer. Secara rata-rata proses panghitungan dengan rekursif akan lebih cepat karena kompleksitasnya lebih kecil dibanding dengan iteratif.

Kadang seseorang bertanya, loh saya kan juga bisa membuat fungsi pangkat di atas dengan menggunakan teknik iteratif. Misalnya menggunakan while loop. Kenapa harus repot-repot menggunakan rekursif? Memang benar bahwa semua fungsi rekursif dapat dibuat versi iterasinya. Namun demikian, ada beberapa masalah yang jauh lebih mudah jika dipecahkan dengan rekursif. Kode program untuk beberapa masalah rekursif juga relatif lebih mudah dipahami dibandingkan versi iterasinya. Berikut adalah versi iteratif dari fungsi pangkat. Sekaranng perhatikan pada aplikasi permainan menara Hanoi di bawah ini

Menara Hanoi ialah salah satu permainan yang dulunya dimainkan oleh seorang pendeta di Hanoi. Tujuan permainan ini ialah memindahkan n buah pringan dari tonggak asal (A) melalui tonggak bantu (B) menuju tonggak tujuan (C). Dengan aturan–aturan bahwa piringan yang lebih kecil tidak boleh berada di bawah piringan yang lebih besar.

Menara A B C :



Gambar 10.3. Menara dengan Tiga Piringan

Seperti biasa untuk memecahkan masalah kita daftarkan dulu langkah-langkah yang diambil mulai n = 1. Dengan dfinisi piringan yang paling atas ialah piringan 1.

Untuk n = 1 :

- Pindahkan piringan 1 dari A ke C

Untuk n = 2 :

- Pindahkan piringan 1 dari A ke B
- Pindahkan piringan 2 dari A ke C
- Pindahkan piringan 3 dari B ke C

Dari contoh diatas dapat diambil kesimpulan , untuk memindahkan N piringan dari tonggak asal (A) ke tonggak tujuan (C) maka piringan ke N harus berada di tonggak tujuan (C), sedangkan piringan ke 1 sampai (N - 1) harus berada di tonggak bantu (B). Setelah piringan ke 1 sampai (N-1) berada pada tonggak bantu (B), kemudian pindahkan piringan ke 1 sampai (n-1) tersebut dari tonggak bantu (B) ke tonggak tujuan (C).

Nah bagaimana caranya membawa piringan ke 1 sampai (N-1) dari tonggak asal ke tonggak bantu (B), caranya sama saja yaitu dengan memindahkan piringan ke (n-1) dari tonggak asal (A) ke tonggak tujuan yang pada saat ini berada pada tonggak (B) sedangkan piringan dari 1 sampai ke (N-2) dipindahkan ke tonggak bantu yang saat ini berada di tonggak (C) , Setelah piringan ke 1 sampai (N-2) berada pada tonggak bantu (C), kemudian pindahkan piringan ke 1 sampai (N-2) ke tonggak tujuan (B) dan seterusnya.

Metode penyelesaian permainan Hanoi di atas sering disebut sebagai metode back tracking, jadi solusi dicari dengan cara mundur ke belakang dahulu, baru kemudian solusi bisa ditemukan. Berikut Listing program Menara Hanoi dalam bahasa C

Program 10.2

```
include <stdio.h>

void Hanoi(int n,char asal,char bantu, char tujuan)
                                                    // pindahkan piringan ke n
{                                                   // dari asal menuju tujuan
                                                    // melalui bantu
        if (n == 0) return;
        Hanoi(n-1,asal,tujuan,bantu);               // pindahkan piringan ke n-1
                                                    // dari asal ke bantu melalui
                                                    // tonggak tujuan

        printf("Pindah piring ke %d ke dari %c ke %c\n",n,asal,tujuan);
        Hanoi(n-1,bantu,asal,tujuan)     // pindahkan piringan ke n - 1

                                                    // dari bantu menuju tujuan
                                                    // melalu asal
}
```

```
int main(void)
{
        int n;
        printf("Jumlah piringan ? ");
        scanf("%d",&n);
        Hanoi(n,'a','b','c');
        return 0;
}
```

10.4. Algoritma Teknik Rekursif

Telah disinggung di depan untuk Teknik Rekursif harus dicari mana *base step*-nya dan mana *rekursif step*-nya. Dari problem yang hendak dipecahkan dicari dahulu keduanya. Apabila sudah ketemu baru melangkah ke prose berikutnya yaitu menentukan *pseudocode*-nya, sekaligus menentukan apakah memakai *procedure* atau *function*.

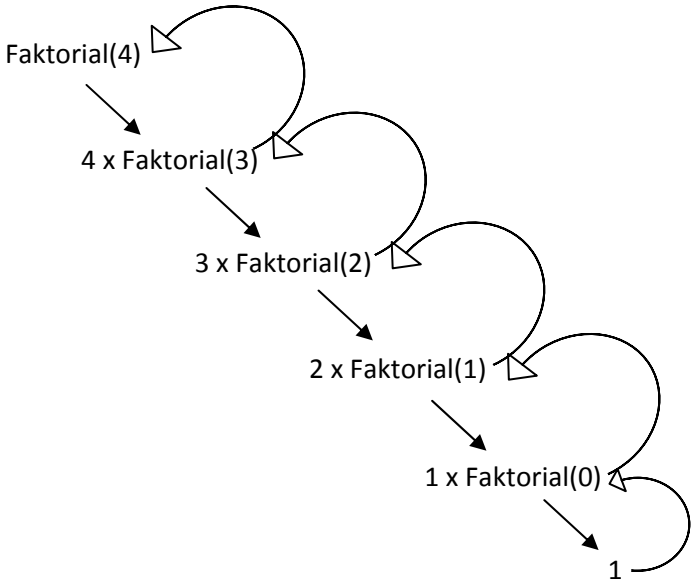
Untuk Teknik Rekursif ini lebih cocok memakai *function*, karena harus bisa dipanggil di dalam program tersebut. Sebagai contoh mencari faktorial seperti di atas. Algoritma rekursifnya sebagai berikut:

```
0! = 1;    base step
n! = n x (n – 1)!; n > 0   recursive step
```

Pseudocode-nya:

```
Int Faktorial(int n)
{
   If (n==0)
      Return 1;
   Else
      Return n * Faktorial(n-1)
}
```

Jika dijalankan gambaran prosesnya sebagai berikut: Misal untuk menghitung 4!



Gambar 10.4. Ilustrasi Rekursif Untuk Menghitung 4! (faktorial 4)

Dalam hal ini memang untuk rekursif harus memakai *function* karena *procedure* tidak dapat dipanggil di dalam *procedure* itu sendiri.

10.5. Algoritma Teknik Iteratif

Teknik Iteratif algoritmanya seperti apa adanya terlihat jadi tidak begitu rumit. Pada tulisan di atas telah dijelaskan beberapa contoh menghitung faktorial secara iteratif. Untuk lebih jelasnya lihat contoh berikut:

```
4! = 4 x 3 x 2 x 1
3! = 3 x 2 x 1
2! = 2 x 1
1! = 1
```

Algoritmanya disini sangat jelas untuk faktorial menggunakan Teknik Iteratif tinggal mengulang operasi perkalian dari 1 sampai dengan banyaknya n.

Pseudocode Teknik Iteratif:

```
int Faktorial(int n)
{
  int x = 1;
  for(int i = 1; i <= n; i++)
    x = x * i;
  return x;
}
```

Dapat dilihat di atas bahwa iterasinya adalah mengulang perkalian x = x * i dari i = 1 sampai dengan I = n dengan penambahan 1 setiap perubahan i. Perlu diketahui juga untuk harga awal x harus sama dengan 1, jika tidak diberi harga awal maka hasilnya akan salah.

Jadi jelas sekali perbedaan algoritma Teknik Rekursif dengan algoritma Teknik Iteratif.

10.6. Penerapan Algoritma Rekursif

Dibawah ini merupakan beberapa contoh program rekursif dengan berbagai kasus dilapangan seperti faktorial, program pemangkatan dan lain sebagainya. Untuk lebih jelasnya perhatikan program-program dibawah ini:

Program 10.3. Faktorial:

```
#include <cstdlib>
#include <iostream>

using namespace std;

int faktorial(int n)
{
  if (n==0)
    return 1;
  else
    return n * faktorial(n-1);
}

int main(int argc, char *argv[])
{
  int n;
  cout<<"Program Faktorial Rekursif\n";
  cout<<"Masukkan Nilai n : ";
  cin>>n;
  cout<<"Faktorial("<<n<<") = "<<faktorial(n)<<"\n";
  system("PAUSE");
  return EXIT_SUCCESS;
}
```

Keluaran program tersebut adalah sebagai berikut:

```
Program factorial Rekursif
Masukan Nilai n : 4
Faktorial (4) = 24
```

Program 10.4. Pangkat (a^n):

```
#include <cstdlib>
#include <iostream>
```

```
using namespace std;

float Pangkat(float a, int n)
{
   if (n==0)
     return 1;
   else
     return a * Pangkat(a, n-1);
}

int main(int argc, char *argv[])
{
       float a;
       int n;
       cout<<"Program Pangkat Rekursif\n";
       cout<<"Isilah Nilai a = ";
       cin>>a;
       cout<<"Isilah Nilai n = ";
       cin>>n;
       cout<<"Hasil dari "<<a<<" pangkat "<<n<<" adalah "<<Pangkat(a,n)<<"\n";
       system("PAUSE");
       return EXIT_SUCCESS;
}
```

Keluaran program di atas seperti berikut:

```
Program pangkat rekursif
Istilah Nilai a     = 2
Istilah Nilai a     = 2
Hasil dari 2 pangkat  3 adalah 8
```

Program 10.5. Menghitung jumlah bilangan Integer positif dari 1 sampai dengan n secara rekursif.

```
#include <cstdlib>
#include <iostream>

using namespace std;

int JumlahInt(int n)
{
   if (n==1)
     return 1;
   else
     return n + JumlahInt(n-1);
```

```
}

int main(int argc, char *argv[])
{
        int n;
        cout<<"Program Jumlah Integer\n";
        cout<<"Masukkan Nilai n = ";
        cin>>n;
        cout<<"Jumlah bilangan Integer dari 1 sampai "<<n<<" = "<<JumlahInt(n)<<"\n";
        system("PAUSE");
        return EXIT_SUCCESS;
}
```

Keluaran program tersebut dapat dilihat dalam gambar berikut ini:

```
Program Jumlah Integer
Masukan Nilai n = 4
Jumlah bilangan Integer dari 1 sampai 4 = 10
```

10.7. Penerapan Algoritma Iteratif

Dibawah ini merupakan beberapa contoh program iteratif dengan berbagai kasus dilapangan seperti faktorial, program pemangkatan dan lain sebagainya. Setelah memperhatikan contoh program dibawah ini harapanya menjadi tahu mengenai perbedaan antara keduanya. Untuk lebih jelasnya perhatikan program-program dibawah ini:

Program 10.6. Faktorial:

```
#include <cstdlib>
#include <iostream>

using namespace std;

int Faktorial(int n)
{
   int x = 1;
   for(int i = 1; i <= n; i++)
     x = x * i;
   return x;
}

int main(int argc, char *argv[])
{
```

```
        int n;
        cout<<"Program Faktorial Iteratif\n";
        cout<<"Masukkan Nilai n : ";
        cin>>n;
        cout<<"Faktorial("<<n<<") = "<<Faktorial(n)<<"\n";
        system("PAUSE");
        return EXIT_SUCCESS;
}
```

Keluaran program sebagai berikut:

```
Program Faktorial Iteratif
Masukan Nilai n : 3
Faktorial (3) = 6
```

Program 10.7. Pangkat secara Iteratif:

```
#include <cstdlib>
#include <iostream>

using namespace std;

float Pangkat(float a, int n)
{
    float hasil = 1;
    if (n==0)
      return 1;
    for (int i=1; i<=n; i++)
    {
       hasil = hasil * a;
    }
    return hasil;
}

int main(int argc, char *argv[])
{
        float a;
        int n;
        cout<<"Program Pangkat Iteratif\n";
        cout<<"Isilah Nilai a = ";
        cin>>a;
        cout<<"Isilah Nilai n = ";
        cin>>n;
        cout<<"Hasil dari "<<a<<" pangkat "<<n<<" adalah "<<Pangkat(a,n)<<"\n";
        system("PAUSE");
```

```
        return EXIT_SUCCESS;
}
```

Keluaran program diatas adalah sebgai berikut:

```
Program Pangkat Iteratif
Istilah Nilai a = 2
Istilah Nilai n = 5
Hasil dari 2 pangkat 5 adalah 32
```

Program 10.8. Menghitung Jumlah Integer Positif secara iteratif:

```
#include <cstdlib>
#include <iostream>

using namespace std;

int JumlahInt(int n)
{
   int hasil = 0;
   for (int i=1; i<=n; i++)
   {
      hasil = hasil + i;
   }
   return hasil;
}

int main(int argc, char *argv[])
{
        int n;
        cout<<"Program Jumlah Integer Iteratif\n";
        cout<<"Masukkan Nilai n = ";
        cin>>n;
        cout<<"Jumlah bilangan Integer dari 1 sampai "<<n<<" = "<<JumlahInt(n)<<"\n";
        system("PAUSE");
        return EXIT_SUCCESS;
}
```

Keluaran program adalah sebagai berikut:

```
Program Jumlah Integer Iteratif
Masukan Nilai n = 5
Jumlah bilangan Integer dari 1 sampai 5 = 15
```

Dari berbagai contoh aplikasi di atas kiranya cukup jelas nampak ciri-ciri algoritma program baik secara Teknik Rekursif maupun Teknik

Iteratif. Apakah semua problem bisa dipecahkan secara Teknik Rekursif ? Ternyata tidak semuanya. Apakah juga harus diselesaikan secara Teknik Iteratif ? Ternyata juga tidak. Sebagai contoh Jumlah Bilangan Integer Positif bisa juga diselesaikan dengan rumus : n x (n + 1) / 2 atau $(n^2 + n) / 2$ jadi tidak perlu Iteratif.

Perhatikan beberapa kasus berikut ini pun dapat diselesaikan dengan rekursif:

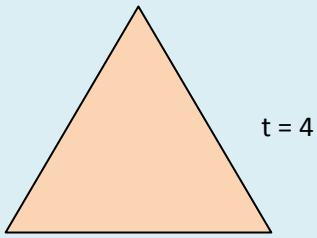
1. Menggambar karakter '*' sebanyak n

```
**********
n=11

Basic          : n=0
    tidak mencetak apa-apa
Rekursif       : n>0
    gambar karakter '*' sebanyak (n-1)
    cetak '*'
```

2. Menggambar persegi panjang dengan lebar l dan tinggi t


```
Basic          : n=0
    tidak mencetak apa-apa
Rekursif       : n>0
    gambar persegi panjang dengan lebar l dan tinggi (t-1)
    gambar '*" sebanyak l
```

3. Menggambar segitiga/piramid dengan tinggi n mulai kolom k




```
Basic          : n=0
```

- tidak mencetak apa-apa

Rekursif : n>0

- gambar piramid dengan tinggi (n-1) mulai kolom (k+1)
- cetak karakter ' ' sebanyak k
- cetak karakter '*' sebanyak 2n – 1

10.8. Latihan Soal

Jawablah soal latihan dibawah ini dengan baik dan benar.

1. Apa yang dimaksud dengan rekursif, jelaskan
2. Apakan perbedaan antara program menggunakan rekursif dan interatif
3. Dari soal diatas jelaskan kekurangan dan kelebihan kedua program tersebut
4. Buatlah program faktorial dengan rekursif
5. Buatlah program pemangkatan dengan menggunakan rekursif

BAB. 11
GRAFIK

11.1. Pengertian Grafik

Dalam pemrogram C++ untuk membuat grafik kita harus mengenal GUI (Graphics User Interface). Windows yang kita kenal merupakan aplikasi yang berjalan di atas GUI, jadi tampilan yang keluar pasti berbasis grafik. Berbeda dengan DOS, system operasi ini berbasis teks bukan grafik. Untuk input/outpu berbasis grafik windows mempunyai kumpulan fungsi-fungsi di dalam Windows API (*Application Program Interface*) yang sering juga disebut GDI (*Graphics Device Interface*).

Apa keistimewaan GDI ? Pemrogram yang berkecimpung dalam dunia grafik dengan DOS mernah merasakan sulitnya membuat aplikasi grafik yang kompatibel dengan card monitor, karena resolusi card yang berbeda kadang harus menyesuaikan dengan membuat rutin yang berbeda.

Dengan menggunakan GDI seorang pemrogram dapat dengan mudah membuat aplikasi grafik tanpa mempedulikan kompatibilitas dengan card monitor yang dipakai atau perangkat keras yang lain. Hal ini dikarenakan Windows sudah mengatur semuanya, jadi pemrogram tinggal konsentrasi membuat aplikasi grafiknya atau pembuatan aplikasi citra.

Selain itu GDI bersifat *device independent*, artinya tidak tergantung peralatan fisik yang digunakan. Pernyataan untuk membuat lingkaran ke layar monitor, maupun ke printer, atau ke plotter sama. Tentu saja hal ini amat menguntungkan pemrogram karena hanya mengingat satu perintah saja untuk semua peralatan atau berbagai macam peralatan.

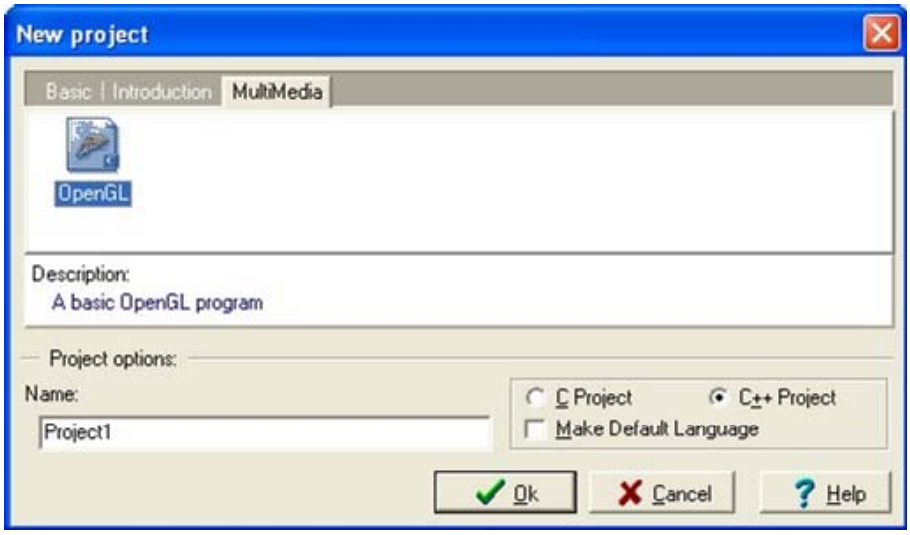
11.2. Grafik Library

Pada pemrograman C++ mempunyai library yang disebut OpenGL. Library ini memanfaatkan Windows API dalam proses pembentukan grafik. Dengan menggunakan OpenGL pembuatan grafik jadi sangat mudah. Namun demikian perlu diketahui bahwa pemrograman menggunakan library harus mengikuti prosedur yang telah ditetapkan oleh pembuatnya.

Pada prinsipnya kita tinggal memodifikasi program yang telah ada, pada bagian mana yang dimodifikasi agar sesuai dengan keperluannya itulah yang harus dipelajari. Untuk lebih jelasnya ikuti langkah-langkah program berikut:

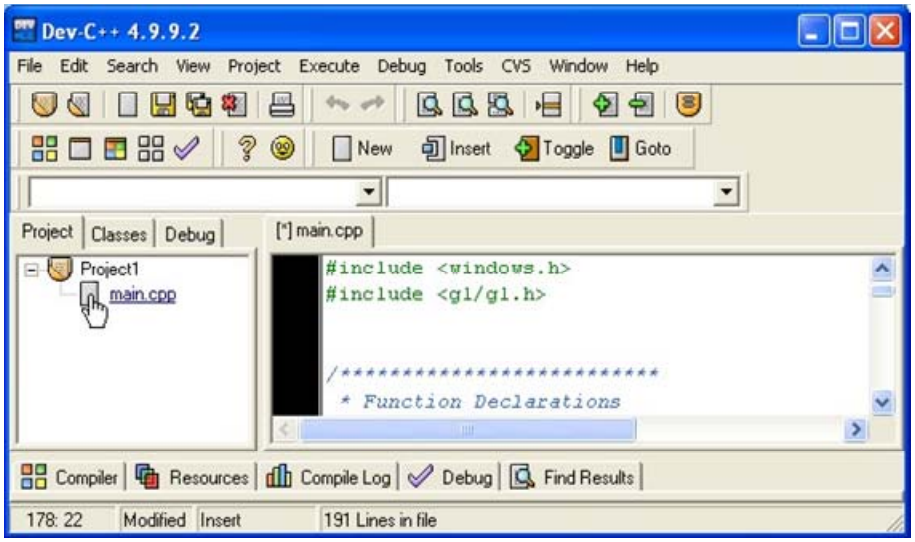
- Pada saat membuat proyek baru harus kita arahkan ke library OpenGL
- Setelah proyek diciptakan maka kita langsung mempunyai main() program yang apabila dijalankan (*run*) langsung ada hasilnya
- Selanjutnya tugas pemrogram tinggal memodifikasi bagian-bagian tertentu disesuaikan dengan rancangan program yang telah ditetapkan

Gambaran pembuatan proyek baru seperti berikut:



Gambar 11.1. pembuatan proyek

Pada saat anda sudah memilih membuat proyek baru maka pilihlan OpenGL sebagai basis proyek ini, seperti terlihat dalam gambar di atas. Setelah OK maka proyek langsung siap dan di dalamnya sudah ada main() program yang siap ditampilkan; program inilah yang siap dimodifikasi. Perhaikan gambar di bawah:



Gambar 11.2. proyek OpenGL

Terlihat pada gambar bahwa sudah ada main() program dan yang penting dalam editor terlihat library yang hendak dipakai untuk membuat grafik ini. Lihat cara penulisan dibawah ini:

```
#include <windows>
#include <gl/gl.h>
```

Dua buah header tersebut yang dibutuhkan untuk menghubungi library pada saat dibutuhkan oleh program dalam membentuk grafik.

11.3. Grafik Sederhana

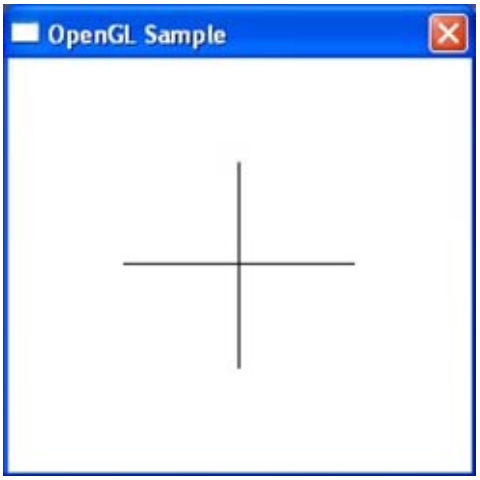
Pengenalan grafik biasanya dimulai dengan membuat garis. Garis di dalam grafik merupakan kumpulan titik-titik yang berupa pixel, resolusi pixel akan mempengaruhi hasil tampilan; makin rapat pixel grafik yang dihasilkan makin halus sehingga kadang kelihatan jadi lebih indah dan menarik. Perintah dalam OpenGL untuk membuat garis:

Program 11.1

```
glClearColor (1.0f, 1.0f, 1.0f, 0.0f);
glClear (GL_COLOR_BUFFER_BIT);

glBegin (GL_LINES);

glColor3f (0.0f, 0.0f, 0.0f);   glVertex2f (0.0f, 0.5f);
glColor3f (0.0f, 0.0f, 0.0f);   glVertex2f (0.0f, -0.5f);

glColor3f (0.0f, 0.0f, 0.0f);   glVertex2f (0.5f, 0.0f);
```

```
glColor3f (0.0f, 0.0f, 0.0f);  glVertex2f (-0.5f, 0.0f);

glEnd ();

SwapBuffers (hDC);
```

Setelah dieksekusi potongan program diatas maka akan menampilan hasil grafik seperti gambar dibawah ini:



Gambar 11.3. Garis

Fungsi utama sehingga dapat menghasilkan grafik seperti gambar di atas adalah:

```
glColor3f (0.0f, 0.0f, 0.0f);  glVertex2f (0.0f, 0.5f);
glColor3f (0.0f, 0.0f, 0.0f);  glVertex2f (0.0f, -0.5f);
```

bisa dipahami bahwa glColor3f(0.0f, 0.0f, 0.0f) merupakan perintah untuk member warna titik-titik dari pixel yang dihasilkan oleh glVertex2f(0.0f, 0.5f). Sebenarnya masingmasing baris pada potongan program di atas merupakan dua perintah bukan satu perintah. Namun karena keduanya saling terkait satu sama lain maka dijadikan satu baris, walaupun jika hendak ditulis pada baris yang berbeda juga tidak mengapa. Penjelasan lebih lanjut, format glColor3f adalah :

```
glColor3f(red, green, blue);
```

Argumen red, green, blue untuk mengaktifkan masing-masing diisi bilangan pecahan 1 (1.0), untuk mematikan diisi bilangan pecahan 0 (0.0). Dapat dilihat contoh di atas semuanya dimatikan berarti titik-titik pixel akan berwarna hitam (seperti dilihat di gambar hasil eksekusi). Kombinasi dari ketiga argument ini akan menghasilkan warna yang indah dan sangat menarik. Perlu diketahui pula bahwa grafik yang terjadi merupakan grafik Cartesian yaitu grafik yang titik nolnya di tengah, oleh karena itu untuk membuat garis tegak dibutuhkan dua glVertex2f yang masing-masing argumen diisi dengan(0.0, 0.5) dan (0.0, -0.5). Begitu juga dengan garis mendatar dibutuhkan dua glVertex2f, tentu saja dengan argumen yang berbeda dalam hal ini (-0.5, 0.0) dan (0.5, 0.0). Dengan memodifikasi program tersebut bisa didapatkan tampilan grafik yang menarik.

Selanjutnya harus ada glBegin() dan glEnd(), karena perintah itu

merupakan awal dan akhir program grafik yang dijalankan. Pada glBegin ada argumen GL_LINES, ini dimaksudkan untuk membuat garis. Sedangkan apabila hendak membuat bentuk yang lain maka perlu disesuaikan, contoh ada GL_TRIANGLES untuk segitiga, GL_QUADS untuk segi empat, atau GL_POLYGON untuk segi banyak.

Perintah glClearColor untuk mengatur warna latar belakang window, dengan format sebagai berikut :

```
glClearColor(red, green, blue, alpha);
```

seperti perintah sebelumnya argumen yang ada bisa diisi dengan angka pecahan dan dapat dikombinasikan sehingga mendapatkan warna yang menarik. Pada contoh program ini sengaja dibuat putih dengan mengkombinasikan red, green, blue semuanya diisi 1 (1.0). Untuk menciptakan atau mendapatkan perubahan argumen (jika diubah) diperlukan perintah:

```
glClear(GL_COLOR_BUFFER_BIT);
```

yang berarti perubahan warna pada perintah sebelumnya disimpan di buffer yang nanti pada saat eksekusi akan ditampilkan.

Program selengkapnya dapat dilihat di bawah ini:

Program 11.2.

```
#include <windows.h>
#include <gl/gl.h>

/*************************
 * Function Declarations
 *************************/
LRESULT CALLBACK WndProc (HWND hWnd, UINT message,
WPARAM wParam, LPARAM lParam);
void EnableOpenGL (HWND hWnd, HDC *hDC, HGLRC *hRC);
void DisableOpenGL (HWND hWnd, HDC hDC, HGLRC hRC);

/*************************
 * WinMain
 *************************/
int WINAPI WinMain (HINSTANCE hInstance,
                    HINSTANCE hPrevInstance,
                    LPSTR lpCmdLine,
                    int iCmdShow)
{
   WNDCLASS wc;
   HWND hWnd;
   HDC hDC;
```

```
HGLRC hRC;
MSG msg;
BOOL bQuit = FALSE;
float theta = 0.0f;

/* register window class */
wc.style = CS_OWNDC;
wc.lpfnWndProc = WndProc;
wc.cbClsExtra = 0;
wc.cbWndExtra = 0;
wc.hInstance = hInstance;
wc.hIcon = LoadIcon (NULL, IDI_APPLICATION);
wc.hCursor = LoadCursor (NULL, IDC_ARROW);
wc.hbrBackground = (HBRUSH) GetStockObject (BLACK_BRUSH);
wc.lpszMenuName = NULL;
wc.lpszClassName = "GLSample";
RegisterClass (&wc);

/* create main window */
hWnd = CreateWindow (
  "GLSample", "OpenGL Sample",
  WS_CAPTION | WS_POPUPWINDOW | WS_VISIBLE,
  0, 0, 256, 256,
  NULL, NULL, hInstance, NULL);

/* enable OpenGL for the window */
EnableOpenGL (hWnd, &hDC, &hRC);

/* program main loop */
while (!bQuit)
{
   /* check for messages */
   if (PeekMessage (&msg, NULL, 0, 0, PM_REMOVE))
   {
      /* handle or dispatch messages */
      if (msg.message == WM_QUIT)
      {
         bQuit = TRUE;
      }
      else
      {
         TranslateMessage (&msg);
         DispatchMessage (&msg);
```

```
      }
    }
    else
    {
      /* OpenGL animation code goes here */

      glClearColor (1.0f, 1.0f, 1.0f, 0.0f);
      glClear (GL_COLOR_BUFFER_BIT);

      glBegin (GL_LINES);
      glColor3f (0.0f, 0.0f, 0.0f);   glVertex2f (0.0f, 0.5f);
      glColor3f (0.0f, 0.0f, 0.0f);   glVertex2f (0.0f, -0.5f);

      glColor3f (0.0f, 0.0f, 0.0f);   glVertex2f (0.5f, 0.0f);
      glColor3f (0.0f, 0.0f, 0.0f);   glVertex2f (-0.5f, 0.0f);
      glEnd ();

      SwapBuffers (hDC);
    }
  }

  /* shutdown OpenGL */
  DisableOpenGL (hWnd, hDC, hRC);

  /* destroy the window explicitly */
  DestroyWindow (hWnd);

  return msg.wParam;
}

/********************
 * Window Procedure
 ********************/
LRESULT CALLBACK WndProc (HWND hWnd, UINT message,
                          WPARAM wParam, LPARAM lParam)
{

  switch (message)
  {
  case WM_CREATE:
    return 0;
  case WM_CLOSE:
    PostQuitMessage (0);
```

```
      return 0;

   case WM_DESTROY:
      return 0;

   case WM_KEYDOWN:
      switch (wParam)
      {
      case VK_ESCAPE:
         PostQuitMessage(0);
         return 0;
      }
      return 0;

   default:
      return DefWindowProc (hWnd, message, wParam, lParam);
   }
}

/*******************
 * Enable OpenGL
 *******************/
void EnableOpenGL (HWND hWnd, HDC *hDC, HGLRC *hRC)
{
   PIXELFORMATDESCRIPTOR pfd;
   int iFormat;

   /* get the device context (DC) */
   *hDC = GetDC (hWnd);

   /* set the pixel format for the DC */
   ZeroMemory (&pfd, sizeof (pfd));
   pfd.nSize = sizeof (pfd);
   pfd.nVersion = 1;
   pfd.dwFlags = PFD_DRAW_TO_WINDOW |
     PFD_SUPPORT_OPENGL | PFD_DOUBLEBUFFER;
   pfd.iPixelType = PFD_TYPE_RGBA;
   pfd.cColorBits = 24;
   pfd.cDepthBits = 16;
   pfd.iLayerType = PFD_MAIN_PLANE;
   iFormat = ChoosePixelFormat (*hDC, &pfd);
   SetPixelFormat (*hDC, iFormat, &pfd);
```

```
  /* create and enable the render context (RC) */
  *hRC = wglCreateContext( *hDC );
  wglMakeCurrent( *hDC, *hRC );
}

/******************
 * Disable OpenGL
 ******************/
void DisableOpenGL (HWND hWnd, HDC hDC, HGLRC hRC)
{
  wglMakeCurrent (NULL, NULL);
  wglDeleteContext (hRC);
  ReleaseDC (hWnd, hDC);
}
```

Dalam program ini terlihat sangat panjang, tetapi itulah yang terjadi apabila memprogram menggunakan library yang tersedia. Namun jika harus membuat sendiri dari awal maka pemrograman kita akan lebih lama. Dengan memanfaatkan library yang ada maka proses pemrograman akan menjadi lebih cepat efektif. Seperti telah diuraikan sebelumnya pemrogram tinggal mencari dimana yang harus dimodifikasi sesuai kebutuhan. Contoh untuk mengubah ukuran window tinggal mengubah argumen di dalam create main window, disana defaultnya adalah 0, 0, 256, 256, angka ini dapat diubah untuk mendapatkan window yang diinginkan. Begitu pula untuk header window dapat disesuaikan.

Lebih lanjut mengenai grafik untuk membuat segitiga misalnya diperlukan tiga buah vertex, sehingga perintah-perintah yang dibutuhkan segabai berikut:

Program 11.3

```
glClear (GL_COLOR_BUFFER_BIT);

glBegin (GL_TRIAGLES);

glColor3f (0.0f, 0.0f, 0.0f);   glVertex2f (0.0f, 0.5f);
glColor3f (0.0f, 0.0f, 0.0f);   glVertex2f (0.5f, -0.5f);
glColor3f (0.0f, 0.0f, 0.0f);   glVertex2f (-0.5f, -0.5f);

glEnd ();
```

Jangan lupa mengubah GL_LINES pada glBegin menjadi GL_TRIANGLES, kalau tidak diubah maka yang muncul akan satu garis saja.

Begitu juga untuk membuat segi empat (kotak), segi lima, segi enam, dan sebagainya. Makin banyak segi dibutuhkan makin banyak vertex.

11.4. Animasi Grafik

Pembuatan animasi bentuk-bentuk dasar dalam OpenGL telah disediakan. Untuk dapat memanfaatkannya kita harus memahami perintah-perintah itu beserta arti argumen-argumen yang dipakainya. Sebagai contoh akan dibuat sebuah program yang menganimasi kotak. Cermati program berikut:

Program 11.4

```
#include <windows.h>
#include <gl/gl.h>


/************************
 * Function Declarations
 ************************/
LRESULT CALLBACK WndProc (HWND hWnd, UINT message,
WPARAM wParam, LPARAM lParam);
void EnableOpenGL (HWND hWnd, HDC *hDC, HGLRC *hRC);
void DisableOpenGL (HWND hWnd, HDC hDC, HGLRC hRC);

/************************
 * WinMain
 ************************/
int WINAPI WinMain (HINSTANCE hInstance,
                    HINSTANCE hPrevInstance,
                    LPSTR lpCmdLine,
                    int iCmdShow)
{
   WNDCLASS wc;
   HWND hWnd;
   HDC hDC;
   HGLRC hRC;
   MSG msg;
   BOOL bQuit = FALSE;
   float theta = 0.0f;

   /* register window class */
```

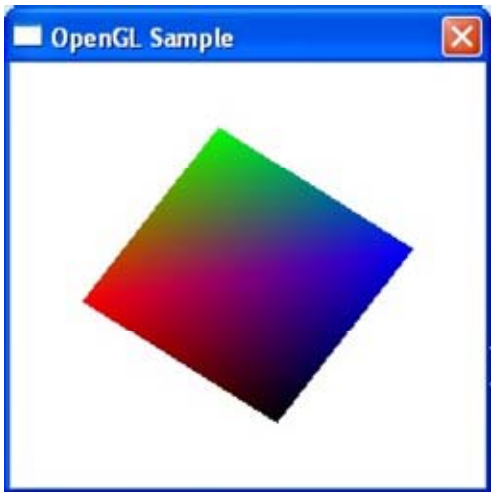
```
wc.style = CS_OWNDC;
wc.lpfnWndProc = WndProc;
wc.cbClsExtra = 0;
wc.cbWndExtra = 0;
wc.hInstance = hInstance;
wc.hIcon = LoadIcon (NULL, IDI_APPLICATION);
wc.hCursor = LoadCursor (NULL, IDC_ARROW);
wc.hbrBackground = (HBRUSH) GetStockObject (BLACK_BRUSH);
wc.lpszMenuName = NULL;
wc.lpszClassName = "GLSample";
RegisterClass (&wc);

/* create main window */
hWnd = CreateWindow (
  "GLSample", "OpenGL Sample",
  WS_CAPTION | WS_POPUPWINDOW | WS_VISIBLE,
  0, 0, 256, 256,
  NULL, NULL, hInstance, NULL);

/* enable OpenGL for the window */
EnableOpenGL (hWnd, &hDC, &hRC);

/* program main loop */
while (!bQuit)
{
   /* check for messages */
   if (PeekMessage (&msg, NULL, 0, 0, PM_REMOVE))
   {
      /* handle or dispatch messages */
      if (msg.message == WM_QUIT)
      {
         bQuit = TRUE;
      }
      else
      {
         TranslateMessage (&msg);
         DispatchMessage (&msg);
      }
   }
   else
   {
      /* OpenGL animation code goes here */
```

```
        glClearColor (1.0f, 1.0f, 1.0f, 0.0f);
        glClear (GL_COLOR_BUFFER_BIT);

        glPushMatrix ();
        glRotatef (theta, 0.0f, 0.0f, 1.0f);
        glBegin (GL_QUADS);
        glColor3f (1.0f, 0.0f, 0.0f);   glVertex2f (-0.5f, 0.5f);
        glColor3f (0.0f, 1.0f, 0.0f);   glVertex2f (0.5f, 0.5f);
        glColor3f (0.0f, 0.0f, 1.0f);   glVertex2f (0.5f, -0.5f);
        glColor3f (0.0f, 0.0f, 0.0f);   glVertex2f (-0.5f, -0.5f);
        glEnd ();
        glPopMatrix ();

        SwapBuffers (hDC);

        theta += 1.0f;
        Sleep (1);
      }
   }

   /* shutdown OpenGL */
   DisableOpenGL (hWnd, hDC, hRC);

   /* destroy the window explicitly */
   DestroyWindow (hWnd);

   return msg.wParam;
}


/********************
 * Window Procedure
 ********************/
LRESULT CALLBACK WndProc (HWND hWnd, UINT message,
                          WPARAM wParam, LPARAM lParam)
{

   switch (message)
   {
   case WM_CREATE:
      return 0;
   case WM_CLOSE:
      PostQuitMessage (0);
```

```
      return 0;

   case WM_DESTROY:
      return 0;

   case WM_KEYDOWN:
      switch (wParam)
      {
      case VK_ESCAPE:
         PostQuitMessage(0);
         return 0;
      }
      return 0;

   default:
      return DefWindowProc (hWnd, message, wParam, lParam);
   }
}

/*******************
 * Enable OpenGL
 *******************/
void EnableOpenGL (HWND hWnd, HDC *hDC, HGLRC *hRC)
{
   PIXELFORMATDESCRIPTOR pfd;
   int iFormat;

   /* get the device context (DC) */
   *hDC = GetDC (hWnd);

   /* set the pixel format for the DC */
   ZeroMemory (&pfd, sizeof (pfd));
   pfd.nSize = sizeof (pfd);
   pfd.nVersion = 1;
   pfd.dwFlags = PFD_DRAW_TO_WINDOW |
     PFD_SUPPORT_OPENGL | PFD_DOUBLEBUFFER;
   pfd.iPixelType = PFD_TYPE_RGBA;
   pfd.cColorBits = 24;
   pfd.cDepthBits = 16;
   pfd.iLayerType = PFD_MAIN_PLANE;
   iFormat = ChoosePixelFormat (*hDC, &pfd);
   SetPixelFormat (*hDC, iFormat, &pfd);
```

```
   /* create and enable the render context (RC) */
   *hRC = wglCreateContext( *hDC );
   wglMakeCurrent( *hDC, *hRC );

}

/******************
 * Disable OpenGL
 ******************/
void DisableOpenGL (HWND hWnd, HDC hDC, HGLRC hRC)
{
   wglMakeCurrent (NULL, NULL);
   wglDeleteContext (hRC);
   ReleaseDC (hWnd, hDC);
}
```

Keluaran program di atas akan menganimasikan kotak dengan warna-warni yang menarik dan berputar berlawanan arah jarum jam. Gambar kotak yang terjadi seperti berikut:



Gambar 11.4. Grafik berwarna warni

Jika diperhatikan atau dipelajari pasti akan Menarik bukan? Warna-warna tersebut dimunculkan dengan mengkombinasikan argumen-argumen pada perintah glColor3f, cermati dalam listing program. Dalam program yang sesungguhnya kotak tersebut selain berwarna-warni juga berputar berlawanan arah putaran jarum jam (*counter clockwise*). Perintah apa yang membuat kotak ini berputar ? Kita lihat program di atas:

```
glRotatef (theta, 0.0f, 0.0f, 1.0f);
```

sesuai namanya perintah di ataslah yang menyebabkan kotak ini berputar. Format perintah ini secara lengkapnya sebagai berikut :

- Argumen pertama theta untuk mengatur kecepatan putar makin besar angkanya makin cepat putaran yang terjadi
- Argumen kedua jika diisi 1 (1.0) maka kotak akan berputar dengan sumbu garis horizontal (mendatar) ke arah tertentu. Untuk memutar ke arah yang berlawanan dari

contoh, argumen diisi angka -1 (-1.0)

- Argumen ketiga juga mirip dengan argumen kedua namun berputarnya terhadap sumbu vertical (tegak lurus)
- Argumen keempat untuk memutar kotak flat dengan sumbu titik nol diagram cartesius. Jika diisi angka 1 (1.0) maka akan berputar berlawanan arah putaran jarum jam (*counter clockwise*), sedangkan untuk berputar searah dengan arah putaran jarum jam (*clockwise*) diisi angka -1 (-1.0). Hal yang menarik jika argumen ini diisi angka 0 (0.0) maka kotak akan *zoom in* dan *zoom out* bergantian.
- Kombinasi dari argumen-argumen yang ada akan membuat variasi pergerakan yang sangat menarik

Untuk mengatur laju perputaran animasi yang terjadi bisa dengan memodifikasi theta, atau dengan perintah sleep(argumen). Laju kecepatan putar tergantung pada nilai argumen pada sleep, makin besar angkanya makin lambat bukan makin cepat. Lebih lanjut dengan grafik, jika akan membuat segi enam maka perubahan yang terjadi seperti berikut:

Program 11.5

```
/* OpenGL animation code goes here */

glClearColor (0.0f, 0.0f, 0.0f, 1.0f);
glClear (GL_COLOR_BUFFER_BIT);

glPushMatrix ();
glRotatef (theta, 0.0f, 0.0f, -1.0f);
glBegin (GL_POLYGON);

glColor3f (1.0f, 0.0f, 0.0f);   glVertex2f (-0.3f, 0.54f);
glColor3f (0.0f, 1.0f, 0.0f);   glVertex2f (0.3f, 0.54f);
glColor3f (0.0f, 0.0f, 1.0f);   glVertex2f (0.61f, 0.0f);
glColor3f (0.0f, 1.0f, 1.0f);   glVertex2f (0.3f, -0.54f);
glColor3f (1.0f, 1.0f, 0.0f);   glVertex2f (-0.3f, -0.54f);
glColor3f (1.0f, 1.0f, 1.0f);   glVertex2f (-0.61f, 0.0f);

glEnd ();
glPopMatrix ();

SwapBuffers (hDC);

theta += 2.0f;
Sleep (1);
```

Tentu saja vertex berubah bertambah banyak, banyaknya vertex sesuai dengan jumlah sudut yang dibentuk. Jangan lupa argumen pada glBegin diisi dengan GL_POLYGON sehingga bisa menciptakan segi banyak seperti yang dikehendaki. Selain glRotatef() pengaturan animasi dapat juga dengan glTranslatef(). Dengan perintah ini kita dapat menggerakkan obyek kekiri, kekanan, ke atas, maupun ke bawah. Contoh program sebagai berikut:

Program 11.6

```
#include <windows.h>
#include <gl/gl.h>

/*************************
 * Function Declarations
 *************************/
LRESULT CALLBACK WndProc (HWND hWnd, UINT message,
WPARAM wParam, LPARAM lParam);
void EnableOpenGL (HWND hWnd, HDC *hDC, HGLRC *hRC);
void DisableOpenGL (HWND hWnd, HDC hDC, HGLRC hRC);

/*************************
 * WinMain
 *************************/
int WINAPI WinMain (HINSTANCE hInstance,
                    HINSTANCE hPrevInstance,
                    LPSTR lpCmdLine,
                    int iCmdShow)
{
   WNDCLASS wc;
   HWND hWnd;
   HDC hDC;
   HGLRC hRC;
   MSG msg;
   BOOL bQuit = FALSE;
   float theta = -1.0f;
   float alpha = -1.0f;

   /* register window class */
   wc.style = CS_OWNDC;
   wc.lpfnWndProc = WndProc;
   wc.cbClsExtra = 0;
```

```
wc.cbWndExtra = 0;
wc.hInstance = hInstance;
wc.hIcon = LoadIcon (NULL, IDI_APPLICATION);
wc.hCursor = LoadCursor (NULL, IDC_ARROW);
wc.hbrBackground = (HBRUSH) GetStockObject (BLACK_BRUSH);
wc.lpszMenuName = NULL;
wc.lpszClassName = "GraphGL";
RegisterClass (&wc);

/* create main window */
hWnd = CreateWindow (
  "GraphGL", "OpenGL Graphics Animation",
  WS_CAPTION | WS_POPUPWINDOW | WS_VISIBLE,
  0, 0, 512, 512,
  NULL, NULL, hInstance, NULL);

/* enable OpenGL for the window */
EnableOpenGL (hWnd, &hDC, &hRC);

/* program main loop */
while (!bQuit)
{
   /* check for messages */
   if (PeekMessage (&msg, NULL, 0, 0, PM_REMOVE))
   {
      /* handle or dispatch messages */
      if (msg.message == WM_QUIT)
      {
         bQuit = TRUE;
      }
      else
      {
         TranslateMessage (&msg);
         DispatchMessage (&msg);
      }
   }
   else
   {
      /* OpenGL animation code goes here */

      glClearColor (1.0f, 1.0f, 1.0f, 1.0f);
      glClear (GL_COLOR_BUFFER_BIT);
```

```
        glPushMatrix ();

        glTranslatef (theta, alpha, 0.0f);
        glColor3f (1.0f, 0.0f, 1.0f);
        glRectf(-0.2, 0.2, 0.2, -0.2);

        glPopMatrix ();

        SwapBuffers (hDC);

        if (theta >= 1.0)
        {
          theta = -1.0f;
          alpha = -1.0f;
        }
        else if (theta <= 1.0)
        {
           theta += 0.01f;
           alpha += 0.01f;
        }
        Sleep (10);
     }
  }

  /* shutdown OpenGL */
  DisableOpenGL (hWnd, hDC, hRC);

  /* destroy the window explicitly */
  DestroyWindow (hWnd);

  return msg.wParam;
}


/********************
 * Window Procedure
 ********************/
LRESULT CALLBACK WndProc (HWND hWnd, UINT message,
                  WPARAM wParam, LPARAM lParam)
{

  switch (message)
  {
```

```
   case WM_CREATE:
      return 0;
   case WM_CLOSE:
      PostQuitMessage (0);
      return 0;

   case WM_DESTROY:
      return 0;

   case WM_KEYDOWN:
      switch (wParam)
      {
      case VK_ESCAPE:
         PostQuitMessage(0);
         return 0;
      }
      return 0;

   default:
      return DefWindowProc (hWnd, message, wParam, lParam);
   }
}

/*******************
 * Enable OpenGL
 *******************/
void EnableOpenGL (HWND hWnd, HDC *hDC, HGLRC *hRC)
{
   PIXELFORMATDESCRIPTOR pfd;
   int iFormat;

   /* get the device context (DC) */
   *hDC = GetDC (hWnd);

   /* set the pixel format for the DC */
   ZeroMemory (&pfd, sizeof (pfd));
   pfd.nSize = sizeof (pfd);
   pfd.nVersion = 1;
   pfd.dwFlags = PFD_DRAW_TO_WINDOW |
     PFD_SUPPORT_OPENGL | PFD_DOUBLEBUFFER;
   pfd.iPixelType = PFD_TYPE_RGBA;
   pfd.cColorBits = 24;
   pfd.cDepthBits = 16;
```

```
   pfd.iLayerType = PFD_MAIN_PLANE;
   iFormat = ChoosePixelFormat (*hDC, &pfd);
   SetPixelFormat (*hDC, iFormat, &pfd);

   /* create and enable the render context (RC) */
   *hRC = wglCreateContext( *hDC );
   wglMakeCurrent( *hDC, *hRC );

}

/******************
 * Disable OpenGL
 ******************/
void DisableOpenGL (HWND hWnd, HDC hDC, HGLRC hRC)
{
   wglMakeCurrent (NULL, NULL);
   wglDeleteContext (hRC);
   ReleaseDC (hWnd, hDC);
}
```

Hasil eksekusi program di atas akan menggerakkan sebuah kotak berjalan secara diagonal dari pojok kiri bawah ke kanan atas secara terus menerus. Akan tetapi jika ada penekanan mouse di taskbar maka kotak berhenti, jika penekanan dilepas kotak jalan lagi. Untuk selesai bisa menggunakan tombol ESCape.

11.5. Dasar-dasar Game

Untuk memahami pembuatan game contoh program di atas bisa dimodifikasi bagian glTranslatef(). Argumen dalam glTranslatef () dibuat variabel theta untuk argumen pertama, alpha untuk argumen kedua. Theta untuk mengatur gerak horisontal dan alpha untuk mengatur gerak vertikal. Dengan mengubah-ubah nilai variabel tersebut kita bisa menggerakkan obyek tersebut secara bebas sesuai dengan kehendak. Pada saat obyek bergerak bisa juga diatur perubahan warna atau dikombinasi sehingga menjadi warna-warni. Untuk lebih jelasnya ikuti program berikut:

Program 11.7

```
#include <windows.h>
#include <gl/gl.h>

/*************************
 * Function Declarations
 *************************/
LRESULT CALLBACK WndProc (HWND hWnd, UINT message,
```

```
WPARAM wParam, LPARAM lParam);
void EnableOpenGL (HWND hWnd, HDC *hDC, HGLRC *hRC);
void DisableOpenGL (HWND hWnd, HDC hDC, HGLRC hRC);

/************************
 * WinMain
 ************************/
int WINAPI WinMain (HINSTANCE hInstance,
                    HINSTANCE hPrevInstance,
                    LPSTR lpCmdLine,
                    int iCmdShow)
{
   WNDCLASS wc;
   HWND hWnd;
   HDC hDC;
   HGLRC hRC;
   MSG msg;
   BOOL bQuit = FALSE;
   float theta = -0.8f;
   float alpha = -0.8f;
   int count = 0;
   float r, g, b;

   /* register window class */
   wc.style = CS_OWNDC;
   wc.lpfnWndProc = WndProc;
   wc.cbClsExtra = 0;
   wc.cbWndExtra = 0;
   wc.hInstance = hInstance;
   wc.hIcon = LoadIcon (NULL, IDI_APPLICATION);
   wc.hCursor = LoadCursor (NULL, IDC_ARROW);
   wc.hbrBackground = (HBRUSH) GetStockObject (BLACK_BRUSH);
   wc.lpszMenuName = NULL;
   wc.lpszClassName = "GraphGL";
   RegisterClass (&wc);

   /* create main window */
   hWnd = CreateWindow (
    "GraphGL", "OpenGL Graphics",
    WS_CAPTION | WS_POPUPWINDOW | WS_VISIBLE,
    0, 0, 512, 512,
    NULL, NULL, hInstance, NULL);
```

```
/* enable OpenGL for the window */
EnableOpenGL (hWnd, &hDC, &hRC);

/* program main loop */
while (!bQuit)
{
  /* check for messages */
  if (PeekMessage (&msg, NULL, 0, 0, PM_REMOVE))
  {
    /* handle or dispatch messages */
    if (msg.message == WM_QUIT)
    {
      bQuit = TRUE;
    }
    else
    {
      TranslateMessage (&msg);
      DispatchMessage (&msg);
    }
  }
  else
  {
    /* OpenGL animation code goes here */

    glClearColor (1.0f, 1.0f, 1.0f, 1.0f);
    glClear (GL_COLOR_BUFFER_BIT);

    glPushMatrix ();

    glTranslatef (theta, alpha, 0.0f);
    glColor3f (r, g, b);
    glRectf(-0.2, 0.2, 0.2, -0.2);

    glPopMatrix ();

    SwapBuffers (hDC);

    if (count > 160)
    {
     theta -= 0.01f;
     alpha -= 0.01f;
     if (count == 320)
     {
```

```
            count = 0;
            theta = -0.8f;
            alpha = -0.8f;
          }
          else
            count += 1;

          r = 0.0f;
          g = 1.0f;
          b = 0.0f;
        }
        else if (count <= 160)
        {
          theta += 0.01f;
          alpha += 0.01f;
          count += 1;
          r = 1.0f;
          g = 0.0f;
          b = 1.0f;
        }

        Sleep (10);
      }
    }

    /* shutdown OpenGL */
    DisableOpenGL (hWnd, hDC, hRC);

    /* destroy the window explicitly */
    DestroyWindow (hWnd);

    return msg.wParam;
}

/********************
 * Window Procedure
********************/
LRESULT CALLBACK WndProc (HWND hWnd, UINT message,
                          WPARAM wParam, LPARAM lParam)
{

    switch (message)
    {
```

```
   case WM_CREATE:
      return 0;
   case WM_CLOSE:
      PostQuitMessage (0);
      return 0;

   case WM_DESTROY:
      return 0;

   case WM_KEYDOWN:
      switch (wParam)
      {
      case VK_ESCAPE:
         PostQuitMessage(0);
         return 0;
      }
      return 0;

   default:
      return DefWindowProc (hWnd, message, wParam, lParam);
   }
}

/********************
 * Enable OpenGL
 ********************/
void EnableOpenGL (HWND hWnd, HDC *hDC, HGLRC *hRC)
{
   PIXELFORMATDESCRIPTOR pfd;
   int iFormat;

   /* get the device context (DC) */
   *hDC = GetDC (hWnd);

   /* set the pixel format for the DC */
   ZeroMemory (&pfd, sizeof (pfd));
   pfd.nSize = sizeof (pfd);
   pfd.nVersion = 1;
   pfd.dwFlags = PFD_DRAW_TO_WINDOW |
     PFD_SUPPORT_OPENGL | PFD_DOUBLEBUFFER;
   pfd.iPixelType = PFD_TYPE_RGBA;
   pfd.cColorBits = 24;
   pfd.cDepthBits = 16;
```

```
   pfd.iLayerType = PFD_MAIN_PLANE;
   iFormat = ChoosePixelFormat (*hDC, &pfd);
   SetPixelFormat (*hDC, iFormat, &pfd);

   /* create and enable the render context (RC) */
   *hRC = wglCreateContext( *hDC );
   wglMakeCurrent( *hDC, *hRC );

}

/******************
 * Disable OpenGL
 ******************/
void DisableOpenGL (HWND hWnd, HDC hDC, HGLRC hRC)
{
   wglMakeCurrent (NULL, NULL);
   wglDeleteContext (hRC);
   ReleaseDC (hWnd, hDC);
}
```

Jika program dijalankan maka kotak akan bergerak secara diagonal dari pojok kiri bawah ke arah pojok kanan atas dengan warna ungu, setelah sampai di pojok kanan atas maka akan berbalik arah menuju pojok kiri bawah lagi dengan warna hijau, begitu seterusnya bolak-balik. Akan tetapi jika kursor menunjuk taskbar dan tombol mouse ditekan maka gerak obyek akan berhenti dan jika tombol dilepas lagi maka obyek tersebut gergerak lagi.

Selain argumen-argumen glTranslatef() yang dimodifikasi, argumen glColor3f() juga dibuat variabel sehingga bisa dimainkan warnanya menjadi dinamis tidak monoton. Penambahan variabel pada perintah glColor3f() argumen pertama r, kedua g, ketiga b, sehingga formatnya menjadi glColor3f(r, g, b). Modifikasi yang lain yaitu menambah variabel count untuk menghitung banyaknya gerak. Hal ini perlu ditambahkan karena pengaturan gerak melalui variabel count ini. Contoh ini merupakan dasar pembuatan game yang harus dipelajari untuk bisa dikembangkan dikemudian hari. Bagian yang penting dicermati adalah trik pengaturan geraknya, bagian ini merupakan hal tersulit karena harus menemukan algoritma yang tepat dan efisien.

11.6. Latihan Soal

Jawablah soal latihan dibawah ini dengan baik dan benar.

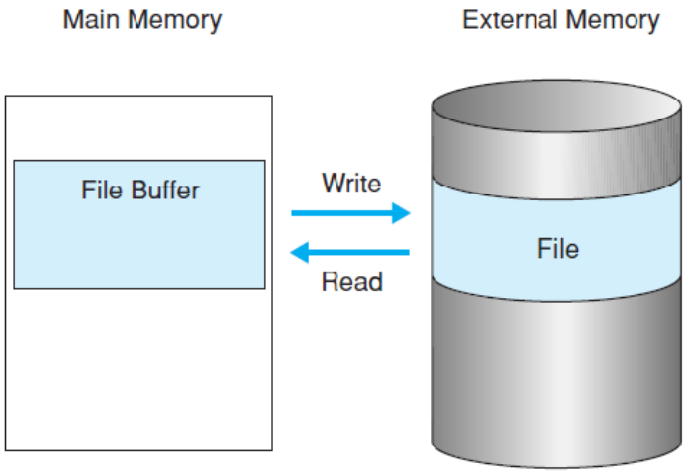
1. Apa yang dimaksud dengan grafik
2. Dalam bahasa c++ terdapat library opengl, apa yang dimaksud dengan istilah tersebut
3. Apa keuntungan dan kekurangan menggunakan library opengl
4. Buatlah program grafik sederhana untuk menampilkan garis
5. Buatlah program grafik sederhana untuk menampilkan poligon
6. Buatlah program animasi sederhana dengan memanfaatkan library opengl

BAB 12
OPERASI FILE

12.1. Pengertian File

File atau Berkas komputer atau berkas adalah entitas dari data yang disimpan di dalam sistem berkas yang dapat diakses dan diatur oleh pengguna. File juga dapat didefinisikan sebagai kumpulan informasi yang biasanya disimpan dalam sebuah disk komputer, dimana informasi disimpan dalam bentuk file-file yang dikemudian hari dapat diambil kembali.

Sebuah berkas memiliki nama yang unik dalam direktori dimana ia berada. Alamat direktori dimana suatu berkas ditempatkan diistilahkan dengan ***path.*** File atau berkas diorganisasikan dari sejumlah record. Masing-masing record bisa terdiri dari satu atau beberapa field. Setiap field terdiri dari satu atau beberapa byte.



Gambar 12.1. Berkas/file

Sistem berkas akan memberikan sebuah nama terhadap sebuah berkas agar dapat dikelola dengan mudah. Meski oleh sistem berkas penamaan dilakukan dengan menggunakan angka-angka biner, sistem operasi dapat menerjemahkan angka-angka biner tersebut menjadi karakter yang mudah dipahami.

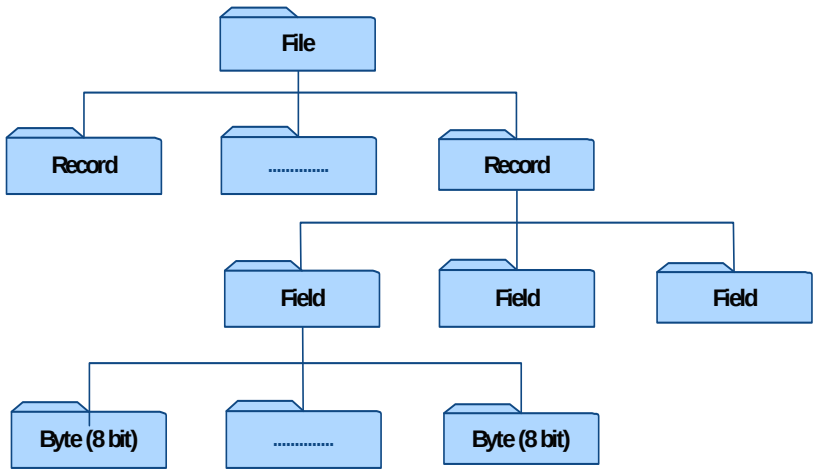
Sebuah berkas berisi aliran data (*data stream*) yang berisi sekumpulan data yang saling berkaitan serta atribut berkas (yang bersifat wajib atau opsional), yang kadang-kadang disebut *properties* yang berisi informasi yang berkaitan dengan berkas yang bersangkutan. Informasi mengenai kapan sebuah berkas dibuat adalah contoh dari atribut berkas.

Ukuran sebuah berkas umumnya direpresentasikan dalam satuan *byte*. Jika bilangan terlalu besar untuk direpresentasikan dalam satuan byte, maka dapat menggunakan satuan KB (*Kilobyte*, yang berarti 1,024 byte), MB (*Megabyte*, yang berarti 1,048,576 byte), GB (*Gigabyte*, yang berarti 1,073,741,824 byte), dan TB (*Terabyte*, yang berarti 1,099,511,627,776 byte). Dalam mekanisme penyimpanan berkas, komputer akan menyimpan berkas dalam dua jenis ukuran: ukuran fisik dan ukuran logis. Ukuran fisik berkas merujuk kepada ukuran aktual dari berkas, yakni berapa banyak byte yang terdapat di dalam berkas. Sementara ukuran logis merujuk kepada jumlah ruangan yang dialokasikan oleh sistem berkas untuk menempatkan berkas yang bersangkutan di dalam media penyimpanan.

Ukuran berkas fisik umumnya lebih besar dibandingkan dengan ukuran berkas logisnya. Sebagai contoh, untuk mengalokasikan berkas berukuran logis 5125 byte, dalam sebuah media penyimpanan yang diformat dengan sistem berkas yang menggunakan ukuran unit

alokasi 4096 byte, komputer akan mengalokasikan dua buah unit alokasi, yang berukuran 4096 dan 4096, sehingga menghabiskan 8192 byte. Meski ukuran logis berkas tersebut 5125 *byte*, komputer mengalokasikan 8192 byte, membuat 3067 byte tidak digunakan (disebut sebagai *wasted space* atau *slack space*).

Kebanyakan sebuah program melibatkan media disk sebagai tempat untuk membaca atau menyimpan data. Data sendiri disimpan dalam disk dalam suatu kesatuan yang disebut file. Suatu file merupakan organisasi dari sejumlah record. Masing-masing record dapat terdri dari satu atau beberapa field dan setiap field terdiri dari satu atau beberapa byte data. Satu byte data terdiri dari susuan 8 bit data. Struktur data dari fle ditunjukkan pada gambar berikut dibawah ini:



Gambar 12.2. Struktur Data Dari File

Dalam C++ file adalah sebuah stream yang disimpan dalam media penyimpanan luar. Karena merupakan sebuah stream, operasi yang berlaku pada stream berlaku juga untuk file. Stream adalah suatu logika device yang menghasilkan dan menerima informasi atau wadah yang digunakan untuk menampung keluaran dan menampung aliran data.

12.2. Class stream

Stream (aliran) merupakan nama yang secara umum diberikan untuk sebuah aliran data. Dalam C++ stream ditunjukan oleh sebuah objek dari class khusus, dimana telah kita ketahui cin dan cout merupakan abjek stream. Perbedaan stream yang digunakan untuk menunjukan perbedaan beberapa macam dari aliran data. Sebagai contoh sebuah class ifstream merepresentasikan aliran data dari masukan file-file disk.

Programer C mungkin akan heran dengan melihat kelebihan-kelebihan menggunakan class stream untuk operasi I/O, sebagai ganti dari fungsi-fungsi C sebelumnya seperi printf() dan scanf() dan operasi file seperti fprintf(), fscanf() dan lain sebagainya. Satu alasannya adalah kesederhanaan. Jika kita pernah menggunakan format karakter %d anda baru akan menyadarinya. Disana ternyata tidak ditemui format dalam stream semenjak sebuah objek siap ditunjukan pada tampilan komputer.

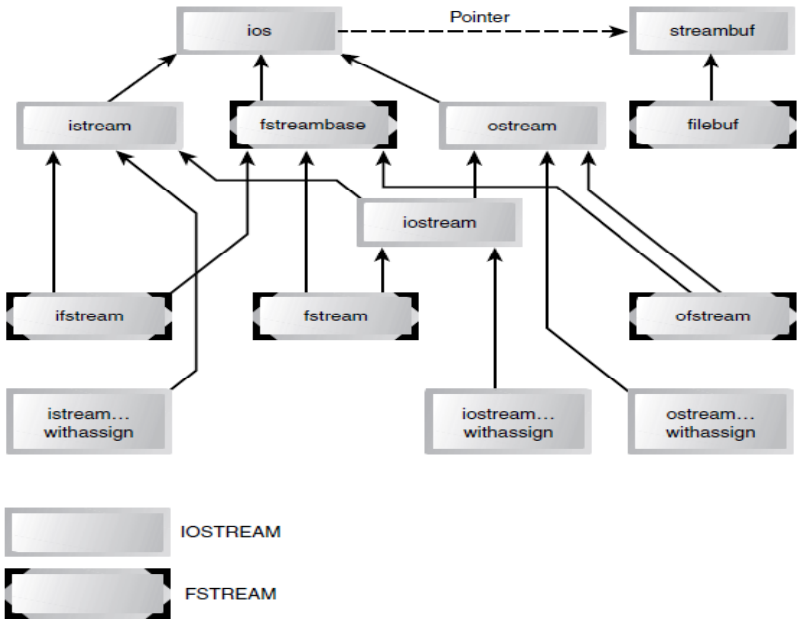
Alasam lain adalah bahwa kamu dapat memberi beban pada sebuah operator dan fungsi seperti operator insertion (<<) dan extraction (>>), dimana untuk dapat bekerja dengan class tersebut anda harus membuatnya. Hal ini akan membuat class anda bekerja pada jalan yang sama sebagai satu-kesatuan tipe. Hal ini jelas membuat programer lebih mudah dan lebih terbebas dari kesalahan.

Anda mungkin heran jika stream I/O ternyata penting jika anda merancang progra dalam sebuah lingkungan dengan Graphic user Interface (GUI) seperti halnya windows. Dengan menggunakan GUI ini secara langsung teks keluaran ditampilkan pada screen . Anda juga masih dan selalu membutuhkan mengenai stream pada bahasa C++. Hal ini akan terjadi karena mereka tahu jalan terbaik untuk menulis data menjadi file dan juga memformat data yang aka digunakan kemudian dalam teks I/O atau elemen GUI lainnya.

12.3. Hirarki class stream

Sebuah class stream diatur dalam hirarki yang agak komplek. Dengan menggunakan class ini, kita dapat membuat tambahan class dengan menggunakan beberapa class istream. Operator turunan >> merupakan bagian dari class stream, dan operator insertion << adalah bagian dari class ostream. Keduanya merupakan class-class yang diturunkan dari class ios. Sebuah objek cout merepresentasikan keluaran stream keluaran, yang biasanya diarahkan pada tampilan monitor. Obyek tersebut sudah dikenal sebagai class penentu ostream. Sama halnya dengan cin sebagai obyek penentu istream yang diturunkan dari class istream.

Gambar dibawah ini menunjukan pengaturan class-class yang paling penting.



Gambar 12.3. Hirarki Class Stream

Class-class tersebut diatas digunakan sebagai masukan dan keluaran dan menampilkan pada layar monitor dan keboard dideklarasikan oleh file header IOSTREAM. Class-class yang digunakan secara khusus untuk input atau ouput file pada disk dideklarasikan oleh file header FSTREAM. Gambar diatas menunjukan class-class yang mana didalamnya dapat ditemukan dua file header tersebut. Selain itu juga ditunjukan beberapa manipulator yang dideklarasikan dalam IOMANIP dan dalam memori, class dideklarasikan dalam STRSTREAM.

Seperti yang dapat dilihat pada gambar diatas, class ios yang merupakan class dasar pada sebuah hirarky. Class tersebut berisi beberapa konstanta atau keanggotaan fungsi umum sebagai operasi masukan dan keluaran. Class ios juga berisi senuah pointer yang digunakan sebagai class streambuf. Dimana hal tersebut berisi buffer memori aktual pada saat data dibaca atau ditulis. Biasanya programmer tidak memikirkan class streambuf, karena hal tersebut akan ditangani secara otomatis oleh class lain.

Class istream dan ostream diturunkan dari ios dan akan digunakan sebagai masukan dan keluaran secara terus-menerus. Class istream berisi seperti fungsi get(), getline(), read() dan operator ekstrasi overloded (>>), sedangkan ostream berisi put() dan write() serta operator insersi overloaded (<<).

Class iostream adalah diturunkan dari dua class istream dan ostream melalui multiple inheritance. Class yang diturunkan tersebut akan digunakan oleh beberapa piranti seperti file pada disk, yang mungkin

akan dibuka sebagai masukan dan keluaran pada saat yang sama. Tiga class istream_withassign, ostream_withassign dan iostream_withassign diwariskan dari istream, ostream dan iostream secara berturut-turut.

Class ios adalah kakek dari semua class-class stream dan berisi secara umum dari hal-hal yang penting yang dibutuhkan untuk mengoperasikan stream pada C++. Ada tiga hal penting yaitu pengaturan flag, flag status kesalahan dan file mode operasi. Pengaturan flag merupakan sekumpulan enum yang didefinisikan dalam ios. Hal tersebut merupakan kondisi seperti saklar on atau off yang memilih

12.4. File Input/Output C++

Input/output file pada bahasa C++ ditentukan oleh tiga class yaitu class istream untuk menangani masukan file, class ostream untuk menangani keluaran file serta class iostream untuk menangani file masukan atau keluaran. Dalam C++ operasi ini berkenaan bahwa file sebagai stream dan saat itu pula file dianggap sebagai aliran byte data. Empat class variabel akan secara otomatis dibuat ketika program dimulai. Perhatikan tabel dibawah ini:

Tabel 12.1. Variabel class I/O pada C++ yang dikenal

VARIABEL	DIGUNAKAN
cin	Console input (standar input)
cout	Console output (standar output)
cerr	Console error (standar error)
clog	Console log

Variabel-variabel tersebut di definisikan dalam file standar include <iostream>. Pada umumnya cin digunakan oleh keyboard dan cout, ceer dan clog digunakan untuk menampilkan ke screen monitor. Kebanyakan sistem operasi memperbolehkan programer mengubah arah melalui I/O. sebagai contoh perintah dibawah ini:

```
my_prog <file.in
```

ketika dijalankan program dengan nama my_prog dan tugas yang diberikan cin adalah file.in. ketika sedang mengerjakan operasi file pada disk, diharuskan menggunakan versi file pada class stream. Dalam hal ini ifstream, ofstream dan fstream akan didefinisikan dalam file include <fstream>.

Seandainya programmer ingin membaca 100 angka yang berurutan dari sebuah file dengan nama number.dat misalnya, maka programmer harus mendeklarasikan variabel masukan file sebagai berikut:

```
ifstream data_file;
```

kemudian yang dibutuhkan adalah C++ mengambil file pada hardisk yang digunakan. Hal ini dikerjakan melalui dengan membuka fungsi keanggotaan sebagai berikut:

```
data_file.open("angka.dat");
```

sekarang programmer dpat membaca file mengunakan statement yang sama untuk membaca yaitu dengan cin. Perhatikan program dibawah ini:

```
for (i = 0; i < 100; ++i) {
   assert(i >= 0);
   assert(i< sizeof(data_array)
/sizeof(data_array[0]));
   data_file >> data_array[i];
}
```

Terakhir yang dibutuhkan untuk mengetahui bahwa sistem I/O telah mengerjakannya dan kemudian menuliskan sebagai berikut:

```
data_file.close( );
```

instruksi tersebut menutup program yang kemudian dapat menggunakan program lagi. C++ memperbolehkan membuka panggilan yang bervariasi denga construktor, sebagai contoh mengganti pada penulisan

```
ifstream data_file;   // File dibaca dari
```

```
data_file.open("angka.dat");
```

dapat ditulis dengan

```
ifstream data_file("angka.dat");
```

apalagi sebuah destruktor dengan otomatis memanggil close. Pada kasus tertentu jika sebuah file number.dat hilang, bagaimana dan dimana programer akan memberitahukan permasalahan tersebut?. Maka keanggotaan fungsi akan kembali dengan pesan kesalahan jika pada fungsi tersebut terdapat masalah, atau kesalahan yang mungkin terjadi. Sehigga untuk menguji apakah terdapat kesalahan atau tidak programer perlu menuliskan potongan program dibawah ini:

```
if (data_file.bad( )) {
   cerr << "Unable to open
numbers.dat\n";
   exit (8);
}
```

Program tersebut dibawah ini merupakan program pembacaan angka

Program 12.1

```
#include<conio.h>
#include <iostream>
#include <fstream>
#include <cstdlib>

using namespace std;

int main( )
```

```
{
   const int DATA_SIZE = 100;              // Jumlah item dalam data
   int data_array[DATA_SIZE];              // ukuran data
   ifstream data_file("angkan.dat");       // file masukan
   int i;                                  //penghitung  Loop

   if (data_file.bad(  )) {
       cerr << "Error: tidak dapat membuka file angka.dat\n";
      exit (8);
   }

   for (i = 0; i < DATA_SIZE; ++i) {
      assert(i >= 0);
      assert(i < sizeof(data_array)/sizeof(data_array[0]));

      data_file >> data_array[i];
   }

   int total;                              // Total angka

   total = 0;
   for (i = 0; i < DATA_SIZE; ++i) {
      assert(i >= 0);
      assert(i < sizeof(data_array)/sizeof(data_array[0]));

      total += data_array[i];
   }

    cout << "Total dari semua angka adalah: " << total << '\n';

    getch();
    return (0);
}
```

Jika programmer ingin membaca baris dari data, maka harus menggunakan fungsi getline. Dimana hal tersebut didefinisikan pada potongan program dibawah ini:

```
istream& getline( istream& input_file,
                  string& the_string);
istream& getline( istream& input_file,
                  string& the_string, char
delim)
```

fungsi tersebut membaca baris yag disimpannya dalam string. Sebuah fungsi referensi kembali menuju input stream. Bentuk kedua dari fungsi ini memperbolehkan programmer menuju baris khusus atau batas

akhir. Jika tidak dikehendaki maka programer bisa menggunakan pengaturan default untuk menuju baris baru (‘\n’)

12.5. Pembacaan String

Untuk membaca gaya bahasa string pada C, dapat menggunakan fungsi getline. Fungsi ini dapat didefinisikan sebagai berikut:

```
istream& getline(char *buffer, int len, char delim = '\n')
```

dimana parameter-parameter pada fungsi ini antara lain:

1. Buffer, gaya bahasa string, dimana string disimpan setelah dilakukan pembacaan
2. Len, panjang buffer yang digunakan dalam byte. Fungsi ini akan membaca sampai len-1 byte data menuju buffer. Parameter ini biasanya disebut dengan sizeof(buffer)
3. Delim, yaitu karakter yang digunakan.

Fungsi ini akan mengembalikan referensi menjadi file masukan. Kemudian fungsi ini akan membaca terus-menerus sampai ditemukan tanda karakter akhir baris (‘\n’). karakter tanda akhir baris ini tidak disimpan dalam buffer, yang disimpan sebagai akhir string adalah jika ditemukan tanda bahwa string telah berakhir (‘\0’). Perhatikan contoh dibawah ini:

```
char buffer[30];
cin.getline(buffer, sizeof(buffer));
```

Fungsi untuk file keluaran adalah sama dengan file masukan, sebagai contoh pada deklarasi dibawah ini:

```
ofstream out_file("out.dat");
```

deklarasi diatas merupakan membuat/menulis file dengan nama out.dat dan dalam penulisan nama juga diperbolehkan mnulis dengan nama seperti out_file. Pada kenyataanya, construktor dapat mengambil dua argumen tambahan. Definisi penuh dari file keluaran construktor adalah:

```
ofstream::ofstream(const char *name, int mode= ios::out,
                   int prot = filebuf::openprot);
```

dimana parameter-parameter pada fungsi ini adalah:

- Name, nama dari nama file yang ditulis
- Mode, merupakan sekumpulan flag/tanda O merah bersama-sama bahwa hal tersebut merupakan mode open. Flag ios::out dibutuhkan sebagai file keluaran. Flag-flag lain yang dibutuhkan seperti dalam tabel dibawah ini:

Tabel 12.2.Open Flag

FLAG	ARTI
ios::app	menambahkan data ke akhir output file.
ios::ate	Pergi ke akhir file ketika dibuka.
ios::in	Membuka untuk masukan (harus diberikan kepada fungsi buka ifstream variabel).
ios::out	Buka file untuk output (harus diberikan kepada fungsi anggota buka ofstream variabel).
ios::binary	File binary (jika tidak ada, maka file dibuka sebagai file ASCII).
ios::trunc	Membuang isi yang ada saat membuka file untuk menulis.
ios::nocreate	Mengalami gagal jika file tersebut tidak ada. (Output file saja. Membuka sebuah file input selalu gagal jika tidak ada file.).
ios::noreplace	Jangan menimpa file yang ada. Jika file ada, menyebabkan rusak ketika buka.

- Prot, file protection dimana hal ini tergantung pada sistem operasi yang digunakan untuk menentukan mode proteksi file tersebut. Misalnya pada UNIX proteksi default sampai 0644 untuk baca/tulis sendiri, baca pada group maupun penulisan lainnya, sedangkan MS-DOS/windows defaultnya adalah 0 dalam kondisi file normal.

Perhatikan contoh pernyataan dibawah ini:

```
ofstream out_file("data.new",
ios::out| ios::binary| ios::nocreate|
            ios::app);
```

potongan pogram diatas dapat diartikan digunakan menambahkan (ios::app) data biner menggunakan (ios::binary), kalau file sudah ada atau telah ditemukan filenya (ios::nocreate) sedangkan data.new merupakan nama file yang akan ditulis.

Pada contoh dibawah terdiri dari fungsi pendek tersebut digunakan untuk menulis pesan pada sebuah file catatan. Sesuatu yang pertama yang dilakukan adalah membuka fungsi untuk operasi output (ios::out), menambahkan catatan (ios::app), dengan menulis dari permulaan sampai akhir penulisan (ios::ate). Setelah fungsi tersebut menulis pesan serta terahir menutup file.

Fungsi ini telah dirancang dengan sederhana, dimana hal tersebut juga tidak memperdulikan mengenai efisiensi dan sebagai hasil dari fungsi adalah sangat tidak efisien. Masalah tersebut dibuka dan ditutup setiap saat memanggil log_message. Membuka file merupakan sebuah operasi yang cukup mahal, dan sesuatu juga harus mempunyai kecepatan lebih tinggi, jika akan membuka file hanya sekali dan mengingatkan tersebut.

Program 12.2.

```
#include <iostream>
#include <fstream>

using namespace std;

void log_message(const string & msg)
{
        ofstream out_file("data.log",
    if (out_file.bad( ))
        out_file << msg << endl;
    return;
}
```

12.6. Routin-routin Konversi File

Untuk menulis angka pada printer atau sebuah terminal, programer harus mengkonversi angka ke karakter. Sebuah printer hanya bisa mengerti karakter bukan angka. Sebagai contoh 567 harus dikonversi menjadi tig karake yaitu: “5”, ”6”, dan “7” dan kemudian dicetak.

Operator << digunakan untuk mengkonversi data menuju karakter dan meletakanya dalam sebuah file. Fungsi ini adalah sangat flexibel. Hal tersebut mengubah integer sederhana menjadi variabel tetap atau yang sesuai dengan stering dalam bentuk bilangan hexadesimal, oktal atau desimal dengan penulisan rata pada margin kiri atau kanan. Jika selama menulis program tidak melakukan pengaturan apapun maka hasil konversi dalam kondisi default.

Keanggotan fungsi setf dan unsetf digunakan untuk mengatur kondisi flag menjadi berlogika set “1" atau clear “0” dimana hal ini digunakan kendali proses konversi. Secara umum bentu penulisan fungsi adalah sebagai berikut:

```
file_var.setf(flags);   // Set flags
file_var.unsetf(flags); // Clear flags
```

Tabel dibawah ini merupakan daftar flag dan penjelasnya.

Tabel 12.3. I/O konversi flag

FLAG	ARTI
ios::skipws	loncati karakter yang mendahului spasi sebagai masukan.
ios::left	Output sebelah kiri dibenarkan.
ios::right	Output seblah kanan dibenarkan.
ios::internal	Numerik keluaran adalah memasukkan padded oleh karakter yang mengisi antara tanda atau dasar karakter dan jumlah itu sendiri.

ios::boolalpha	Gunakan versi karakter yang benar dan salah untuk input dan output.
ios::dec	Output dalam dasar angka 10, format desimal.
ios::oct	Keluaran angka dengan format 8 angka oktal.
ios::hex	Output dalam format angka 16, heksadesimal.
ios::showbase	Mencetak indikator inti pada setiap awal nomor. Misalnya, angka heksadesimal yang diawali dengan "0x".
ios::showpoint	Menunjukkan titik desimal untuk semua angka floating-point apakah ia dibutuhkan.
ios::uppercase	Ketika konversi heksadesimal angka, menunjukkan angka AF sebagai huruf besar.
ios::showpos	Menempatkan tanda positif sebelum semua nomor.
ios::scientific	Mengkonversi semua angka floating-point untuk notasi ilmiah pada output.
ios::fixed	Mengkonversi semua floating-point nomor ke titik tetap pada output.
ios::unitbuf	Buffer output.

Jika pada saat mnulis program menginginkan keluaran berupa format bilangan hexadesimal, maka yang harus dikerjakan adalah menambah dengan menulis potongan program dibawah ini:

```
number = 0x3FF;
cout << "Dec: " << number << '\n';

cout.setf( ios::hex);
cout << "Hex: " << number << '\n';

cout.setf( ios::dec);
```

ketika di"run", maka pprogram akan menghasilkan keluaran sebagai berikut:

```
Dec: 1023
Hex: 3ff
```

Ketika konversi angka ke karakter dilakukan, maka fungsi keanggotaannya ditulis:

```
int file_var.width(int size);
```

suatu saat output harus ditentukan sesuai dengan jumlah karakter yang digunakan. Sebagai contoh misalnya sejumlah 3 angka, secara umum dikonversi menjadi karakter string "3". Jika diatur sampai empat, maka hasilnya menjadi ␣␣␣ dimana ␣ menunjukan satu ruang karakter. Perhatikan fungsi dibawah ini:

```
int file_var.precision(int digits);
```

pernyataan program diatas digunakan untuk mengatur berapa banyak angkat yang akan dimunculkan setelah tanda koma (,). Dan kemudian menuliskan fungsi seperti dibawah ini:

```
char file_var.fill(char pad);
```

program diatas digunakan untuk menentukan karakter, dimana karakter digunakan sebagai penambah angka ketika angka terlalu kecil. Fungsi-fungsi tersebut dapat disebut dengan pengarah, atau programer dapat menggunakan I/O manipulator.

Sebuah I/O manipulator adalah sebuah fungsi khusus yang digunakan dalam pernyataan I/O untuk mengubah format. Jika dipikirkan sebuah manipulator dapat dianalogikan seperti halnya peluru ajaib, dimana ketika ditembakan melalui/masuk atau setelah keluar dari sebuah file, akan mengubah kondisi sebuah file tersebut. Manipulator tidak menyebabkan beberapa keluaran tetapi hanya mengubah kondisinya saja.

Sebagai contoh pada manipulator hex hanya melakukan konversi keluaran menjadi hexadesimal.

```
#include <iostream>

number = 0x3FF;
 cout << "Number is " <<  hex << number
<<  dec << '\n';
```

sebuah file header <iostream> mendefinisikan kumpulan dasar dari sebuah menipulator yang dapat dilihat pada tabel dibawah ini:

Tabel 12.4. I/O manipulator

MANIPULATOR	DESKRIPSI
dec	Keluaran angka dalam format desimal.
hex	Keluaranangka dalam format hexadesimal.
oct	Keluaran angka dalam format oktal.
ws	Loncati space pada masukan.
endl	Keluaran pada end-of-line
ends	Keluaran pada end-of-string (`\0').
flush	Lakukan pembufferan keluaran out.

Selain tebal diatas ada juga manipulator yang didefinisikan ole file header <iomanip> dapat dilihat pada tabel dibawah ini:

Tabel 12.5. Fungsi I/O manipulator pada C

MANIPULATOR	DESKRIPSI
setiosflags(long flags)	Atur pemilih flag konversi.
resetiosflags(long flags)	Reset flag terpilih.
setbase(int base)	Atur konersi dasar menuju 8, 10, atau 16. Urutkan secara umum dec, hex, oct.
setw(int width)	Atur lebar keluaran.
setprecision(int precision)	Atur presisi dari keluaran floating-point.
setfill(char ch)	Atur karakter yang di blok/ditandai.

Untuk lebih jelasnya perhatikan penggunaan I/O manipulator pada program dibawah ini.

program 12.3

```
#include<conio.h>
#include <iostream>
#include <iomanip>

using namespace std;

int main( )
{
   int number = 12;                                    // angka untuk keluarkan
   float real = 12.34;                                 // angka yang nyata

    cout << "12345678901234567890123456789O\n";     // ruler keluaran
    cout << number << "<-\n";
    cout <<  setw(5) << number << "<-\n";
    cout <<  setw(5) <<  setfill('*') << number << "<-\n";
    cout << setiosflags( ios::showpos| ios::left) << setw(5) << number << "<-\n";
    cout << real << "<-\n";
    cout <<  setprecision(1) <<setiosflags( ios::fixed) << real << "<-\n";
    cout <<  setiosflags( ios::scientific) << real << "<-\n";
    getch();
    return (0);
}
```

Keluaran dari program diatas setelah di compile adalah:

```
12345678901234567890123456789O
12<-
  12<-
***12<-
+12**<-
12.34<-
12.3<-
1e+01<-
```

12.7. File Binary dan ASCII

American Standard Code for Information Interchange (ASCII) merupakan kode yang terdiri dari 95 kode karakter dan 33 kode kendali (*lihat lampiran 1*). Dengan kode ASCII memungkinkan manusia

pengguna komputer dapat mengerti karena ACSII merupakan pengkodean yang mengacu pada bahasa manusia.

Misalnya ketika programer menulis file dengan nama prog.cc secara otomatis file tersebut berupa kode ASCII. Sebuah terminal seperti keyboard, printer berhubungan dengan data karakter. Ketika sesorang menulis angka seperti 1234 yang kemudian muncul pada layar monitor, angka tersebut harus dikonversi menjadi empat karakter ("1", "2", "3", dan "4") .

Hal yang sama juga ketika seseorang membaca angka dari keyboard, sebuah data harus dikonversi dari data karakter menjadi data integer. Hal ini dikerjakan oleh operator >>. Karater ASCII "0" mempunyai nilai 48, "1" mempunyai nilai 49 dan seterusnya seperti terlihat pada lampiran kode ASCII. Ketika sesorang akan mengubah digit tunggal dari kode ASCII menjadi integer akan sama halnya anda mengubah menjadi data 48. Perhatikan program dibawah ini:

```
int integer;
char ch;

ch = '5';
integer = ch - 48;
 cout << "Integer " << integer << '\n';
```

Daripada mengingat karakter "0" yang lebih sulit dingat dengan 48, lebih baik mengkonversi menjadi karakter '0'. Perhatikan program dibawah ini:

```
integer = ch - '0';
```

komputer bekerja dengan data biner. Ketika komputer membaca dari file ASCII, program pada komputer memproses data karakter melalui routine konversi seperti routine konversi integer yang telah didefinisikan. Operasi ini akan menjadi lebih mahal dikarenakan membutuhkan ruang memori dan waktu. Sebuah file biner tidak membutuhkan konversi. File-file tersebut umumnya memakai ruang yang kecil dibandingkan file kode ASCII. Kelemahan pada file biner adalah tidak bisa dicetak secara langsung pada terminal atau printer. Jika file biner dicetak pada printer secara langsung tanpa adanya konversi yang terjadi adalah data cetak tidak sesuai seperti terlihat "#@$%@^Aa^AA^^JHC%^X". Tentunya apa yang akan terjadi jika dicoba mencetak file biner.

File ASCII marupakan file portbel karena banyak dipakai. File ini dapat dipindah dari satu mesin ke mesin yang lain dengan sedikit kesalahan. File biner bisa dikatakan pasti tidak portabel. File mana yang harus digunakan? Dalam beberapa kasus ASCII sangat baik.

Alasan ini dikemukakan jika mempunyai data dengan jumlah kecil atau medium waktu konversi tidak begitu mempengaruhi unjuk kerja program yang telah dibuat. Selain itu file ASCII juga membuat dalam melakukan verifikasi data. Hanya ketika menggunakan sejumlah data yang besar saja akan memakan ruang dan mempengaruhi unjuk kerja sehingga diperlukan data dalam format biner.

12.8. Binary I/O

Input atau output file biner dapat diselesaikan melalui dua fungsi keanggotaan yaitu: read and write. Penulisan atau sintak untuk membaca adalah:

```
in_file.read(data_ptr, size);
```

dimana data_ptr merupakan pointer sebagai tempat untuk meletakan data dan size merupakan angka dalam ukuran byte yang akan dibaca.

Fungsi keanggotaan qcount akan mengembalikan sejumlah byte data yang telah dibaca pada akhir. Hal ini dimungkinkan akan berkurang diandingkan dengan jumlah byte yang diminta. Sebagai contoh pembacaan yang mungkin ditemui pada akhir file atau ditemukan adanya kesalahan.

```
struct {
   int    width;
   int    height;
} rectangle;

in_file.read(static_cast<char *>(&rectangle), sizeof(rectangle));
if (in_file.bad( )) {
   cerr << "Tidak dimungkinkan membaca rectangle\n";
   exit (8);
}
if (in_file.gcount( ) != sizeof(rectangle)) {
   cerr << "Error: tidak bisa untuk membaca \n";
   cerr << "I/O mengalami error, EOF tidak bisa dihitung\n";
}
```

Dalam contoh program diatas ketika sedang membaca sebuah struktur persegi panjang. Sebuah operator & akan membuat persegi panjang tersebut manuju sebuah pointer. Kemudian sebuah sintak static_cast<char *> dibutuhkan setelah read menginginkan sebuah deretan array. Sebuah operator sizeof digunakan untuk menentukan berapa banyak byte yang dibaca sebaik pengecekan tersebut sampai operasi tersebut menjadi berhasil.
Sebuah keanggotaan fungsi write telah memanggil secara sekuensial sama dengan operasi read adalah dengan sintak:

```
out_file.write(data_ptr, size);
```

12.9. Buffer

Buffer atau sering kita artikan sebagai penahan atau penyangga suatu memori sebuah data. Penyangga sebuah I/O tidak bisa ditulis secara langsung pada sebuah file. Sebuah data akan dijaga keberadaanya dalam sebuah buffer sampai disana cukup untuk menulis

dalam kapasitas data besar atau sampai sebuah buffer disegarkan kembali. Sebuah program yang mengikuti dirancang untuk dicetak pesan cepat sampai setiap bagian terselesaikan. Perhatikan potongan program dibawah ini:

```
cout << "mulai";
do_step_1( );
 cout << "langkah 1 complete";
do_step_2( );
 cout << "langkah 2 complete";
do_step_3( );
 cout << "langkah 3 complete\n";
```

sebagai ganti dari menulis sebuah pesan sampai pada tiap langkah selesai, cout pesan tersebut diletakan dalam sebuah buffer. Hanya setelah program selesai maka selanjutnya buffer memperleh penyegaran atau pengosongan dan semua pesan yang datang ditumpahkan keluar dengan seketika. Sebuah manipulator I/O flush akan menyegarkan buffer-buffer tersebut. Sifat yang ditulis diatas diberikan contoh seperti pada program dibawah ini:

```
cout << "mulai" << flush;
do_step_1( );
 cout << "Langkah 1 complete" << flush;
do_step_2( );
 cout << "langkah 2 complete" << flush;
do_step_3( );
 cout << "langkah 3 complete\n" <<
flush;
```

karena tiap pernyataan keluaran berakhir dengan flush, sebuah keluaran akan ditampilkan secara langsung. Ini artinya bahwa penyelesaian pesan akan datang sesuai dengan waktunya.

Dalam I/O buffer, data ditahan dan kemudian dikirim dalam bentuk file, sedangkan dalam I/O tidak terbuffer data secara langsung dikirim dalam bentuk file. Analogi ini disampaikan dalam bentuk kejadian di lingkunang kita pada sejumlah penjepit kertas yang jatuh kelantai seseorang dapat mengambilnya dalam mode buffer atau tanpa buffer. Dengan mode buffer jika anda menggunakan tangan kanan untuk mengambil penjepit kertas dan kemudian memindahkan penjepit ke tangan kiri. Proses tersebut diulang-ulang sampai pada tangan kiri penuh kemudian penjepit tersebut dipindahkan ke dalam kotak yang ada diatas meja. Dengan buffer I/O akan ada media sementara yang digunakan untuk menampung sebelum dipindahkan ke media yang utama dan lebih besar. Dalam mode tanpa buffer, dapat dianalogikan seperti anda mengambil sejumlah penjepit kertas yang jatuh dilantai dan langsung memasukan kedalam kotak yang berada diatas meja. Pekerjaan ini dilakukan dengan tangan kanan tanpa menggunakan tangan kiri sebagai penampung sementara.

Dalam kebanyakan kasus, I/O dengan buffer sering digunakan daripada tanpa buffer. Dalam I/O tanpa buffer tiap pembacaan dan penulisan membutuhkan pemanggilan system. Beberapa pemanggilan pada sisem operasi sangat mahal dan membutuhkan waktu yang banyak, sehingga dengan I/O yang menggunakan

buffer dapat meminimalkan panggilan-panggilan yang dilakukan.

I/O tanpa buffer biasanya digunakan ketika hanya membaca atau menulis data biner dalam jumlah besar ketika kendali langsung dari piranti atau sebuah file dibutuhkan.

Kembali pada penjepit kertas seperti dicontohkan diatas, jika seseorang menggambil benda kecil seperti penjepit tersebut, tentunya oran tersebut tidak akan menemui kesulitan menggunakan tangan kiri sebagai penahan/penampung sementara. Tetapi jika seseorang menambil bola meriam yam mana bendanya lebih besar ddibandingkan penampungnya maka kasus seperti ini tidak menggunakan buffer.

Sebuah open system, pemanggilan digunakan untuk membuka file tanpa buffer. Sebuah definisi makro digunakan dengan cara memanggil dari system yang berbeda. Sebagai contoh bekerja pada dua system yang berbeda seperti pada system operasi UNIX dan MS-DOS/windows. Dalam kasus tersebut maka sebuah kondisi compilasi (#ifdef/#endif) digunakan untuk memperbaiki file tersebut seperti contoh program dibawah ini:

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

#ifdef _ _MSDOS_ _     // jika bekerja pada MS-DOS
#include <io.h>         // memanggil include file i/o pada MS-DOS
#else                   /* _ _MSDOS_ _ */
#include <unistd.h>     // memanggil include file i/o pada UNIX
#endif                  /* _ _MSDOS_ _ */
```

Sebuah perintah atau sintak untuk membuka akan memanggil:

```
int file_descriptor = open(name, flags);          // file yang sudah ada
file_descriptor = open(name, flags, mode);        // file baru
```

dimana, file_descriptor merupakan integer yang digunakan untuk mengidentifikasikan file untuk dibaca, ditulis atau ditutup pemanggilannya. Jika file_descriptor kurang dari 0 berarti kesalahan telah terjadi. Name merupakan nama file. Flag merupakan definisi dalam file header fcntl.h, dimana flag tersebut dijelaskan dalam table dibawah. Sedangkan mode adalah mode proteksi file dimana mode ini biasanya 0644.

Tabel 12.6. Open flag

FLAG	ARTI
O_RDONLY	Open for reading only.
O_WRONLY	Open for writing only.
O_RDWR	Open for reading and writing.
O_APPEND	Append new data at the end of the file.
O_CREAT	Create file (the file mode parameter required when this flag is present).
O_TRUNC	If the file exists, truncate it to 0 length.
O_EXCL	Fail if file exists.
O_BINARY	Open in binary mode (older Unix systems may not have this flag).

Sebagai contoh pada saat operasi membuka file dengan nama file data.txt dalam mode pembacaan text, maka potongan program yang dituliskan adalah sebagai berikut:

```
data_fd = open("data.txt", O_RDONLY);
```

selain contoh diatas, bagaimana untuk membuat sebuah file output.dat dapat ditulis, maka potongan program adalah sebagai berikut:

```
out_fd = open("output.dat",
O_CREAT|O_WRONLY, 0666);
```

Pada sintak tersebut diatas menggunakan operator OR (|) yang artinya merupakan pengabungan flag yang melebihi dari satu, untuk mempermudah dan mempercepat penulisan ketika beberapa program telah berjalan pada awal, dan tiga file sudah terbuka, dimana hal tersebut dapat dideskrisikan pada table dibawah ini:

Tabel 12.7. Standar file tidak terbuffer

NOMOR FILE	PENJELASAN
0	Standard in
1	Standard out
2	Standard error

Format dari pemanggilan read adalah:

```
read_size = read(file_descriptor, buffer, size);
```

dimana: read_size, merupakan jumlah actual dari byte yang dibaca, 0 mengindikasikan akhir file dan bilangan negative mengindikasikan adanya kesalahan dalam file.

File_descriptor, merupakan sebuah file yang mendeskripsikan dari file yang dibuka, buffer merupakan sebuah pointer untuk menempatkan dan meletakan data yang dibaca dari file. Size merupakan ukuran data yang dibaca. Hal ini merupakan ukuran dari file yang diminta. Pada jumlah sebenarnya dari byte yang dibaca mungkin lebih

kecil dibandingkan dengannya. Format pemanggilan penulisan adalah sebagai berikut:

```
write_size = write(file_descriptor, buffer, size);
```

pada sintaks diatas, write_size merupakan jumlah byte data actual yang ditulis, bilangan negative menunjukan adanya kesalahan. File descriptor adalah file yang meneskripsikan saat sebuah file dibuka.

Buffer adalah sebuah pointer dimanadata ditulis. Size adalah ukuran data yang ditulis. Sebuah system akan dicoba untuk ditulisnya dalam jumlah besar, tetapi jika piranti telah penuh ata disana ada beberapa masalah, sejumlah bilangan dari byte mungkin akan ditulis. Terakhir adalah melakukan penutupan file dimana sintaknya adalah sebagai berikut:

```
flag = close(file_descriptor)
```

pada sintaks tersebut diatas terdapat flag yang digunakan untuk menandai bahwa ketika data 0 untuk menandai bahwa penutupan berhasil sedangkan, bilangan negative ketika mengalami kegagalan atau error. File descriptor untuk mendeskripsikan saat file dibuka.

Program dibawah ini merupakan pencopyan file. Operasi ini mengunakan I/O tanpa buffer karena program tersebut membutuhkan buffer yang besar. Program dibuat tidak menggunakan I/O buffer untuk membaca 1 KB data dan kemudian dipindahkan menuju buffer dengan menggunakan ifstream dikirim menuju buffer 16KB.

Program 12.4.

```
#include<conio.h>
#include <iostream>
#include <cstdlib>

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

using namespace std;


#ifdef _ _WIN32_ _     // jika kita menggunakan Windows32
#include <io.h>        // include file i/o bekerja pada Windows32
#else                  /*_ _WIN32_ _ */
#include <unistd.h>    // include file i/o bekerja pada unix
#endif                 /*_ _WIN32_ _ */
```

```
const int BUFFER_SIZE = (16 * 1024); // menggunakan buffer 16k

int main(int argc, char *argv[])
{
   char  buffer[BUFFER_SIZE];  // buffer data
   int   in_file;              // input file descriptor
   int   out_file;             // output file descriptor
   int   read_size;            // number of bytes on last read

   if (argc != 3) {
       cerr << "Error: Angka sebagai arugmen salah \n";
       cerr << "penggunaanya adalah : copy <from> <to>\n";
      exit(8);
   }
   in_file = open(argv[1], O_RDONLY);
   if (in_file < 0) {
       cerr << "Error:tidak bisa dibuka " << argv[1] << '\n';
      exit(8);
   }
   out_file = open(argv[2], O_WRONLY | O_TRUNC | O_CREAT, 0666);
   if (out_file < 0) {
       cerr << "Error:Tidak bisa dibuka " << argv[2] << '\n';
      exit(8);
   }
   while (true) {
      read_size = read(in_file, buffer, sizeof(buffer));

      if (read_size == 0)
         break;          // akhir file

      if (read_size < 0) {
          cerr << "Error:kesalahan baca\n";
         exit(8);
      }
      write(out_file, buffer, (unsigned int) read_size);
   }
   close(in_file);
   close(out_file);
   getch();
   return (0);
}
```

Langkah pertama harus membuat semua ukuran buffer didefinisikan sebagai nilai konstan, sehingga hal tersebut sangat mudah dimodifikasi, biasanya cukup mudah diingat yaitu misalnya jika ulkurannya 16K maka programmer dapat menggunakan dengan menuliskan 16 * 1024. Atau bentuk konstannya dapat ditulis seperti sebelumnya yaitu 16K.

Jika pengguna menggunakan program yang tidak sesuai, akan muncul pesan bahsa hasil atau program mengalami kesalahan. Untuk menolongnya atau membetulkannya maka sebuah pesan akan muncul bagaimana menggunakan program yang betul. Suatu ketika programmer dimungkinkan tidak mengetahui bahwa buffer telah penuh, sehingga perlu ditambahkan read_size yang digunakan untuk menentukan jumlah byte yang akan ditulis

12.10. Rutin-Rutin pada C++

C++ mengizinkan anda untuk menggunakan library I/O pada C didalam program C++. Hal ini terjadi banyak kali terjadi karena seseorang membutuhkan suatu program C, menterjemahkannya kepada C++, dan tidak ingin mengganggu pemanggilan I/O. Dalam beberapa hal, library bahasa C yang lama bisa lebih baik dan lebih mudah digunakan dibandingkan dengan yang baru dalam hal ini C++.

Sebagai contoh, rutin-rutin string pada C seperti `sscanf` dan `sprintf` menggunakan suatu jauh lebih kompak dalam sistim pengaturannya dibanding C++. Deklarasi-deklarasi pada struktur dan fungsi yang digunakan oleh fungsi C I/O disimpan di dalam standar file include `<cstdio>`.

```
FILE *file_variable;     /* Comment */
```

sebagai contoh:

```
#include <cstdio>
```

```
FILE *in_file;          /* File berisi data masukan data */
```

Sebelum suatu file dapat digunakan, yang harus dibuka dengan fungsi adalah fopen. fopen kembali pada sebuah pointer ke file struktur. Format untuk fopen adalah:

```
file_variable = fopen(name, mode);
```

dimana file_variablenya adalah: name adalah sebuah nama yang digunakan oleh sebuah file seperti ("data.txt", "temp.dat", dan lain-lain). Mode merupakan sebagai indikasi apakah file akan dibaca atau ditulis. Mode adalah dengan tanda “w” adalah untuk menulis atau “r” untuk membaca. Fungsi fclose untuk menutup file, dimana format fclose adalah:

```
status = fclose(file_variable);
```

variabel status akan menjadi no jika fclose telah sukses atau bukan nol (nonzero) jika mengalami kesalahan.

C menyediakan tiga three langkah sebelum membuka sebuah

file. Hal ini dapat dilihat pada daftar dibawah ini:

Table 12.8. File Standard

FILE	DESCRIPTION
stdin	Standar masukan (membuka untuk dibaca). Pada bahasa C++ ekivalen dengan **cin**.
stdout	Standard keluaran (membuka untuk ditulis). Pada bahasa C++ ekivalen dengan **cout**.
stderr	Standard salah (membuka untuk ditulis). Pada bahasa C++ ekivalalen dengan **cerr**.
	(dalam bahasa C tidak ada sedangkan dalam bahasa C++ ekivalen dengan **clog**.

Fungsi fgetc akan membaca sebuah karakter dari suatu file. Jika tidak ada lebih banyak data di dalam file, fungsi kembali secara konstan ke EOF (EOF dideskkripsikan pada cstdio). Yang perlu dicatat bahwa fgetc kembalikan satu bilangan bulat, bukan suatu karakter. Ini diperlukan karena EOF flag harus mempunyai nilai bukan karakter.

Program 12.5.

```
#include <cstdio>
#include <cstdlib>                  /* File standar ANSI C */
#include <iostream>

using namespace std;

const char FILE_NAME[] = "input.txt";  // nama file masukan

int main( )
{
   int  count = 0;                  // jumlah karakter
   FILE *in_file;                   // masukan file

   int ch;                           // karakter atau EOF flag dari masukan

   in_file =  fopen(FILE_NAME, "rb");
   if (in_file == NULL) {
       cerr << "tidak bisa membuka File " << FILE_NAME << '\n';
      exit(8);
   }

   while (true) {
```

```
    ch =  fgetc(in_file);
    if (ch == EOF)
       break;
    ++count;
  }
   cout << "jumlah karakter" << FILE_NAME << " adalah " << count << '\n';

   fclose(in_file);
  return (0);
}
```

12.11. File sekuensial

Sequential file (Arsip sekuensial) adalah sekumpulan rekaman yang disimpan dalam media penyimpanan sekunder komputer, yang dapat diakses secara sekuensial mulai dari rekaman pertama sampai dengan rekaman yang terakhir, rekaman per rekaman secara searah. File sekuensial berisi rekord-rekord data yang tidak mengenal posisi baris atau nomor rekord pada saat aksesnya, dan setiap record dapat mempunyai lebar yang berbeda-beda.

Akses terhadapnya selalu dimulai dari awal file dan berjalan satu persatu menuju akhir dari file. Dengan demikian, penambahan file hanya dapat dilakukan terhadap akhir file, dan akses terhadap baris tertentu harus dimulai dari awal file.

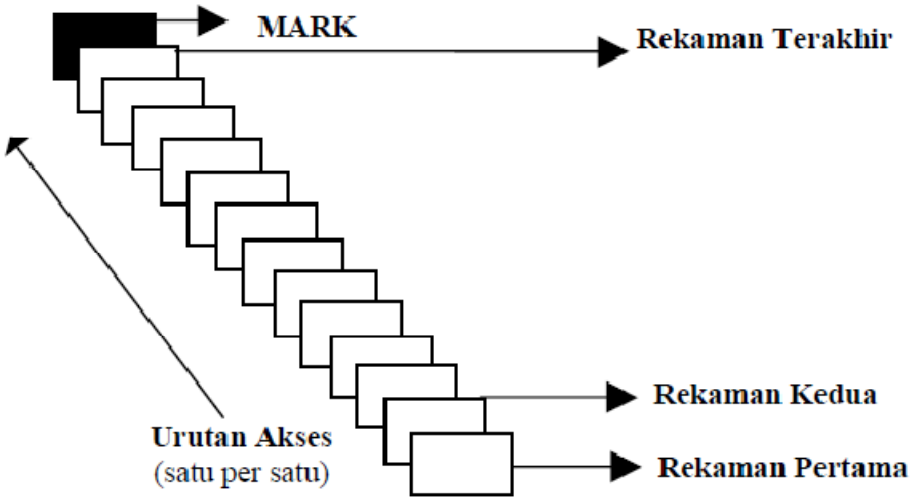
Fungsi baku yang terkait dengan file sekuensial ini antara lain adalah fprintf, fscanf, dan rewind . Program berikut menyajikan penanganan file sekuensial tentang data nasabah yang berisi tiga field, yaitu nomor identitas (*account*), nama (*name*), dan posisi tabungannya (*balance*) untuk (1) menyajikan yang tabungannya bernilai nol, (2) berstatus kredit, dan (3) berstatus debet. Misalny sebuah file data tersimpan dengan nama klien.dat.

Rekaman terakhir adalah rekaman fiktif, yang menandai akhir dari arsip. Pada beberapa implementasi, rekaman fiktif ini disebut sebagai EOF (End Of File). Arsip sekuensial berasal dari hasil perekaman (penulisan) yang juga dilakukan rekaman per rekaman. Setiap rekaman boleh berisi type dasar ataupun type terstruktur yang telah didefinisikan, setiap rekaman strukturnya sama.

Elemen dari rekaman disebut sebagai "field", ada field (atau juga sekumpulan field) rekaman yang disebut sebagai "key" karena kekhususannya dalam proses. Jika key dari setiap rekaman tidak ada yang sama (unik), maka key menjadi identitas rekaman, dan disebut "primary key". Setiap rekaman dapat diakses dan dikonsultasi (dibaca) menurut urutannya dengan pemanggilan primitif akses yang tersedia. Perekaman dapat dilakukan melalui primitif penulisan.

Perhatikan bahwa suatu arsip sekuensial hanya dapat disiapkan hanya pada salah satu modus operasi: diakses/dibaca, **atau** ditulis.

Dalam definisi arsip sekuensial, tidak pernah sebuah arsip diproses untuk kedua modus: dibaca dan sekaligus ditulis. Primitif-primitif tsb, akan didefinisikan kemudian.



Gambar 12.3. Operasi File skuensial

Program 12.6

```
#include <stdio.h>

using namespace std;

void main() {
int request, account;
float balance;

char name[25];
  FILE *cfPtr;
   if ( (cfPtr = fopen("klien.dat", "r+") ) == NULL )
      printf("File could not be opened\n");
   else {
       printf ( "Enter request\n"
       "1 - List accounts with zero balances\n"
       "2 - List accounts with credit balances\n"
       "3 - List accounts with debit balances\n"
       "4 - End of run\n? " ) ;
       scanf( "%d", &request );
```

```
   while (request != 4) {
   fscanf (cfPtr, "%d%s%f", &account, name, &balance);

   switch (request) {
   case 1:
      printf ("\nAccounts with zero balances:\n");
      while ( !feof(cfPtr) ) {
      if (balance == 0)
        printf ("%-10d%-13s7.2f\n", account, name, balance);
        fscanf (cfPtr, "%d%s%f", &account, name, &balance);
      }
   break;

   case 2:
      printf ("\nAccounts with credit balances:\n");
      while ( !feof(cfPtr) ) {
      if (balance < 0)
        printf ("%-10d%-13s7.2f\n", account, name, balance);
        fscanf (cfPtr, "%d%s%f", &account, name, &balance);
      }
   break;

   case 3:
      printf ("\nAccounts with debit balances:\n");
      while ( !feof(cfPtr) ) {
      if (balance > 0)
        printf ("%-10d%-13s7.2f\n", account, name, balance);
        fscanf (cfPtr, "%d%s%f", &account, name, &balance);
      }
   break;
   }

   rewind(cfPtr);
   printf( "\n? ");
   scanf ("%d", &request);
   }
 printf ("End of run.\n");
 fclose(cfPtr);
 }
}
```

12.12. Program Operasi File

Dari uraian operasi file diatas, dibawah ini merupakan beberapa operasi file yang sering dilakukan sehari-hari. Setiap kita bekerja dengan menggunakan komputer kita biasanya melakukan operasi membuka file, menulis file, menambah data file, menghapus file, menutip file dan lain sebagainya. Operasi tersebut dilakukan dengan menggunakan mouse atau keyboard secara langsung

Bagaimana jika hal tersebut dilakukan leh program?. Tentunya hal tersebut harus ada instruksi atau perintah-perintah yang digunakan oleh tiap operasi tersebut. Dibawah ini merupakan beberapa operasi file yang dilakukan dengan bahasa C++.

12.12. 1. Membuka file

Perintah membuka file dan menghubungkannya dengan stream dengan fopen (library stdio.h) dan menutupnya dengan fclose

```
FILE *fopen(char *nama_file, char *mode)
FILE *fclose(FILE *pointer_file)
```

Dalam membuka file diperlukan mode, dimana hal tersebut merupakan cara pengaksesan file. Berikut daftar mode yang dapat digunakan:

Tabel 12.9. Mode operasi file pada C

MODE	ARTI
r	Membuka sebuah file teks untuk pembacaan
w	Membuat sebuah file teks untuk penulisan
a	Menambahkan data ke sebuah file teks
rb	Membuka sebuah file binary untuk pembacaan
wb	Membuat sebuah file binary untuk penulisan
ab	Menambahkan data ke sebuah file binary
r+	Membuka sebuah file teks untuk pembacaan/penulisan
w+	Membuat sebuah file teks untuk pembacaan/penulisan
a+	Menambahkan data/membuat file teks untuk pembacaan/penulisan
r+b atau rb+	Membuka sebuah file binary untuk pembacaan/penulisan
w+b atau wb+	Membuat sebuah file binary untuk pembacaan/penulisan
a+b atau ab+	Menambahkan data ke file binary untuk pembacaan/penulisan

Dalam operasi file dilakukan ada beberapa hal yang perlu diperhatikan antara lain: jika operasi open berhasil, **fopen()** mengembalikan sebuah **file pointer** yang valid, sedangkan jika operasi gagal, maka **fopen()** mengembalikan sebuah **null pointer**, sehingga harus selalu dicek

pada saat pembukaan file. Perhatikan program dibawah ini:

```
FILE *fp;
    if((fp=fopen("fileku.txt","r")==NULL) {
        cout << "Error dalam pembukaan file\n";
        exit(1);
        }
    fclose(fp);          //menutup stream file
```

Dibawah ini merupakan fungsi untuk operasi file teks antara lain

a. fgetc() dan fputc(), dimana Sintaknya dapat ditulis:

```
int fgetc(FILE *fp);
int fputc(int ch, FILE *fp);
```

Program 12.7

```
include <stdio.h>
#include <stdlib.h>

using namespace std;

int main()
{
   FILE *fp;
          int i;
          int ch;
   fp = fopen("foo.abc", "w");     //buka file foo.abc untuk ditulisi
          for (i=0;i<10;i++) {      //loop untuk meletakkan karakter2
                    fputc('A',fp);     //menuliskan karakter A
                    fputc('\n',fp);    //menuliskan pergantian baris
          }
          fclose(fp);

   if((fp = fopen("foo.abc", "r"))==NULL) {
                    cout<<"Error reading file...\n";
                    exit(1);
          }
          while (ch!=EOF) {         //baca file sampai tanda EOF (End of File)

      ch=fgetc(fp);                 //ambil satu karakter
                     putchar(ch);   //menampilkan karakter ke layar
          }

   fclose(fp);
```

```
}
```

b. fgets() dan fputs(), Sintaknya dapat diltulis sebagai berikut:

```
int fputs(char *str, FILE *fp);
char *fgets(char *str, int num, FILE *fp);
```

Program 12.8.

```
#include <stdio.h>
#include <stdlib.h>

void main()
{
   FILE *fp;
  char ch[14];
   fp = fopen("foo.abc", "w");              //buka file foo.abc untuk ditulisi
  fputs("String Contoh",fp);                //menuliskan string
  fclose(fp);

   if((fp = fopen("foo.abc", "r"))==NULL) {
                printf("Error reading file...\n");
                exit(1);
          }
          puts(fgets(ch,sizeof(ch),fp));      //cetak string di foo ke layar

   fclose(fp);
}
```

c. fscanf() dan fprintf()
dimana operasi ini mempunyai ciri-ciri sebagai berikut:

- Mirip dengan sintaks scanf() dan printf()
- Dapat digunakan untuk sembarang file (tidak hanya monitor/layar)
- Dapat menggunakan format data

Sintaks adalah sebagai berikut:

```
int fprintf(FILE *fp, const char *format, …);
int fscanf(FILE *fp, const char *format, …);
```

Program 12.9

```
#include <stdio.h>
#include <stdlib.h>

void main()
{
```

```
  FILE *fp;
  int i;
        char x[100];
  fp = fopen("foo.abc", "w");  //buka file foo.abc untuk ditulisi
  fprintf(fp, "\nSample Code\n\n");  //menuliskan sesuatu
  for (i = 1; i <= 10 ; i++) {
                fprintf(fp, "i = %d\n", i);
        }
  fclose(fp);

        if((fp=fopen("foo.abc","r"))==NULL) {
                printf("Error membuka file\n");
                exit(1);
        }
        while(!feof(fp)) {
                fscanf(fp,"%s",&x);
                puts(x);
        }
        fclose(fp);
}
```

d. fread() dan fwrite()

fungsi operasi ini antara lain:

- Untuk membaca dan menulis blok data (misalnya: Karakter, integer, structure,dan lain-lain)
- Untuk dapat menggunakan fwrite(), file harus dibuka dengan tambahan opsi “b” (binary)

Sintaks:

```
fwrite(void *buffer, int b_byte, int c,
file *fp);
fread(void *buffer, int b_byte, int c,
file *fp);
```

Keterangan:
buffer: pointer ke area di memori yang menampung data yg akan dibaca ke file
b_byte: banyaknya byte data yang akan dibaca/tulis (dapat menggunakan sizeof(buffer))
c: banyaknya blok data yang akan dibaca/ditulis
fp: pointer ke file

12.12.2. Menulis File

Salah satu jenis pemrosesan pada file adalah menulis atau merekam data ke file. Untuk menulis sebuah karakter, bentuk yang digunakan adalah dengan menggunakan Sintak sebagai berikut:

```
putc(int ch, file *fp)
```

dimana

- fp adalah pointer file yang dihasilkan oleh fopen()
- ch adalah karakter yang akan ditulis.

Program 12.10

```
#include "stdio.h"
#include "conio.h"
#define CTRL_Z 26

void main(){
        file *pf;          /* pointer ke file */
        char kar;
            if((pf = fopen("COBA.TXT", "w")) == NULL) /* ciptakan file */
        { cputs("File tak dapat diciptakan !\r\n");
        exit(1);           /* selesai */
}
while((kar=getche()) != CTRL_Z)
putc(kar, pf);             /* tulis ke file */
fclose(pf);                /* tutup file */
}
```

12.12.3. Menutup File

Setelah pemrosesan file selesai, file dapat ditutup menggunakan perintah

```
nama_obyek.close();
```

Program 12.11. Program untuk menulis teks ke dalam file

```
#include<iostream.h>
#include<fstream.h>

void main(){
      ofstream fileteks;
      fileteks.open("C:/Catat.txt");
      fileteks << "Untuk mencapai tujuan yg besar, maka tujuan itu"<< endl;
      fileteks << "harus dibagi-bagi menjadi tujuan kecil"<< endl;
      fileteks << "sampai tujuan itu merupakan tujuan yg dapat "<< "dicapai" << endl;
      fileteks << "berdasarkan kondisi dan potensi yg dimiliki saat "<< "itu " << endl;
      fileteks.close();
}
```

perintah fileteks.Open(“C:/catat.txt”); akan membuka file catatan.txt yang ada di C:\ . Apabila file tersebut belum ada maka akan dibuat secara otomatis, dan apabila sudah ada isi file catatan.txt akan terhapus.

12.12.4. Menambah Data File

Suatu file yang sudah ada sebelumnya dapat ditambah data yang baru (tidak menghapus data

lama). Caranya adalah dengan menambahkan perintah **ios::app** pada open().

```
nama_obyek.open(“nama file”, ios::app)
```

Program 12.12

```
#include<iostream.h>
#include<fstream.h>

void main(){
      ofstream fileteks;
      fileteks.open("C:/catatan.txt", ios::app);
      fileteks << endl;
      fileteks << "Oleh: Al Khowarizmi << endl;
      fileteks.close();
}
```

12.12.5. Memeriksa File

Dalam penulisan tidak selamanya jalan yang mulus ditemui. Ada kemungkinan terjadi saat file dibuka, ternyata file tidak ada. Dalam C++ tersedia fungsi untuk memeriksa kondisi-kondisi pada operasi file, sehingga kesalahan saat eksekusi dapat dikendalikan. Fungsi yang dimaksud adalah **fail()**.

Program 12.13.

```
#include<iostream.h>
#include<fstream.h>

void main(){
    ifstream fileteks;
          {
          ifstream digunakan u/ membaca file
          }
    fileteks.open("C:/catatan.txt");
    if (fileteks.fail()) cout << "Maaf file takdapat dibuka/"<< "tidak ditemukan";
    fileteks.close();
}
```

Operasi file dapat dilakukan dalam bentuk karakter. Misalnya proses penyimpanan data ke file dilakukan setiap karakter, atau membaca data file karakter per karakter. Operasi ini didukung oleh function **put()** dan **get()**.

Program 12.14. Menyimpan data karakter per karakter ke dalam file.

```
#include<iostream.h>
#include<fstream.h>

void main()
{
      ofstream fileteks;
      fileteks.open("C:/contoh.txt");
      fileteks.put('A');
      fileteks.put('B');
      fileteks.put('C');
      fileteks.close();
}
```

Program 12.15. Program untuk membaca file karakter per karakter

```
#include<iostream.h>
#include<fstream.h>

void main()
{
    char karakter;
    ifstream fileteks; {}
    fileteks.open("C:/contoh.txt");
    while(!fileteks.eof())
    {
          fileteks.get(karakter);
          cout << karakter;
    }
    fileteks.close();
}
```

12.17.Soal Latihan

Jawablah soal latihan dibawah ini dengan baik dan benar.

1. Apa yang dimaksud dengan file
2. Apa yang dimaksud dengan kode ascii
3. Apakah fungsi buffer pada operasi string
4. Buatlah program untuk menulis file dengan nama latih.txt
5. Buatlah program untuk menambah data pada file latih.txt
6. Buatlah program untuk menghapus file latih.txt
7. Apa yang dimaksud dengan file skuensial

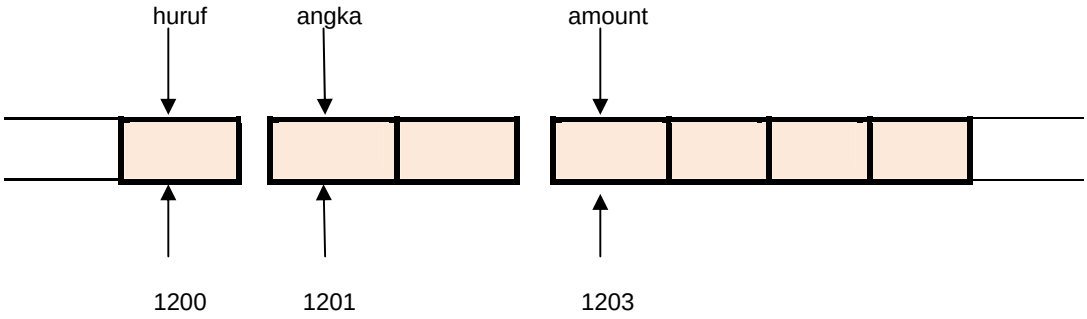
BAB 13
POINTER

13.1. Pemrograman pointer

Pada programmer pemula hal yang sulit dipahami adalah pointer. Pointer adalah pada dasarnya sama dengan veriabel lain. Dengan pointer setiap variable diletakan pada alamat memori yang besarnya cukup untuk menampung nilai dari sebuah variable tipe data. Pada sebuah komputer PC misalnya satu byte data secara umum dialokasikan untuk tipe data char, 2 byte untuk tipe data short, 4 byte untuk tipa data int, long dan float serta 8 byte untuk tipe data double. Tiap-tiap byte data memori mempunyai alamat yang unik. Sebuah variable alamat merupakan alamat dimana sebuah byte data pertama dapat dialokasikan. Sebagai contoh misalnya dapat didefinisikan dalam program sebagai berikut;

```
char huruf;
short angka;
float amount;
```

sehingga pengaturan memorinya dapat diilustrasikan sebagai berikut



Gambar 13.1. Ilustrasi Pengaturan Data Pada Sebuah Memori

Pada gambar diatas variable letter diletakan pada alamat 1200, number pada alamat 1201 dan amount pada alamat 1203.

Ketika programmer mendeklarasikan sebuah variable, sama halnya dengan menginformasikan compiler dua sesuatu yaitu : nama variable dan tipe variable. Misalnya kita akan mendeklarasikan tipe variable integer dengan nama variable k sehingga dapat ditulis:

```
Int k
```

Pada tipe variable integer merupakan tipe data yang digunakan compiler dimana memori yang terpakai adalah dua byte.

Setiap kali komputer menyimpan data, maka sistem operasi akan mengorganisasikan lokasi pada memori pada alamat yang unik. Misal untuk alamat memori 1776, hanya sebuah lokasi yang memiliki alamat tersebut. Dan alamat 1776 pasti terletak antara 1775 dan 1777. Dalam pointer, terdapat 2 jenis operator yang biasa digunakan.

Kegunaan pointer yang utama adalah untuk menyimpan alamat memori dari sebuah variabel (*data type* atau *object* dari *class*). Selain menyimpan alamat dari variabel, pointer juga dapat digunakan untuk menyimpan alamat dari sebuah fungsi (*function pointer*).

Function pointer telah digunakan sejak dikenalkannya bahasa C, dan banyak digunakan untuk sebuah fungsi callback atau untuk meningkatkan readability dari sebuah code

Anda dapat memperlakukan function pointer seperti pointer biasa (pointer ke datatype/object), anda dapat menyimpan,mengirimkan, merubah address, atau meng-evaluasi address dari pointer ke fungsi ini dengan sifat tambahan anda dapat memanggil fungsi yang ditunjuk oleh function pointer.

Setiap variabel yang dideklarasikan, disimpan dalam sebuah lokasi memori dan pengguna biasanya tidak mengetahui di alamat mana data tersebut disimpan. Dalam C++, untuk mengetahui alamat tempat penyimpanan data, dapat

digunakan tanda ampersand (&) yang dapat diartikan “alamat”.

Sebenarnya jika programer akan mendeklarasian sebuah variable, seorang programer tidak diharuskan menentukan lokasi sesungguhnya pada memory, karena hal ini akan dilakukan secara otomatis oleh kompiler dan operating sysem pada saat run-time.

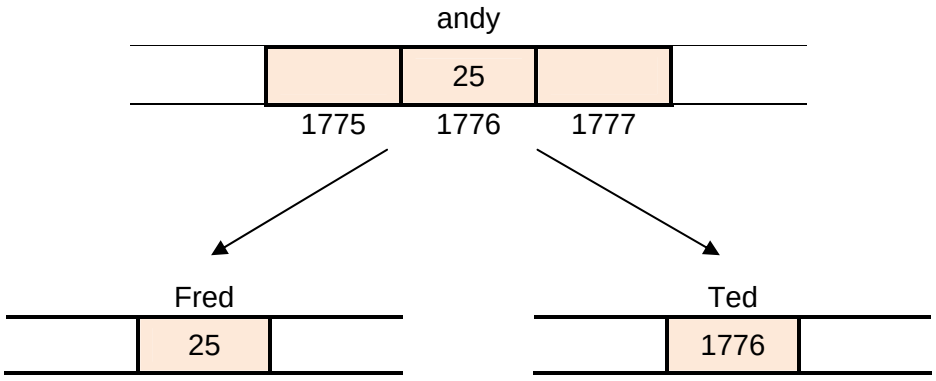
Jika ingin mengetahui dimana suatu variable akan disimpan, dapat dilakukan dengan memberikan tanda *ampersand* (&) didepan variable , yang berarti ***"address of"***. Perhatikan contoh dibawah ini:

```
ted = &andy;
```

Akan memberikan variable **ted** alamat dari variable **andy**, karena variable **andy** diberi awalan karakter *ampersand* (&), maka yang menjadi pokok disini adalah alamat dalam memory, bukan isi variable. Misalkan **andy** diletakkan pada alamat **1776** kemudian dituliskan instruksi sebagai berikut :

```
andy = 25;
fred = andy;
ted = &andy;
```

Maka hasilnya adalah sebagai berikut:

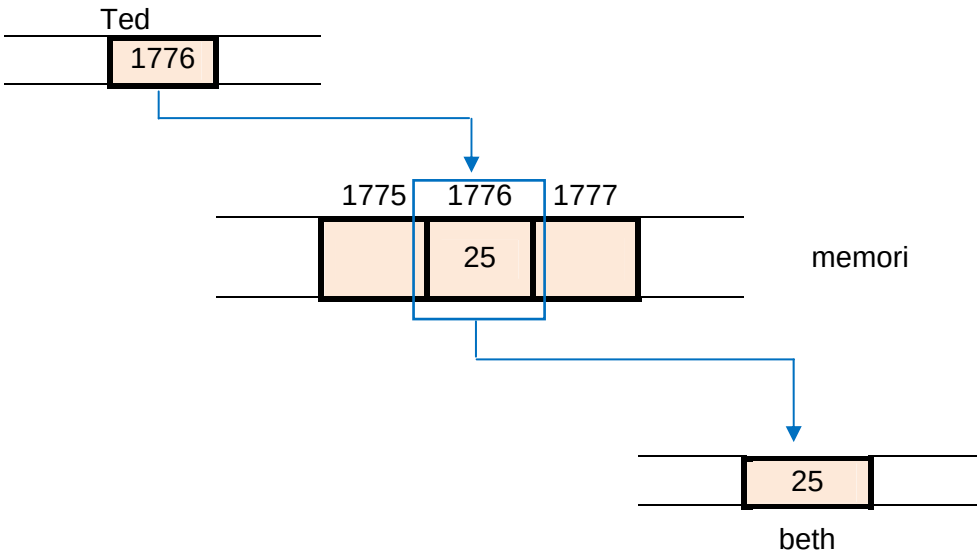


Gambar 13.2. Operator Alamat (&)

Dengan menggunakan pointer, kita dapat mengakses nilai yang tersimpan secara langsung dengan memberikan awalan operator *asterisk* (*) pada identifier pointer, yang berarti *"value pointed by"*. Contoh :

```
beth = *ted;
```

(dapat dikatakan:"beth sama dengan nilai yang ditunjuk oleh_ted") beth = 25, karena ted dialamat 1776, dan nilai yang berada pada alamat 1776 adalah 25.



Gambar 13.3. Operator Reference

Ekspresi dibawah ini semuanya benar, perhatikan pernyataan program dibawah:

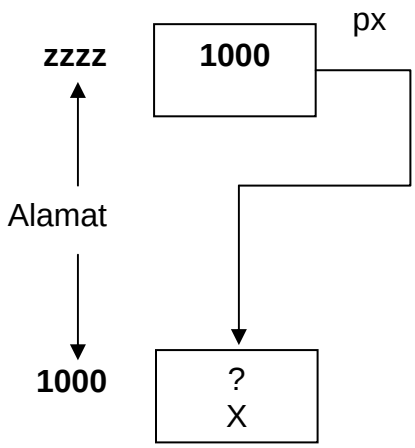
```
andy == 25
&andy == 1776
ted == 1776
*ted == 25
```

Ekspresi pertama merupakan *assignation* bahwa **andy=25;**. Kedua, menggunakan operator alamat (address/derefence operator (&)), sehingga akan mengembalikan alamat dari variabel **andy**. Ketiga bernilai benar karena *assignation* untuk **ted** adalah **ted = &andy;**. Keempat menggunakan reference operator (*) yang berarti nilai yang ada pada alamat yang ditunjuk oleh **ted**, yaitu **25**. Maka ekspresi dibawah ini pun akan bernilai benar :

```
*ted == andy
```

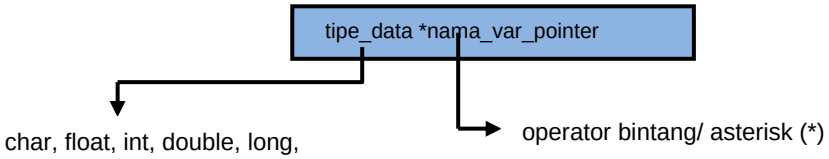
13.2. Deklarasi variabel bertipe pointer

Variabel pointer sering dikatakan sebagai variabel yang menunjuk ke obyek lain. Pada kenyataan yang sebenarnya, variabel pointer berisi alamat dari suatu obyek lain (yaitu obyek yang dikatakan ditunjuk oleh pointer). Sebagai contoh, **px** adalah variable pointer dan **x** adalah variabel yang ditunjuk oleh **px**. Kalau **x** berada pada alamat memori (alamat awal) 1000, maka **px** akan berisi 1000. Sebagaimana diilustrasikan pada gambar di bawah ini:



Gambar 13.4. Variabel pointer px menunjuk ke variabel x

Suatu variabel pointer dideklarasikan dengan bentuk sebagai berikut :

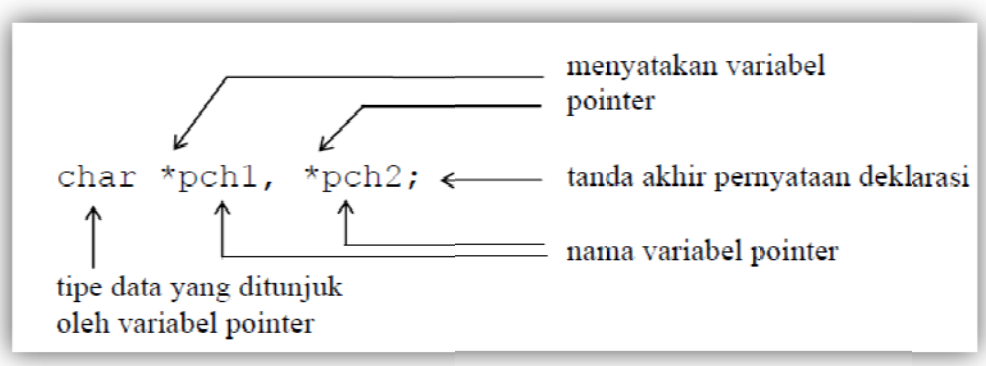


Gambar 13.7. deklarasi variabel pointer

Dimana ***type*** merupakan tipe dari data yang ditunjuk, bukan tipe dari pointer-nya. Dengan **tipe** dapat berupa sembarang tipe yang sudah dibahas pada bab-bab sebelumnya, maupun bab-bab berikutnya. Adapun nama_var_pointer adalah nama dari variabel pointer. Perhatikan contoh berikut ini:

```
int *px;
char *pch1, *pch2;
float * greatnumber;
```

Contoh pertama menyatakan bahwa **px** adalah variabel pointer yang menunjuk ke suatu data bertipe *int*, sedangkan contoh kedua masing **pch1** dan **pch2** adalah variabel pointer yang menunjuk ke data bertipe *char*.



Gambar 13.6. Ilustrasi pendeklarasian variabel pointer

Program 13.1

```
#include <iostream>
#include<conio.h>

using namespace std;

int main ()
{
        int nilai1 = 5, nilai2 = 15;
        int * mypointer;
        mypointer = &nilai1;
        mypointer = 10;
        mypointer = &nilai2;
        *mypointer = 20;
        cout << "nilai1==" << nilai1 << "/ nilai2==" << nilai2;
        return 0;
}
```

Keluaran program diatas adalah sebagai berikut:

```
Output : nilai1==10 / nilai2==20
```

Perhatikan bagaimana nilai dari nilai1 dan nilai2 diubah secara tidak langsung. Pertama mypointer diberikan alamat nilai1 dengan menggunakan tanda ampersand (&). Kemudian memberikan nilai 10 ke nilai yang ditunjuk oleh mypointer, yaitu alamat dari nilai1, maka secara tidak langsung nilai1 telah dimodifikasi. Begitu pula untuk nilai2.

Program 13.2

```
#include <iostream.h>

using namespace std;
```

```
int main ()
{
        int nilai1 = 5, nilai2 = 15;
        int *p1, *p2;
        p1 = &nilai1;                   // p1 = address of nilai1
        p2 = &nilai2;                   // p2 = address of nilai2
        *p1 = 10;                       // nilai pointed by p1 = 10
        *p2 = *p1;                      // nilai pointed by p2 = nilai pointed by p1
        p1 = p2;                        // p1 = p2 (nilai of pointer copied)
        *p1 = 20;                       // nilai pointed by p1 = 20
        cout << "nilai1==" << nilai1 << "/ nilai2==" << nilai2;
        return 0;
}
```

Keluaran program diatas adalah sebagai berikut:

```
nilai1==10 / nilai2==20
```

13.3. Inisialisasi Pointer

Dalam melakukan pemrogaman dengan menggunakan pointer yang pertama perlu dilakukan dalam membuat program adalah dengan melakukan inisialisasi pointer tersebut. Untuk lebih jelasnya perhatikan contoh dibawah ini:

```
int number; int *tommy = &number;
```

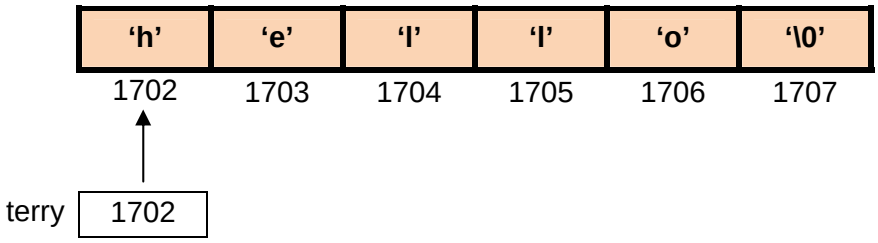
pernyataan diatas akan sama atau ekivalen dengan pernyataan dibawah ini:

```
int number; int *tommy; tommy = &number;
```

Seperti pada array, inisialisasi isi dari pointer dapat dilakukan dengan deklarasi seperti contoh berikut :

```
char * terry = "hello";
```

Misalkan kata "hello" disimpan pada alamat 1702 dan seterusnya, maka deklarasi tadi dapat digambarkan sebagai berikut:

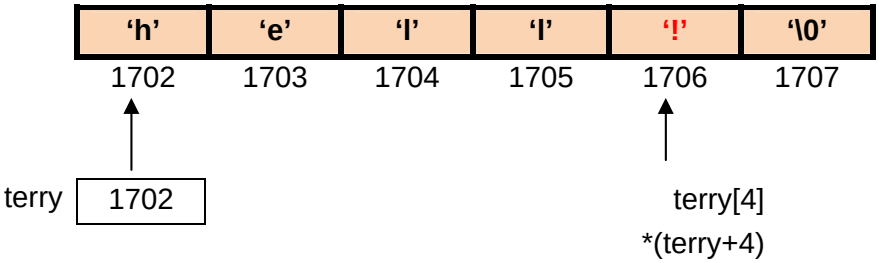


Gambar 13.7. deklarasi hello pada mulai alamat 1702

terry berisi nilai **1702** dan bukan **'h'** atau **"hello"**, walaupun **1702** menunjuk pada karakter tersebut. Sehingga jika akan dilakukan perubahan pada karakter **'o'** diganti dengan tanda **'!'** maka ekspresi yang digunakan ada 2 macam :

```
terry[4] = '!'; *(terry+4) = '!';
```

Penulisan **terry[4]** dan ***(terry+4)**, mempunyai arti yang sama. Jika digambarkan:



Gambar 13.8 Deklarasi "Hello" pada Alamat Terry[4] Diisi !

Perhatikan contoh program dibawah, dimana *char* memerlukan 1 byte, *short* memerlukan 2 bytes dan *long* memerlukan 4. Terdapat 3 buah pointer :

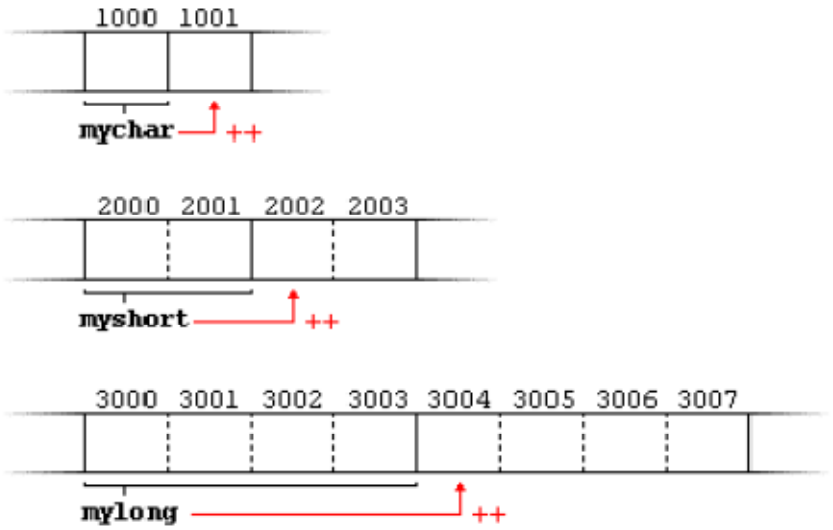
```
char *mychar;
short *myshort;
long *mylong;
```

ekspresi diatas akan menunjuk pada lokasi dimemory masing-masing **1000**, **2000** and **3000**, sehingga jika dituliskan :

```
mychar++; myshort++; mylong++;
```

mychar, akan bernilai 1001, myshort bernilai 2002, dan mylong bernilai 3004. Alasannya adalah ketika terjadi pertambahan maka akan

ditambahkan dengan tipe yang sama seperti yang didefinisikan berupa ukuran dalam bytes.



Gambar 13.9. Lokasi Memori Ukuran Byte

Perhatikan ekspresi dibawah ini :

```
*p++; *p++ = *q++;
```

Ekspresi pertama equivalen dengan *(p++) dan yang dilakukan adalah menambahkan **p** (yaitu alamat yang ditunjuk, bukan nilai yang dikandungnya). Ekspresi kedua, yang dilakukan pertama adalah memberikan nilai *q ke *p dan kemudian keduanya ditambahkan 1 atau dengan kata lain :

```
*p = *q; p++; q++;
```

Tipe pointer *void* merupakan tipe khusus. *void* pointers dapat menunjuk pada tipe data apapun, nilai integer value atau float, maupun string atau karakter. Keterbatasannya adalah tidak dapat menggunakan operator asterisk (*), karena panjang pointer tidak diketahui, sehingga diperlukan operator *type casting* atau *assignations* untuk mengembalikan nilai *void* pointer ketipe data sebenarnya.

Program 13.3

```
include <iostream.h>

using namespace std;

void increase (void* data, int type)
```

```
{
    switch (type)
    {
        case sizeof(char) : (*((char*)data))++; break;
        case sizeof(short): (*((short*)data))++; break;
        case sizeof(long) : (*((long*)data))++; break;
    }
}

int main ()
{
        char a = 5;
        short b = 9;
        long c = 12;
        increase (&a,sizeof(a));
        increase (&b,sizeof(b));
        increase (&c,sizeof(c));
        cout << (int) a << ", " << b << ", " << c;
        return 0;
}
```

Keluaran program diatas adalah sebagai berikut:

```
6, 10, 13
```

13.4. Pointer untuk fungsi

C++ memperbolehkan operasi dengan pointer pada function. Kegunaan yang utama adalah untuk memberikan satu function sebagai parameter untuk function lainnya. Deklarasi pointer untuk function sama seperti prototype function kecuali nama function dituliskan diantara tanda kurung () dan operator asterisk (*) diberikan sebelum nama.

Program 13.4. pointer to functions

```
#include <iostream.h>

using namespace std;

        int addition (int a, int b)
        { return (a+b); }
        int subtraction (int a, int b)
        { return (a-b); }
        int (*minus)(int,int) = subtraction;
```

```
        int operation (int x, int y, int (*functocall)(int,int))
        {
                int g;
                g = (*functocall)(x,y);
                return (g);
        }
        int main ()
        {
                int m,n;
                m = operation (7, 5, addition);
                n = operation (20, m, minus);
                cout <<n;
                return 0;
}
```

Keluaran program diatas adalah sebagai berikut:

```
8
```

Dari contoh diatas, **minus** merupakan pointer global untuk function yang mempunyai 2 parameters bertipe **int**, kemudian diberikan untuk menunjuk function **subtraction**, ditulis dalam satu baris instruksi :

```
int (* minus)(int,int) = subtraction;
```

Agar suatu pointer menunjuk ke variabel lain, mula-mula pointer harus diisi dengan alamat dari variabel yang akan ditunjuk. Untuk menyatakan alamat dari suatu variabel, operator **&** (operator alamat, bersifat *unary*) bisa dipergunakan, dengan menempatkannya di depan nama variabel. Sebagai contoh, bila **x** dideklarasikan sebagai variabel bertipe *int*, maka

```
&x
```

berarti "alamat dari variabel **x**". Adapun contoh pemberian alamat **x** ke suatu variable pointer px (yang dideklarasikan sebagai pointer yang menunjuk ke data bertipe *int*) yaitu:

```
px = &x;
```

Pernyataan di atas berarti bahwa **px** diberi nilai berupa alamat dari variabel **x**. Setelah pernyataan tersebut dieksekusi barulah dapat dikatakan bahwa **px** menunjuk ke variable **x**.

Jika suatu variabel sudah ditunjuk oleh pointer, variabel yang ditunjuk oleh pointer tersebut dapat diakses melalui variabel itu sendiri (pengaksesan langsung) ataupun melalui pointer (pengaksesan tak langsung). Pengaksesan tak langsung dilakukan dengan menggunakan operator *indirection* (tak langsung) berupa simbol ***** (bersifat *unary*). Contoh penerapan operator * yaitu:

```
*px
```

yang menyatakan "isi atau nilai variabel/data yang ditunjuk oleh pointer **px**" . Sebagai contoh jika **y** bertipe *int*, maka sesudah dua pernyataan berikut

```
px = &x;
y = *px;
```

y akan berisi nilai yang sama dengan nilai **x**. Kedua pernyataan di atas

```
px = &x;
y = *px;
```

sebenarnya dapat digantikan dengan sebuah pernyataan berupa

```
y = x;
```

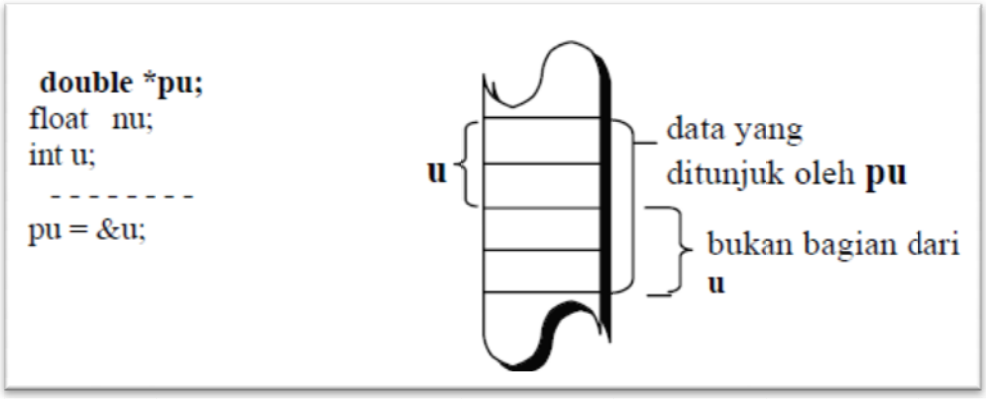
Seandainya pada program di atas tidak terdapat pernyataan

```
px = &x;
```

namun terdapat pernyataan

```
y = *px;
```

maka **y** tidaklah berisi nilai **x**, sebab **px** belum diatur agar menunjuk ke variabel **x.** Hal seperti ini harap diperhatikan. Kalau program melibatkan pointer, dan pointer belum diinisialisasi, ada kemungkinan akan terjadi masalah yang dinamakan "bug" yang bias mengakibatkan komputer tidak dapat dikendalikan *(hang)*. Selain itu tipe variabel pointer dan tipe data yang ditunjuk harus sejenis. Bila tidak sejenis maka akan terjadi hasil yang tidak diinginkan. Untuk lebih jelasnya lihat gambar berikut.



Gambar 13.10. Ilustrasi Kesalahan Yang Terjadi Karena Tipe Tidak Sejenis

Program 13.5. Mengakses isi suatu variabel melalui pointer.

```
#include <stdio.h>
#include <iostream.h>

using namespace std;
```

```
main()
{       int y, x = 87;
        int *px;
        px = &x;
        y = *px;
        cout <<"Alamat x    = \n"<< &x;
        cout <<"Isi px      = \n"<< px;
        cout <<"Isi x       = \n"<< x;
        cout <<"Nilai yang ditunjuk oleh px = \n"<< *px;
        cout <<"Nilai y     = \n"<< y;
}
```

Mengakses isi suatu variabel melalui pointer. Tipe variabel pointer dan tipe data yang ditunjuk harus sejenis

Program 13.6

```
#include <conio.h>
#include <iostream.h>

using namespace std;

main()
{
        int *pu;
        int nu;
        int u = 1234;
        pu = &u;
        nu = *pu;
        cout <<"Alamat dari u = \n"<< &u;
        cout <<"Isi pu = \n"<< pu;
        cout <<"Isi u = \n"<< u;
        cout <<"Nilai yang ditunjuk oleh pu = \n"<< *pu;
        cout <<"Nilai nu = \n"<< nu;
        getch();
        return 0;
}
```

Program 13.7

```
#include <conio.h>
#include <iostream.h>
```

```
using namespace std;

main()
{
        int z = 20, s = 30;
        int *pz, *ps;
        pz = &z;
        ps = &s;
        *pz += *ps;
        cout <<"z = \n"<< z;
        cout <<"s = \n"<< s;
        getch();
        return 0;
}
```

Program 13.8

```
#include <conio.h>
#include <iostream.h>

using namespace std;

main()
{
        char c = 'Q';
        char *char_pointer = &c;
        cout <<"\n"<< c << *char_pointer;
        c = '/';
        cout <<"\n"<< c << *char_pointer;
        *char_pointer = '(';
        cout <<"\n"<< c << *char_pointer;
        getch();
        return 0;
}
```

Program 13.9

```
#include <conio.h>
#include <iostream>

using namespace std;

int main()
{
```

```
        int x = 1, y = 2;
        int *ip;
        ip = &x;
        y = *ip;
        x = ip;
        *ip = 3;
        cout <<"x = "<< x;
        cout <<"y = "<< y;
        getch();
        return 0;
}
```

13.5. Mengakses dan Mengubah isi Pointer

Program berikut memberikan gambaran tentang pengubahan isi suatu variable secara tak langsung (yaitu melalui pointer). Mula-mula **pd** dideklarasikan sebagai pointer yang menunjuk ke suatu data bertipe *float* dan **d** sebagai variabel bertipe *float*. Selanjutnya

```
d = 54.5;
```

digunakan untuk mengisikan nilai 54,5 secara langsung ke variabel **d**. Adapun

```
pd = &d;
```

digunakan untuk memberikan alamat dari **d** ke **pd.** Dengan demikian **pd** menunjuk ke variabel **d.** Sedangkan pernyataan berikutnya

```
*pd = *pd + 10; (atau: *pd += 10; )
```

merupakan instruksi untuk mengubah nilai variabel **d** secara tak langsung. Perintah di atas berarti "jumlahkan yang ditunjuk ***pd*** dengan 10 kemudian berikan ke yang ditunjuk oleh **pd**", atau identik dengan pernyataan

```
d = d + 10;
```

Akan tetapi, seandainya tidak ada instruksi

```
pd = &d;
```

maka pernyataan

```
*pd = *pd + 10;
```

tidaklah sama dengan

```
d = d + 10;
```

Program 13.10

```
#include <conio.h>
#include <iostream>
```

```
using namespace std;

main()
{
	float d = 54.5f, *pd;
	cout <<"Isi d mula-mula = \n"<< d;
	pd = &d;
	*pd += 10;
	cout <<"Isi d sekarang = \n"<< d;
	getch();
	return 0;
}
```

Program 13.11

```
#include <conio.h>
#include <iostream>

using namespace std;

main()
{
	int i1, i2, *p1, *p2;
	i1 = 9;
	p1 = &i1;
	i2 = *p1 / 2 - 2 * 3;
	p2 = p1;
	cout <<"i1="<<i1;
	cout <<"i2="<<i2;
	cout <<"*p1="<<*p1;
	cout <<"*p2="<<*p2;
	getch();
	return 0;
}
```

Program 13.12

```
#include <conio.h>
#include <iostream>

using namespace std;

main()
{
```

```
        int count = 10, *temp, sum = 7;
        temp = &count;
        *temp = 32;
        temp = &sum;
        *temp = count;
        sum = *temp * 4;
        cout <<"count = %d, *temp = %d, sum = %d\n"<< count<<*temp<< sum;
        getch();
        return 0;
}
```

Program 13.13

```
#include <conio.h>
#include <iostream>

using namespace std;

main()
{
        int count = 13, sum = 9, *x, *y;
        x = &count;
        *x = 27;
        y = x;
        x = &sum;
        *x = count;
        sum = *x / 2 * 3;
        cout <<"count = %d, sum = %d, *x = %d, *y = %d\n"<< count<<sum<<*x<<*y;
        getch();
        return 0;
}
```

Program 13.14

```
#include <conio.h>
#include <iostream>

using namespace std;

int r, q = 7;
int go_crazy(int *, int *);
main()
{
        int *ptr1 = &q;
        int *ptr2 = &q;
```

```
        r = go_crazy(ptr1, ptr2);
        cout <<"q = %d, r = %d, *ptr1 = %d, *ptr2 = %d\n"<<q<< r<<*ptr1<<*ptr2;
        ptr2 = &r;
        go_crazy(ptr2, ptr1);
        cout <<"q = %d, r = %d, *ptr1 = %d, *ptr2 = %d\n"<< q<< r<< *ptr1<< *ptr2;
}
int go_crazy(int *p1, int *p2)
{
        int x = 5;
        r = 12;
        *p2 = *p1 * 2;
        p1 = &x;
        return *p1 * 3;
}
```

13.6. Array dan Pointer

Pada bahasan ini programer perlu mengerti perbedaan antara pointer dengan array. Keduanya menggunakan memori yang lokasinya ditentukan, dimana Identifier suatu array equivalen dengan alamat dari elemen pertama, sedangkan pointer equivalen dengan alamat elemen pertama yang ditunjuk. Perhatikan deklarasi veriabel sebagai berikut :

```
int numbers [20];
int * p;
```

maka deklarasi dibawah ini juga benar :

```
p = numbers;
```

dan **numbers** equivalen, dan memiliki sifat (*properties*) yang sama. Perbedaannya, user dapat menentukan nilai lain untuk pointer **p** dimana **numbers** akan selalu menunjuk nilai yang sama seperti yang telah didefinisikan. **p**, merupakan *variable pointer*, **numbers** adalah *constant pointer*. Karena itu walaupun instruksi diatas benar, tetapi tidak untuk instruksi dibawah ini :

```
numbers = p;
```

karena **numbers** adalah array (constant pointer), dan tidak ada nilai yang dapat diberikan untuk identifier konstant (*constant identifiers*). Perhatikan program dibawah ini:

Program 13.15

```
#include <conio.h>
#include <iostream>

using namespace std;
```

```
int main ()
{
        int numbers[5];
        int * p;
        p = numbers;
        *p = 10;
        p++;
        *p = 20;
        p = &numbers[2]; *p = 30;
        p = numbers + 3; *p = 40;
        p = numbers; *(p+4) = 50;
        for (int n=0; n<5; n++)
        cout << numbers[n] << ", ";
        getch();
        return 0;
}
```

Program tersebut setelah dieksekusi maka akan mendapatkan keluaran sebagai berikut:

```
Output : 10, 20, 30, 40, 50,
```

13.6.1. Pointer dan Array (pointer to array)

Hubungan antara pointer dan array pada C sangatlah erat. Sebab sesungguhnya array secara internal akan diterjemahkan dalam bentuk pointer. Pembahasan berikut akan memberikan gambaran hubungan antara pointer dan array. Misalnya dideklarasikan di dalam suatu fungsi

```
static int tgl_lahir[3] = { 01, 09, 64 };
```

dan

```
int *ptgl;
```

Kemudian diberikan instruksi

```
ptgl = &tgl_lahir[0];       //pointer to array of integer
```

maka **ptgl** akan berisi alamat dari elemen array **tgl_lahir** yang berindeks nol. Instruksi di atas bisa juga ditulis menjadi

```
ptgl = tgl_lahir;
```

sebab nama array tanpa tanda kurung menyatakan alamat awal dari array. Sesudah penugasan seperti di atas,

```
*ptgl
```

dengan sendirinya menyatakan elemen pertama (berindeks sama dengan nol) dari array

tgl_lahir.

13.6.2. Pointer dan String

Beberapa contoh hubungan pointer dan string ditunjukkan pada program berikut.

Program 13.16

```
#include <conio.h>
#include <iostream>

using namespace std;

main()
{
        /* pnegara menunjuk konstanta string "INDONESIA" */
        char *pnegara = "INDONESIA";
        cout<<"String yang ditunjuk oleh pnegara = ";
        puts(pnegara);          // printf("%s\n",pnegara);
        getch();
        return 0;
}
```

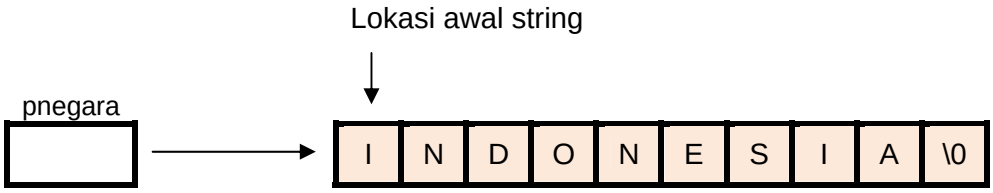
Hasil keluaran program diatas adalah sebagai berikut

```
String yang ditunjuk oleh pnegara = INDONESIA
```

Pada program di atas,

```
char *pnegara = “INDONESIA”;
```

akan menyebabkan kompiler mengalokasikan variabel **pnegara** sebagai variabel pointer yang menunjuk ke obyek bertipe *char* dan menempatkan konstanta **“INDONESIA”** dalam suatu memori, kemudian pointer **pnegara** akan menunjuk ke lokasi string **“INDONESIA”.**



Gambar 13.11. Pointer menunjuk data

Pernyataan di atas akan menyerupai pernyataan seperti dibawah ini:

```
char negara[] = “INDONESIA”;
```

tetapi sebenarnya kedua pernyataan inisialisasi di depan tidaklah tepat sama. Sebab **pnegara** adalah pointer (menyatakan alamat) yang dengan mudah dapat diatur agar menunjuk ke string lain (bukan string “INDONESIA”), sedangkan negara adalah array (array menyatakan alamat yang konstan, tak dapat diubah). Jadi, ada beberapa cara penulisan ekspresi variabel yang menghasilkan sebuah address memory, di antaranya adalah :

1. Menuliskan ampersand di depan sebuah variabel 'normal'
   Misalnya a dideklarasikan sebagai sebuah variabel normal, maka &a akan menghasilkan adress memory dari variabel a tersebut
2. Menuliskan nama dari sebuah variabel pointer
   Misalnya a dideklarasikan sebuah variabel sebagai int *a, maka a akan berisi sebuah address memory yang tersimpan pada variabel a (setelah ada sebuah address memory yang di-assign ke variabel a tersebut)
3. Menuliskan nama sebuah variabel array tanpa indexnya
   Misalnya dideklarasikan
   - float nilai[5], maka penulisan nilai artinya sama dengan &nilai[0]
   - int b[3][4], maka b[3] adalah sama dengan &b[3][0] dan b[0] adalah sama dengan &b[0][0].

   Bisa disimpulkan bahwa sebuah variabel array yang dituliskan tanpa kurung siku indexnya adalah sama dengan address memory dari elemen pertama array tersebut.
1. Hubungan antara pointer dan array. Suatu nama array yang ditulis tanpa dengan indeksnya menunjukkan alamat elemen pertama dari array (versi 1).

Program 13.17

```
#include <conio.h>
#include <iostream>

using namespace std;

main()
{
    static int tgl_lahir[] = {16, 4, 1974};
    int *ptgl;
    cout <<"\nNilai yang ditunjuk oleh ptgl = "<< *ptgl;
    cout <<"\nNilai dari tgl_lahir[0] = "<< tgl_lahir[0];
    getch();
    return 0;
```

```
}
```

2. Hubungan antara pointer dan array. Suatu nama array yang ditulis tanpa dengan indeksnya menunjukkan alamat elemen pertama dari array (versi 2).

Program 13.18

```
#include <conio.h>
#include <iostream>

using namespace std;

main()
{
    static int tgl_lahir[] = {16, 4, 1974};
    int *ptgl, i;
    ptgl = tgl_lahir;
    printf("Nilai yang ditunjuk oleh ptgl = %d\n", *ptgl);
    for (i=0; i<3; i++)
    printf("Nilai dari tgl_lahir[i] = %d\n", *(ptgl+i));
    getch();
    return 0;
}
```

3. Hubungan antara pointer dan array. Suatu nama array yang ditulis tanpa dengan indeksnya menunjukkan alamat elemen pertama dari array (versi 3).

Program 13.19

```
#include <conio.h>
#include <iostream>

using namespace std;

main()
{
    static int tgl_lahir[] = {16, 4, 1974};
    int i;
    int *ptgl;
    ptgl = tgl_lahir;
    printf("Nilai yang ditunjuk oleh ptgl = %d\n", *ptgl);
    for (i=0; i<3; i++)
    printf("Nilai dari tgl_lahir[i] = %d\n", *ptgl++);
```

```
    getch();
    return 0;
}
```

Analisis dan jelaskan perbedaan antara aplikasi pada nomor 1, 2 dan 3

4. Menukarkan isi 2 string tanpa pemakaian pointer.

Program 13.20

```
#include <stdio.h>
#include <string.h>
#define PANJANG 20

char nama1[PANJANG] = "AHMAD";
char nama2[PANJANG] = "RIFDA";

main()
{
    char namax[PANJANG];
    puts("SEMULA : ");
    printf("nama1 --> %s\n", nama1);
    printf("nama2 --> %s\n", nama2);
    strcpy(namax, nama1);
    strcpy(nama1, nama2);
    strcpy(nama2, namax);
    puts("KINI : ");
    printf("nama1 --> %s\n", nama1);
    printf("nama2 --> %s\n", nama2);
}
```

5. Menukarkan isi 2 string dengan fasilitas pointer.

Program 13.21. Menukarkan isi 2 string dengan fasilitas pointer

```
#include <stdio.h>
#include <string.h>
char *nama1 = "AHMAD";
char *nama2 = "RIFDA";

main()
{
    char *namax;
    puts("SEMULA : ");
    printf("nama1 --> %s\n", nama1);
```

```
    printf("nama2 --> %s\n", nama2);
    namax = nama1;
    nama1 = nama2;
    nama2 = namax;
    puts("KINI : ");
    printf("nama1 --> %s\n", nama1);
    printf("nama2 --> %s\n", nama2);
}
```

Analisis dan jelaskan perbedaan antara aplikasi pada nomor 4 dengan nomor 5

6. Mengamati persamaan penggunaan *variable index* pada *array* dan *variable index* pada *pointer*, untuk menunjuk suatu nilai data di dalam suatu *variable array*.

Program13.22.

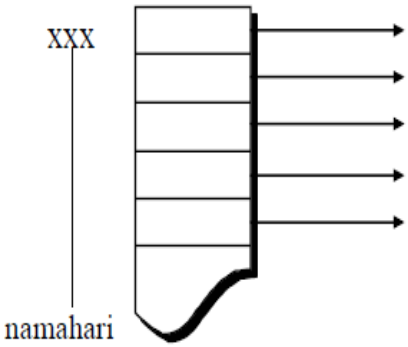
```
main()
{
    int nilai[10]={86,75,98,66,56,76,80,95,70,60};
    int index, *ip;
    printf("Mencetak menggunakan array\n");
    printf("Daftar nilai siswa\n\n");
    for(index=0; index<10; index++)
    printf("%3d",nilai[index]);
    puts("\n");
    printf("Mencetak menggunakan pointer dan index\n");
    printf("Daftar nilai siswa\n\n");
    for(index=0; index<10; index++)
    printf("%3d",*(nilai+index));
    puts("\n");
    printf("Mencetak menggunakan pointer\n");
    printf("Daftar nilai siswa\n\n");
    ip=&nilai;
    for(index=0; index<10; index++)
    printf("%3d",*ip++);
}
```

13.6.3. Array dari Pointer (Array of Pointer)

Suatu array bisa digunakan untuk menyimpan sejumlah pointer. Sebagai contoh:

```
char *namahari[10];
```

merupakan pernyataan untuk mendeklarasikan array pointer. Array **namahari** terdiri dari 10 elemen berupa pointer yang menunjuk ke data bertipe *char*.



Gambar 13.12. Array pointer

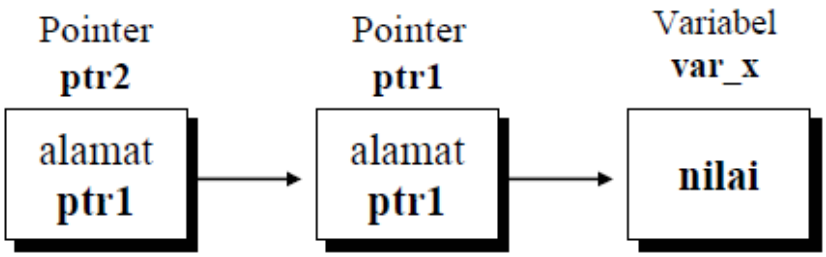
Array pointer bisa diinisialisasi sewaktu pendeklarasian. Sebagai contoh:

```
static char *namahari[] =
{“Senin”,
“Selasa”,
“Rabu”,
“Kamis”,
“Jumat”,
“Sabtu”,
“Minggu”};
```

Perhatikan contoh dibawah ini,

```
namahari[0] berisi alamat yang
menunjuk ke string “Senin”.
namahari[1] berisi alamat yang
menunjuk ke string “Selasa”.
namahari[2] berisi alamat yang
menunjuk ke string “Rabu”.
dan sebagainya.
```

Suatu pointer bisa saja menunjuk ke pointer lain. Gambar berikut memberikan contoh mengenai pointer menunjuk pointer.



Gambar 13.13. Pointer yang menunjuk pointer

Untuk membentuk rantai pointer seperti pada gambar di atas, pendeklarasian yang diperlukan berupa

```
int var_x;
```

```
int *ptr1;
int **ptr2;
```

Perhatikan pada deklarasi di depan: **var_x** adalah variabel bertipe int. **ptr1** adalah variabel pointer yang

menunjuk ke data bertipe *int.* **ptr2** adalah variabel pointer yang menunjuk ke pointer *int.* (itulah sebabnya deklarasinya berupa *int **ptr2;*). Agar **ptr1** menunjuk ke variabel **var_x**, perintah yang diperlukan berupa ptr1 = &var_x; Sedangkan supaya **ptr2** menunjuk ke **ptr1**, instruksi yang diperlukan adalah

```
ptr2 = &ptr1;
```

program 13.23

```
#include <stdio.h>
main()
{
        static char *days[] = {"Sunday", "Monday", "Tuesday",
"Wednesday","Thursday",        "Friday", "Saturday"};
        int i;
        for( i = 0; i < 6; ++i )
        printf( "%s\n", days[i]);
}
```

Program 13.24. Pointer yang menunjuk ke pointer yang lain.

```
#include <stdio.h>
main()
{
        int a, *b, **c;
        a = 155;
        b = &a;
        c = &b;
        printf("Nilai a = %d atau %d atau %d\n", a, *b, **c);
        printf("b = %p = alamat a di memori\n", b);
        printf("c = %p = alamat b di memori\n", c);
        printf("alamat c di memori = %p\n", &c);
}
```

Program 13.25. Pointer yang menunjuk ke pointer yang lain.

```
#include <stdio.h>
main()
{
        int var_x = 273;
        int *ptr1;
        int **ptr2;
        ptr1 = &var_x;
        ptr2 = &ptr1;
        printf("Nilai var_x = *ptr1 = %d\n", *ptr1);
```

```
        printf("Nilai var_x = **ptr2 = %d\n\n", **ptr2);
        printf("ptr1 = &var_x = %p\n", ptr1);
        printf("ptr2 = &ptr1 = %p\n", ptr2);
        printf(" &ptr2 = %p\n", &ptr2);
}
```

Program 13.26.

```
#include <stdio.h>
main()
{
    int a, *b, **c;
    a = 1975;
    b = &a;
    c = &b;
    printf("Nilai a = %d atau %d atau %d\n", a, *b, **c);
    printf("b = %p = alamat a di memori\n", b);
    printf("c = %p = alamat b di memori\n", c);
    printf("alamat c di memori = %p\n", &c);
}
```

13.7. Pointer dalam Fungsi

Pointer dan kaitannya dengan fungsi yang akan dibahas berikut meliputi: Pointer sebagai parameter fungsi dan pointer sebagai keluaran fungsi

13.7.1. Pointer Sebagai Parameter Fungsi

Penerapan pointer sebagai parameter yaitu jika diinginkan agar nilai suatu variabel internal dapat diubah oleh fungsi yang dipanggil. Perhatikan contoh fungsi berikut dibawah ini.

```
void naikkan_nilai (int *x, int *y)
{
*x = *x + 2;
*y = *y + 2;
}
```

Fungsi di atas dimaksudkan agar kalau dipanggil, variabel yang berkenaan dengan parameter aktual dapat diubah nilainya, masing-masing dinaikkan sebesar 2. Contoh pemanggilan :

```
naikkan_nilai(&a, &b);
```

Perhatikan, dalam hal ini variabel **a** dan **b** harus ditulis diawali operator alamat (**&**) yang berarti menyatakan alamat variabel, sebab parameter fungsi dalam pendefinisian berupa pointer.

13.7.2. Pointer Sebagai Fungsi Keluaran

Suatu fungsi dapat dibuat agar keluarannya berupa pointer. Misalnya, suatu fungsi menghasilkan

keluaran berupa pointer yang menunjuk ke string nama_bulan, seperti pada contoh berikut.

```
char *nama_bulan(int n)
{
static char *bulan[]=
{"Kode bulan salah", "Januari",
"Februari", Maret",
"April", "Mei", "Juni", "Juli", "Agustus",
"September", "Oktober",
"Nopember","Desember"
};
return ( (n<1 | | n>12) ? bulan[0] :
bulan[n] );
}
```

Pada definisi fungsi di atas,

```
char *nama_bulan()
```

menyatakan bahwa keluaran fungsi **nama_bulan()** berupa pointer yang menunjuk ke obyek char (atau string). Dalam fungsi **nama_bulan()**, mula-mula array bernama **bulan** dideklarasikan dan sekaligus diinisialisasi agar menunjuk sejumlah string yang menyatakan nama bulan. Di bagian akhir fungsi, pernyataan

```
return ( (n<1 || n>12) ? bulan[0] :
bulan[n] );
```

menyatakan bahwa hasil fungsi berupa pointer yang menunjuk ke

- string "Kode bulan salah" (bulan[0]) jika masukan fungsi n<1 atau n>12
- bulan[n] untuk n yang terletak antara 1 sampai dengan 12.

Program 13.27. Fungsi dengan argumen berupa pointer.

```
#include <stdio.h>
void naikkan_nilai(int *x, int *y);
main()
{
int a = 3;
int b = 7;
printf("SEMULA : a = %d b = %d\n", a, b);
naikkan_nilai(&a, &b);
printf("KINI : a = %d b = %d\n", a, b);
}
void naikkan_nilai(int *x, int *y)
{
*x = *x + 2;
*y = *y + 2;
}
```

Program 13.28. Fungsi dengan keluaran berupa pointer yang menunjuk string.

```
#include <stdio.h>
char *nama_bulan(int n);
main()
```

```
{
int bl;
printf("Bulan 1..12 : ");
scanf("%d", &bl);
printf("Bulan ke-%d adalah %s\n", bl, nama_bulan(bl));
}
char *nama_bulan(int n)
{
static char *bulan[] =
{
"Kode bulan salah",
"Januari",
"Februari",
"Maret",
"April",
"Mei",
"Juni",
"Juli",
"Agustus",
"September",
"Oktober",
"November",
"Desember"
};
return ((n<1||n>12) ? bulan[0] : bulan[n]);
}
```

Program 13.29. Mengakses isi suatu variabel melalui pointer.

```
#include <stdio.h>

main()
{
	int y, x = 87; /* x & y bertipe int */
	int *px;
	/* var pointer yg menunjuk ke data yang bertipe int */
	px = &x; /* px diisi dgn alamat dari variabel x */
	y = *px; /* y diisi dgn nilai yang ditunjuk oleh px */
	printf("Alamat x = %p\n", &x);
	printf("Isi px = %p\n", px);
	printf("Isi x = %d\n", x);
	printf("Nilai yang ditunjuk oleh px = %d\n", *px);
	printf("Nilai y = %d\n", y);
```

```
}
```

Program 13.30. Mengakses isi suatu variabel melalui pointer. Tipe variabel pointer dan tipe data yang ditunjuk harus sejenis.

```
#include <stdio.h>
main()
{
	int *pu;
	int nu;
	int u = 1234;
	pu = &u;
	nu = *pu;
	printf("Alamat dari u = %p\n", &u);
	printf("Isi pu = %p\n", pu);
	printf("Isi u = %d\n", u);
	printf("Nilai yang ditunjuk oleh pu = %d\n", *pu);
	printf("Nilai nu = %d\n", nu);
}
```

Program 13.31. Mengubah isi suatu variabel melalui pointer.

```
#include <stdio.h>
main()
{
	float d = 54.5f, *pd;
	printf("Isi d mula-mula = %g\n", d);
	pd = &d;
	*pd += 10;
	printf("Isi d sekarang = %g\n", d);
}
```

Program 13.32. Hubungan antara pointer dan array. Suatu nama array yang ditulis tanpa dengan indeksnya menunjukkan alamat elemen pertama dari array (versi 1).

```
#include <stdio.h>
main()
{
	static int tgl_lahir[] = {16, 4, 1974};
	int *ptgl;
	printf("Nilai yang ditunjuk oleh ptgl = %d\n", *ptgl);
```

```
    printf("Nilai dari tgl_lahir[0] = %d\n", tgl_lahir[0]);
}
```

Program 13.33. Hubungan antara pointer dan array. Suatu nama array yang ditulis tanpa dengan indeksnya menunjukkan alamat elemen pertama dari array (versi 2).

```
#include <stdio.h>
main()
{
    static int tgl_lahir[] = {16, 4, 1974};
    int *ptgl, i;
    ptgl = tgl_lahir;
    printf("Nilai yang ditunjuk oleh ptgl = %d\n", *ptgl);
    for (i=0; i<3; i++)
    printf("Nilai dari tgl_lahir[i] = %d\n", *(ptgl+i));
}
```

Program 13.34. Hubungan antara pointer dan array. Suatu nama array yang ditulis tanpa dengan indeksnya menunjukkan alamat elemen pertama dari array (versi 3).

```
#include <stdio.h>
main()
{
    static int tgl_lahir[] = {16, 4, 1974};
    int i;
    int *ptgl;
    ptgl = tgl_lahir;
    printf("Nilai yang ditunjuk oleh ptgl = %d\n", *ptgl);
    for (i=0; i<3; i++)
    printf("Nilai dari tgl_lahir[i] = %d\n", *ptgl++);
}
```

Program 13.35. Menukarkan isi 2 string tanpa pemakaian pointer.

```
#include <stdio.h>
#include <string.h>
#define PANJANG 20
char nama1[PANJANG] = "GATUTKACA";
char nama2[PANJANG] = "HANOMAN";

main()
```

```
{
	char namax[PANJANG];
	puts("SEMULA : ");
	printf("nama1 --> %s\n", nama1);
	printf("nama2 --> %s\n", nama2);
	strcpy(namax, nama1);
	strcpy(nama1, nama2);
	strcpy(nama2, namax);
	puts("KINI : ");
	printf("nama1 --> %s\n", nama1);
	printf("nama2 --> %s\n", nama2);
}
```

Program 13.36. Menukarkan isi 2 string dengan fasilitas pointer.

```
#include <stdio.h>
#include <string.h>

char *nama1 = "GATUTKACA";
char *nama2 = "HANOMAN";
main()
{
	char *namax;
	puts("SEMULA : ");
	printf("nama1 --> %s\n", nama1);
	/* nama1 adl pointer yg menunjuk ke string GATUTKACA*/
	printf("nama2 --> %s\n", nama2);
	/* nama2 adl pointer yg menunjuk ke string HANOMAN*/
	namax = nama1;
	nama1 = nama2;
	nama2 = namax;
	puts("KINI : ");
	printf("nama1 --> %s\n", nama1);
	printf("nama2 --> %s\n", nama2);
}
```

Program 13.37. Pointer yang menunjuk ke pointer yang lain.

```
#include <stdio.h>
main()
{
	int a, *b, **c;
	a = 155;
	b = &a;
	c = &b;
```

```
        printf("Nilai a = %d atau %d atau %d\n", a, *b, **c);
        printf("b = %p = alamat a di memori\n", b);
        printf("c = %p = alamat b di memori\n", c);
        printf("alamat c di memori = %p\n", &c);
}
```

Program 13.38. Pointer yang menunjuk ke pointer yang lain.

```
#include <stdio.h>
main()
{
        int var_x = 273;
        int *ptr1;
        int **ptr2;
        ptr1 = &var_x;
        ptr2 = &ptr1;
        printf("Nilai var_x = *ptr1 = %d\n", *ptr1);
        printf("Nilai var_x = **ptr2 = %d\n\n", **ptr2);
        printf("ptr1 = &var_x = %p\n", ptr1);
        printf("ptr2 = &ptr1 = %p\n", ptr2);
        printf(" &ptr2 = %p\n", &ptr2);
}
```

Program 13.39. Fungsi dengan argumen berupa pointer.

```
#include <stdio.h>
void naikkan_nilai(int *x, int *y);
main()
{
        int a = 3;
        int b = 7;
        printf("SEMULA : a = %d b = %d\n", a, b);
        naikkan_nilai(&a, &b);
        printf("KINI : a = %d b = %d\n", a, b);
}

void naikkan_nilai(int *x, int *y)
{
        *x = *x + 2;
        *y = *y + 2;
}
```

Program 13.40. Fungsi dengan keluaran berupa pointer yang menunjuk string.

```
#include <stdio.h>
char *nama_bulan(int n);
```

```
main()
{
      int bl;
      printf("Bulan 1..12 : ");
      scanf("%d", &bl);
      printf("Bulan ke-%d adalah %s\n", bl, nama_bulan(bl));
}

char *nama_bulan(int n)
{
      static char *bulan[] =
      {
          "Kode bulan salah",
          "Januari",
          "Februari",
          "Maret",
          "April",
          "Mei",
          "Juni",
          "Juli",
          "Agustus",
          "September",
          "Oktober",
          "November",
          "Desember"
      };
      return ((n<1||n>12) ? bulan[0] : bulan[n]);
}
```

13.8. Fungsi Pointer ke Static Class Member Function

penggunaan function pointer pada C++ dibatasi, yaitu function pointer tidak boleh menunjuk pada function yang berada dalam sebuah class (class member function) kecuali function tersebut berjenis static.

```
class A
{
  private :
  public  : static void WriteString(char* String);
};
```

```
void A::WriteString(char* String)
{
   printf("A::WriteString() : %s",String);
}

class B
{
   private :
   public  : void Func(void (*Writer)(char* str));
};

void B::Func(void (*Writer)(char* str))
{
   Writer("class B writing \"hello...!\"");
}

int main(void)
{
   A* a = new A;
   B* b = new B;

   b->Func(a->WriteString);

   delete a;
   delete b;
}
```

Namun terkadang yang menjadi masalah adalah, sebuah member function yang berjenis static, tidak memiliki akses ke member dari class, sebagai contoh bila kita rubah class A menjadi :

```
class A
{
   private : int privatenumber;
   public  : static void WriteString(char* String);
};

void A::WriteString(char* String)
{
   printf("A::privatenumber = %d",privatenumber);
   printf("A::WriteString() : %s",String);
}
```

maka compiler menyatakan error, bahwa fungsi static tidak dapat mengakses member dari class A. Namun bila kita rubah fungsi WriteString menjadi non-static (dengan menghapus keyword "static") maka fungsi WriteString tidak dapat di pass melalui argument ke B::func (karena function pointer tidak boleh menunjuk fungsi yang merupakan member class)

13.10.Fungsi Pointer pada Class anggota Fungsi Non-static

Salah satu cara agar kita dapat membuat function pointer yang menunjuk ke non-static class member function bekerja adalah dengan cara memberikan object dari class yang membernya ingin diakses. Pada masalah sebelumnya A::WriteString memerlukan akses ke private member dari A yaitu mengakses int A::Privatenumber, maka solusinya kita pass object dari class A ke argumen dari A::WriteString kode yang sebelumnya dirubah menjadi :

```
class A
{
   private : int privatenumber;
   public  : static void WriteString(char* String,A* Self);
};

void A::WriteString(char* String,A* Self)
{
   printf("A::privatenumber = %d\n",Self->privatenumber);
   printf("A::WriteString() : %s",String);
}

class B
{
   private :
   public  : void Func(void (*Writer)(char* str,A* Self),A* NeedToPass);
};

void B::Func(void (*Writer)(char* str,A* Self),A* NeedToPass)
{
   Writer("class B writing \"hello...!\"",NeedToPass);
}

int main(void)
{
```

```
   A* a = new A;
   B* b = new B;

   b->Func(a->WriteString,a);

   delete a;
   delete b;
}
```

Terdapat cara lain menggunakan member function pointer selain dengan cara di atas. Beberapa hal yang harus diperhatikan adalah:

- Deklarasi Member function harus dengan menggunakan operator ::*
- Member function pointer memerlukan objek atau pointer ke objek supaya dapat digunakan.
- Memanggil member function pointer menggunakan operator .* atau ->* , tergantung apakah digunakan dengan objek (atau reference) atau dengan pointer. Yang menjadi masalah di sini adalah kedua operator tersebut memiliki prioritas yang lebih rendah daripada operator aplikasi fungsi, sehingga anda akan selalu memerlukan tanda kurung untuk menggunakan member function pointer.
- Assignment ke member function pointer harus selalu menggunakan operator &. Hal ini berbeda dengan function pinter biasa dimana operator & adalah optional. Beberapa kompiler akan menerima konstruk assignment tanpa operator &, tapi itu tidak standard. Kompiler yang sesuai dengan standard C++98 akan mereject assignment member function pointer tanpa operator &. untuk lebih jelasnya anda dapat melihat contoh program di bawah:

```
#include <iostream>

using std::cout;
using std::endl;

class SomeMumboJumboClass{
      int i;
public:
      void a_method();
      SomeMumboJumboClass(int p):i(p){}
};

void SomeMumboJumboClass::a_method(){
```

```
	cout << "the mumbo jumbo class has a member variable with value = "<< i<<
endl;
}

typedef void (SomeMumboJumboClass::* MemberFunctionptr)() ; // ini cara
mendekrarasikan typedef dari non-static member function

int main(){
	SomeMumboJumboClass c(2);  // deklarasikan sebuah objek

	void (SomeMumboJumboClass::* the_mem_fun_ptr)();
	the_mem_fun_ptr = &SomeMumboJumboClass::a_method;
	c.a_method(); // pemanggilan tanpa function pointer
	(c.*the_mem_fun_ptr)(); // pemanggilan dengan function pointer

	SomeMumboJumboClass* d = new SomeMumboJumboClass(3); // ini adalah
pointer ke objek
	d->a_method(); // pemanggilan normal
	(d->* the_mem_fun_ptr)(); // pemanggilan dengan pointer fungsi

	MemberFunctionptr the_mem_fun_ptr2 =
&SomeMumboJumboClass::a_method; // deklarasi dan definisi member function
pointer
		(c.* the_mem_fun_ptr2)(); // bisa digunakan seperti ini
	(d->* the_mem_fun_ptr2) (); // atau seperti ini

	delete d;
}
```

13.11. Soal Latihan

Jawablah soal latihan dibawah ini dengan baik dan benar.

1. Apa yang dimaksud dengan pointer pada bahasa C++
2. Gambarkan pengaturan memori pemrograman menggunakan pointer
3. Bagaimana cara mendeklarasikan pointer
4. Bagaimana menginisialisasi pointer
5. Buatlah pemrograman string menggunakan pointer
6. Buatlah program yang menggunakan fungsi pada pointer
7. Bagaimana cara mengubah dan mengisi isi pointer
8. Apakah perbedaan array dan pointer

BAB 14
CLASS

14.1. Obyek dan Class

Dalam C atau bahasa pemrograman prosedural lainnya, pemrogramannya berorientasi kepada aksi, sedangkan pemrograman C++ cenderung berorientasi pada obyek. Disamping itu, unit program dalam C adalah fungsi, tetapi di dalam C++, unit program adalah kelas (*class*) dimana suatu obyek (*object*) secara mendekati kenyataan dapat diciptakan.

Kelas di dalam C++ merupakan pengembangan dari notasi *struct* di dalam C biasa, yaitu struktur untuk mendefinisikan elemen-elemen data. Misalnya didefinisikan data tentang waktu yang terdiri dari jam, menit, dan detik sebagai berikut:

```
struct Time {
    int hour;
    // nilai jam 0 - 23
    int minute;
    // nilai menit 0 - 59
    int second;
    // nilai detik 0 - 59
};
```

Definisi Time ini terdiri dari tiga buah anggota data (*data members*) yang semuanya bertipe int, yaitu hour, minute, dan second. Anggota struktur dapat terdiri dari berbagai macam tipe, kecuali tipe Time itu sendiri. Berikut adalah contoh-contoh deklarasi variabel dan akses anggota data dari struktur

```
Time timeObyek, timeArray[10],
```

```
*timePtr;
cout << timeObyek.hour;
// mengakses data hour dari obyek
cout << timePtr -> hour;
// mengakses data hour dari obyek
// yang ditunjuk oleh pointer timePtr
```

Untuk melakukan proses pengolahan terhadap obyek data ini diperlukan fungsi-fungsi yang ditulis secara terpisah dari definisi strukturnya. Sebagai contoh, fungsi berikut disusun untuk menuliskan obyek **Time** dengan bentuk **hh:mm:ss**.

```
void printTime (const Time &t)
{
      cout << (t.hour < 10 ? "0" : "") <<
t.hour << ":"
      << (t.minute < 10 ? "0" : "") <<
t.minute << ":"
      << (t.second < 10 ? "0" : "") <<
t.second << ":" ;
}
```

Pada contoh ini terlihat adanya pemisahan antara data dan prosedur/fungsi, yang berbeda dengan konsep pemrograman beorientasi obyek dimana antara keduanya dibungkus ke dalam satu kesatuan dengan menggunakan tipe ***class***. Pembungkusan (***encapsulation***) inilah yang merupakan salah satu ciri utama dari pemrograman berorientasi objek.

14.2. Tipe Class

Kata kunci ***class*** dalam C++ digunakan untuk memodelkan suatu obyek terdiri dari dua anggota, yaitu atribut yang direpresentasikan sebagai anggota data (***data members***) dan sifat atau operasi-operasi atau prosedur-prosedur yang direpresentasikan sebagai fungsi anggota (*member functions*). Fungsi anggota sering disebut dengan metode pada bahasa pemrograman berorientasi objek lainnya dan digunakan untuk memberi pesan pada obyek yang bersangkutan. Setiap anggota dari obyek ini dikelompokkan berdasarkan sifat akses terhadap anggotaanggotanya, baik dari dalam obyek itu sendiri maupun dari obyek lainnya. Ada tiga sifat yang dapat diberikan, yaitu *public*, *private*, dan *protected*. Perhatikan contoh berikut:

```
class Time {
    public:
    Time();
    // default constructor
    void setTime(int, int, int);
    // member function
    voit print();
    ~Time();
    // destructor
    private:
    int hour;
    // data member
    int minute;
    int second;
};
```

Anggota kelompok private hanya dapat diakses oleh fungsi anggota dari kelas itu sendiri dan kelas lain yang mempunyai hubungan friend (akan dijelaskan pada bab berikutnya). Hal ini berbeda dengan kelompok public yang dapat diakses dari bagian-bagian lainnya dari program, sedangkan kelompok

protected hanya dapat diakses oleh fungsi anggota dari kelas yang bersangkutan dan kelas-kelas lain yang menjadi turunannya.

Pembatasan akses itu merupakan information hiding yang berguna antara lain jika representasi data dari kelas berubah, hanya fungsi anggota (bukan program dari pengguna) yang perlu diambil, dan jika ada kesalahan dalam manipulasi anggota data dari kelas, hanya fungsi anggota yang perlu di-debug, bukan seluruh program.

Fungsi anggota yang namanya sama dengan nama kelas disebut default constructor, dan fungsi anggota yang namanya terdiri tkita tilde (“~“) diikuti dengan nama kelas disebut destructors. Constructor akan secara otomatis dipanggil oleh kompilator untuk inisialisasi obyek, sedangkan destructor akan secara otomatis dipanggil oleh kompilator untuk dealokasi memori. Nama kelas akan berlaku sebagai kata kunci penentu untuk suatu tipe data baru, misalnya dengan kelas Time tersebut dapat dibuat deklarasi variabel sebagai berikut:

```
Time timeObyek1,
timeObyek2;
```

Pada saat deklarasi inilah diproses fungsi anggota **Time()** yang merupakan default constructor dari kelas ini. Namun demikian, kelas tersebut belum dapat digunakan karena implementasi setiap fungsi anggota belum dibuat. Perhatikan contoh implementasi berikut dibawah ini:

```
Time :: Time()
{
    hour = minute = second = 0;
}
void Time :: setTime( int h = 0, int m = 0,
int s = 0 )
{
    hour = h;
    minute = m;
    second = s;
}

voit Time :: print()
{
    cout << (hour < 10 ? "0" : "")
    << hour << ":"
    << (minute < 10 ? "0" : "")
    << minute << ":"
    << (second < 10 ? "0" : "")
    << second << ":" ;
}
```

Operator “::“ adalah scope-operator, yang menunjukkan bahwa fungsi yang dituliskan di belakangnya merupakan anggota dari kelas yang ditulis di depannya. Dengan demikian, misalnya void Time :: setTime (h = 0, m = 0, s = 0) menunjukkan bahwa fungsi setTime merupakan anggota dari kelas Time. Suatu fungsi dalam C++ dapat dipanggil dengan nol atau lebih argumen, seperti contoh Time() yang tidak mempunyai argumen, dan setTime (h = 0, m = 0, s = 0) mempunyai tiga argumen dengan masing-masing mempunyai nilai *default*. Artinya, fungsi itu dapat dipanggil dengan beberapa cara seperti contoh berikut:

```
setTime( 10, 30, 25 );
// argumen h=10, m=30, s=25
setTime( 10, 30);
```

```
// argumen h=10, m=30, s=0
setTime();
// argumen h=0, m=0, s=0
```

Dengan definisi dan deklarasi kelas Time yang telah dibuat, maka dapat digunakan sebagai tipe baru seperti pada program berikut (misalnya definisi dan deklarasi kelas Time telah disimpan pada file **time.h**):

```
#include <iostream>
#include "time.h"

main() {
Time t1;
t1.setTime(10,30,25);
// obyek t1 memiliki hour=10,
minute=30, second=25
t1.
```

Mekanisme kelas (*class*) memungkinkan pengguna mendefinisikan tipe data abstrak (ADT). Suatu kelas mempunyai empat atribut, yaitu:

- Anggota-anggota data (*data members*) yang merupakan representasi dari kelas. Suatu kelas dapat mempunyai nol atau lebih data member dengan tipe apa saja.
- Fungsi-fungsi anggota (*members function*) yaitu operasi-operasi yang dapat diterapkan pada obyek-obyek dari kelas itu. Sebuah kelas dapat mempunyai nol atau lebih fungsi anggota.
- Tingkat akses.
  Setiap anggota suatu kelas dapat ditentukan tingkat aksesnya sebagai **private**, **protected** atau **public**. Tingkat-tingkat ini mengontrol akses ke anggota kelas. Biasanya representasi kelas bersifat private, sedang operasi-operasinya bersifat public. Spesifikasi private/public ini membentuk *information hidding*.
- Nama kelas yang berlaku sebagai type-specifier.

14.3. Deklarasi Class

Class adalah metode logical untuk organisasi data dan fungsi dalam struktur yang sama. Class dideklarasikan menggunakan keyword **class**, yang secara fungsional sama dengan keyword **struct**, tetapi dengan kemungkinan penyertaan fungsi sebagai anggota, formatnya sebagai berikut :

```
class class_name {
permission_label_1:
member1;
permission_label_2:
member2;
...
} object_name;
```

Dimana **class_name** adalah nama class(user defined type) dan field optional **object_name** adalah satu atau beberapa identifier objek yang valid. Body dari deklarasi berisikan **members**, yang dapat berupa data ataupun deklarasi fungsi, dan **permission labels** (optional), dapat berupa satu dari tiga keyword berikut : **private:**, **public:** atau **protected:**. Digunakan untuk menentukan batasan akses terhadap members yang ada :

1. **private** , anggota class dapat diakses dari danggota lain pada kelas yang sama atau dari class "friend".
2. **protected ,** anggota dapat diakses dari anggota class yang sama atau class friend , dan juga dari anggota class turunannya(derived).
3. **public** , anggota dapat diakses dari class manapun.

Default permission label : **private**
Contoh :

```
class CRectangle {
int x, y;
public:
void set_values (int,int);
int area (void);
} rect;
```

Deklarasi class **CRectangle** dan object bernama **rect**. Class ini berisi empat anggota: dua variable bertipe **int** (**x** and **y**) pada bagian **private** (karena private adalah default permission) dan dua fungsi pada bagian **public** : **set_values()** dan **area()**, dimana hanya dideklarasikan propotype_nya.

Perhatikan perbedaan antara nama classdan nama object. Pada contoh sebelumnya **CRectangle** adalah nama class(contoh, user-defined type), dan **rect** adalah object dari tipe **CRectangle**. Sama halnya dengan deklarasi berikut :

```
int a;
```

int adalah nama class (type)
dan **a** adalah nama object(variable).

Program 14.1

```
#include <conio.h>
#include <iostream>
using namespace std;

class CRectangle {
      int x, y;
      public:
      void set_values (int,int);
      int area (void) {return (x*y);}
      };

void CRectangle::set_values (int a, int b) {
      x = a;
      y = b;
      }
      int main () {
      CRectangle rect;
      rect.set_values (3,4);
      cout << "area: " << rect.area();
      getch();
```

```
    return 0;
}
```

Keluaran program diatas adalah sebagai berikut:

```
area: 12
```

Hal baru dari contoh diatas adalah operator **::** dari lingkup yang disertakan dalam pendefinisian **set_values()**. Digunakan untuk mendeklarasikan anggota dari class diluar class tersebut. Scope operator (::) menspesifikasikan class dimana anggota yang dideklarasikan berada, memberikan scope properties yang sama seperti jika dideklarasikan secara langsung dalam class.

Program 14.2

```
#include <conio.h>
#include <iostream>

using namespace std;

class CRectangle {
    int x, y;
    public:
    void set_values (int,int);
    int area (void) {return (x*y);}
};

void CRectangle::set_values (int a, int b) {
    x = a;
    y = b;
}

int main () {
    CRectangle rect, rectb;
    rect.set_values (3,4);
    rectb.set_values (5,6);
    cout << "rect area: " << rect.area() << endl;
    cout << "rectb area: " << rectb.area() << endl;
    getch();
    return 0;
}
```

Keluaran program diatas adalah sebagai berikut:

```
rect area: 12 rectb area: 30
```

Perhatikan pemanggilan **rect.area()** tidak memberikan hasil yang sama dengan pemanggilan **rectb.area()**. Ini disebabkan karena objek dari class **CRectangle** mempunyai variable sendiri, **x** dan **y**, dan fungsi **set_value()** dan **area()**.

Class terdiri dari dua bagian, yaitu: *Class head* yaitu kata kunci class diikuti oleh nama kelas dan Class body yaitu class yang diapit kurung kurawal { }, diikuti tkita titik koma atau daftar deklarasi. Misalnya:

```
class screen (/*.....*/) ;
class screen (/*.....*/) s1 , s2 ;
```

Dalam class body ditentukan data members, member function dan tingkat-tingkat dari information hidding. Deklarasi untuk data members sama dengan deklarasi untuk variabel, kecuali bahwa inisialisasi eksplisit tidak diperbolehkan.

Perhatikan contoh dibawah ini:

```
class screen {
// height dan width menyatakan jumlah kolom
//crusor menunjukkan posisi screen sekarang
// scr menunjuk ke array height*width

short height, width ;
// menyatakan jumlah kolom
char *crusor ;
// posisi screen saat ini
char scr ;
// menunjuk ke array heightwidth
};
```

Class dapat mendefinisikan data member berupa reference ke tipe dirinya, perhatikan contoh dibawah ini:

```
class linkScreen {
screen window ;
linkScreen *next ;
linkScreen *prev ;
};
```

Member functions dari suatu kelas tidak dideklarasikan di dalam class body. perhatikan contoh dibawah ini:

```
class Screen {

Public :
    void home ( ) { cursor = scr ; }
    void move ( int, int ) ;
    char get ( ) { return *cusor ;}
    char get (int , int ) ;
    void checkRange ( int , int ) ;
    .....
};
```

Member function yang didefinisikan di dalam class-body secara otomatis dianggap inline, dan member function yang lebih dari satu atau dua baris sebaiknya didefinisikan di luar class-body. perhatikan contoh dibawah ini

Program 14.3

```
#include "screen.h"
#include <iostream.h>
#include <stdlib.h>

using namespace std;

void screen :: checkRange (int row , int col ) // validasi koordinat
{
   if (row < 1 || row > height || col < 1 || col > width )
   {
      cerr << "Koordinat screen ( " << row << ", "<< col << " ) keluar batas \n " ; exit (-
1);
   }
}
```

Member function yang didefinisikan di luar class body, jika ingin dibuat inline harus secara eksplisit. perhatikan contoh dibawah ini:

```
inline void screen :: move ( int r , int c )
// memindahkan cursor ke posisi (r,c)

{
    clockRange (r,c) ;
// koordinat sah ?
    int row = (r-1) *width ;
    cursor = ser + row + c-1 ;
}
inline void screen :: move ( int r , int c )
// memindahkan cursor ke posisi (r,c)

{
    clockRange (r,c) ;
// koordinat sah ?
    int row = (r-1) *width ;
    cursor = ser + row + c-1 ;
}
```

Member function dibedakan dari fungsi-fungsi biasa dalam hal-hal:

- Member function punya hak akses penuh ke anggota-anggota kelas yang bersifat public maupun private. Pada umumnya, fungsi-fungsi biasa dapat di akses hanya oleh anggota-anggota kelas yang public. Tentu saja pada umumnya member functions suatu kelas tidak mempunyai hak akses ke anggota-anggota non public dari kelas-kelas lainnya.
- Member function didefinisikan dalam scope dari kelasnya, fungsi-fungsi biasa didefinisikan dalam scope file.

14.4. Struktur dan kelas

Dalam C++, tipe data struktur yang dideklarasikan dengan kata kunci struct, dapat mempunyai komponen dengan sembarang tipe data, baik tipe data dasar maupun tipe data turunan, termasuk fungsi. Dengan kemampuan ini, tipe data struktur menjadi sangat berdaya guna.

Misalnya, pada masalah ketika kita ingin membentuk tipe data struktur yang namanya kotak. Maka dapat dideklarasikan sebagai berikut:

```
struct tkotak
{
      double panjang;
      double lebar;
};
tkotak kotak;
```

Untuk memberi nilai ukuran kotak tersebut, kita dapat menggunakan perintah-perintah ini:

```
kotak.panjang = 10;
kotak.lebar = 7;
```

Untuk memberi nilai panjang dan lebar kotak, salah satu caranya adalah seperti diatas. Cara lain untuk memberi nilai panjang dan lebar adalah dengan membentuk suatu fungsi. Karena fungsi ini hanya digunakan untuk memberi nilai data panjang dan lebar suatu kotak, tentunya fungsi ini khusus milik objek kotak, sehingga harus dianggap sebagai truktur kotak. C++ sebagai bahasa pemrograman dapat mendefinisikan anggota tipe struktur yang berupa fungsi. Dengan menambah fungsi tersebut, maka struktur kotak menjadi lebih jelas bentuknya.

```
struct tkotak
{
      double panjang;
      double lebar;
void SetUkuran(double pj, double lb)
      {
      panjang = pj;
      lebar = lb;
      };
};
tkotak kotak;
```

dengan tipe struktur kotak seperti itu, untuk memberi nilai panjang dan lebar hanya dengan memanggil fungsi SetUkuran()

```
kotak.SetUkuran(10,7);
```

Selain punya ukuran panjang dan lebar, kotak juga mempunyai keliling dan luas. Dengan demikian, kita dapat memasukkan fungsi untuk menghitung keliling dan luas ke dalam struktur kotak. Sebagai catatan, bahwa definisi fungsi yang menjadi anggota struktur dapat ditempatkan di luar tubuh struktur.

Dengan cara ini maka deklarasi struktur kotak menjadi seperti berikut:

```
struct tkotak
{
    double panjang;
    double lebar;
    void SetUkuran(double pj, double lb);
    double Keliling();
    double Luas();
};
tkotak kotak;
```

Class merupakan struktur data dari obyek. Untuk menjelaskan tentang class, lihat perbandingannya dengan struktur. Untuk lebih jelasnya mengenai keduanya perhatikan contoh program di bawah ini:

Program 14.4

```
#include <conio.h>
#include <iostream>
#include <string.h>

using namespace std;

struct siswaSMK
{
        char nis[12];
        char nama[25];
        int umur;
};

void main()
{
        siswaSMK siswakls3;
        strcpy(siswakls3.nis, "95514060");
        strcpy(siswakls3.nama, "Suprapto");
        siswakls3.umur = 20;
        cout << siswakls3.nis << endl;
        cout << siswakls3.nama << endl;
        cout << siswakls3.umur << endl;
        getch();
        return 0;
}
```

Setelah program di atas dicompile, error tidak ada. Berikutnya struktur di atas kita ganti dengan class, sehingga program menjadi seperti berikut dibawah ini:

Program 14.5

```
#include <conio.h>
#include <iostream>
#include <string.h>

using namespace std;

class siswaSMK
```

```
{
        char nis[12]; char nama[25];
        int umur;
};

void main()
{
        siswaSMK siswakls3;
        strcpy(siswakls3.nis, "95514060");
        strcpy(siswakls3.nama, "Suprapto");
        siswakls3.umur = 20;
        cout << siswakls3.nis << endl;
        cout << siswakls3.nama << endl;
        cout << siswakls3.umur << endl;
        getch();
        return 0;
}
```

Setelah program di atas di compile, ternyata error muncul. Error tersebut muncul karena class tidak dikenal dalam main(). Kesalahan ini sekaligus menunjukkan perbedaan dengan struktur. Agar program di atas dapat dicompile, ditambahkan perintah public diikuti dengan tkita titik dua (:), sehingga programnya menjadi:

Program 14.6

```
#include <conio.h>
#include <iostream>
#include <string.h>

using namespace std;

class siswaSMK
{
        char nis[12]; char nama[25];
        int umur;
};

void main()
{
        siswaSMK siswakls3;
        strcpy(siswakls3.nis, "95514060");
        strcpy(siswakls3.nama, " suprapto");
        siswakls3.umur = 20;
```

```
        cout << siswakls3.nis << endl;
        cout << siswakls3.nama << endl;
        cout << siswakls3.umur << endl;
        getch();
        return 0;
}
```

Perintah PUBLIC menyatakan bahwa perintah-perintah yang ada di bawahnya dapat diakses diluar class. Perintah PUBLIC merupakan termasuk a*ccess specifier* (penentu akses). Selain PUBLIC, terdapat perintah lain yang termasuk access specifier, yaitu PRIVATE.

Program 14.7

```
include <iostream>
#include <conio.h>
#include <string.h>

using namespace std;

class siswaSMK
{ private:
        char nis[12];
        char nama[25];
        int umur;
};

int main()
{
        siswaSMK siswakls3;
        strcpy(siswakls3.nis, "95514060");
        strcpy(siswakls3.nama, "suprapto");
        siswakls3.umur = 20;
        cout << siswakls3.nis << endl;
        cout << siswakls3.nama << endl;
        cout << siswakls3.umur << endl;
        getch();
        return 0;
}
```

Setelah dicompile, error yang sama dengan sebelumnya muncul yaitu class tidak dapat diakses di main(). Perintah PRIVATE memberi pengertian bahwa perintah yang ada dibawahnya hanya dapat diakses dalam class tersebut, yang dalam hal ini adalah class siswaSMK. variabel

nis, nama, dan umur dalam class siswaSMK disebut data anggota. Selain data anggota, kita juga dapat menambahkan fungsi anggota.

Program 14.7

```
#include <iostream>
#include <conio.h>
#include <string.h>

using namespace std;

class siswaSMK
{
        private :
        char nis[12];
        char nama[25];
        int umur;

        public :
        void inisialisasi(char *NISSIS, char *NAMASIS, int UMURSIS)
        {
                strcpy(nis, NISSIS);
                strcpy(nama, NAMASIS);
                umur = UMURSIS;
        }
        void tampilkan()
        {
                cout << nis << endl;
                cout << nama << endl;
                cout << umur << endl;
        }
};

void main()
{
 siswaSMK siswakls3;
 siswakls3.inisialisasi("95514060","Suprapto",20);
 siswakls3.tampilkan();
}
```

Pada program di atas, fungsi inisialisasi() dan tampilkan() merupakan fungsi anggota dari class siswaSMK. Keduanya dibuat public karena akan diakses dari luar class, sedangkan data anggotanya (nis,

nama, umur) dibuat private. Mungkin Kita berpikir mengapa program terakhir terlalu panjang, padahal akan diperoleh hasil yang sama dengan program sebelumnya.

Program 14.8

```
#include <iostream.h>
#include <conio.h>
#include <string.h>

class sis
{
        private :
        char nis[12];
        char nama[25];
        int umur;

        public :
        void inisialisasi(char *NISSIS,char *NAMASIS,int UMURSIS);
        void tampilkan();
};

void main()
{
 sis siswakls3;
 siswakls3.inisialisasi("95514060","Suprapto",20);
 siswakls3.tampilkan();
}

void sis::inisialisasi(char *NISSIS, char *NAMASIS, int UMURSIS)
{
        strcpy(nis, NISSIS);
        strcpy(nama, NAMASIS);
        umur = UMURSIS;
}

void sis::tampilkan()
{
        cout << nis << endl;
        cout << nama << endl;
        cout << umur << endl;
}
```

Cara kedua inilah yang sering dipilih oleh para programmer C++. Berikut ini contoh-contoh program yang memanfaatkan class

Program 14.9. Menyimpan data n siswaSMK kemudian menampilkannya.

```
#include <iostream.h>
#include <conio.h>
#include <stdlib.h>

class siswaSMK
{
        public:
        char nis[20];
        char nama[50];
        int umur;

        void tampilkan(char *NISSIS, char *NAMASIS, int UMURSIS)
        {
                cout << "NIS SIS : " << NISSIS << endl;
                cout << "NAMA SIS : " << NAMASIS << endl;
                cout << "UMUR : " << UMURSIS << endl;
        }
};

void main()
{
        siswaSMK siswakls3[50];                 // tipe data array
        char temp[10]; int n, i;

        clrscr();
        cout << "<< ENTRI DATA SISWASMK D3 " << endl;
        cout << endl;
        cout << "Jumlah siswaSMK : ";
        cin.getline(temp, sizeof(temp));
        n = atoi(temp);
        for (i=0;i<=n-1;i++)
        {
                cout << "DATA - " << i+1 << endl;
                cout << "NIS SISWASMK : " ;
                cin.getline(siswakls3[i].nis, sizeof(siswakls3[i].nis));
                cout << "NAMA SISWASMK : " ;
                cin.getline(siswakls3[i].nama, sizeof(siswakls3[i].nama));
                cout << "UMUR : ";
                cin.getline(temp, sizeof(temp));
```

```
                siswakls3[i].umur = atoi(temp);
                cout << endl;
        }

        // tampilkan semua data
        cout << "-------------------------" << endl;
        cout << "DATA YANG MASUK" << endl;
        cout << "-------------------------" << endl;
        for (i=0;i<=n-1;i++)
        {
                cout << "DATA SISWASMK " << i+1 << endl;
                siswakls3[i].tampilkan(siswakls3[i].nis, siswakls3[i].nama,
siswakls3[i].umur);
                cout << endl;
        }
        getch();
}
```

14.5. Constructor dan destructor

Objek biasanya memerlukan inisialisasi variabel atau menentukan memori dinamik selama proses untuk mencapai hasil akhie yang diharapkan dan menghindari pengembalian nilai yang tidak diharapkan. Untuk mengatasinya dapat digunakan fungsi spesial: constructor, yang dapat dideklarasikan dengan pemberian nama fungsi dengan nama yang sama untuk class. Fungsi dari constructor ini akan dipanggil secara otomatis ketika instance baru dari sebuah class dibuat. Pehatikan contoh program dibawah ini:

Program 14.10

```
#include <iostream>

class CRectangle {
int width, height;
public:
CRectangle (int,int);
int area (void) {return (width*height);}
};
CRectangle::CRectangle (int a, int b) {
        width = a;
        height = b;
}
```

```
int main () {
        CRectangle rect (3,4);
        CRectangle rectb (5,6);
        cout << "rect area: " << rect.area() << endl;
        cout << "rectb area: " << rectb.area() << endl;
}
```

Keluaran program diatas adalah sebagai berikut:

```
rect area: 12 rectb area: 30
```

Hasil dari contoh diatas sama seperti contoh sebelumnya. Dalam hal ini, hanya menggantikan fungsi **set_values**, yang sudah tidak ada dengan class constructor. Perhatikan cara parameter diberikan ke constructor pada saat instance class dibuat:

```
CRectangle rect (3,4); CRectangle rectb (5,6);
```

Destructor berfungsi kebalikan dari construktor. Secara otomatis akan dipanggil jika objek di keluarkan dari memory, ataupun karena keberadaannya telah selesai (misalnya: jika didefinisikan sebuah objek local dalam function dan function tersebut selesai) atau karena merupakan objek yang secara dinamis ditetapkan dan dikeluarkan dengan menggunakan operator **delete**. Destuctor harus mempunyai nama yang sama dengan class, diberi awalan tile (~) dan tidak mengembalikan nilai. Untuk lebih jelasnya perhatikan contoh dibawah ini:

Program 14.11

```
#include <iostream.h>

class CRectangle {
int *width, *height;
public:
CRectangle (int,int);
~CRectangle ();
int area (void) {return (*width * *height);}
};
CRectangle::CRectangle (int a, int b) {
width = new int;
height = new int;
*width = a;
*height = b;
}
CRectangle::~CRectangle () {
delete width;
```

```
delete height;
}
int main () {
CRectangle rect (3,4), rectb (5,6);
cout << "rect area: " << rect.area() << endl;
cout << "rectb area: " << rectb.area() << endl;
return 0;
}
```

Keluaran program adalah sebagai berikut :

```
rect area: 12
rectb area: 30
```

14.6. Overloading Constructor

Sama halnya seperti fungsi, constructor juga dapat mempunyai nama yang sama tetapi mempunyai jumlah dan tipe yang berbeda pada parameternya. Pada saat pemanggilan kompiler akan meng-eksekusi yang sesuai pada saat objek class di deklarasikan.

Pada kenyataannya, ketika dideklarasikan sebuah class dan tidak disebutkan constructornya, maka kompiler secara otomatis akan mengasumsikan dua constructor overloaded ("default constructor" dan "copy constructor").

Perhatikan contoh dibawah ini:

```
class CExample {
public:
int a,b,c;
void multiply (int n, int m) { a=n; b=m;
c=a*b; };
};
```

Jika tanpa constructor, Kompiler secara otomatis mengasumsikan anggota-anggota fungsi constructor berikut :

- Empty constructor Merupakan constructor tanpa parameters didefinisikan sebagai nop (blok instruksi kosong). Tidak melakukan apapun.

```
CExample::CExample () { };
```

- Copy constructor Merupakan constructor dengan satu parameter dengan tipe yang sama yang ditetapkan untuk setiap anggota variable class nonstatik objek yang disalin dari objek sebelumnya.

```
CExample::CExample (const CExample&
rv) {
a=rv.a; b=rv.b; c=rv.c;
}
```

Hal yang penting untuk diketahui adalah, bahwa kedua constructor default: empty construction dan copy constructor ada jika tidak ada constructor lain yang dideklarasikan. Jika terdapat constructor dengan sejumlah parameter dideklarasikan, maka tidak satupun dari constructors default ini ada.

Program 14.12

```
#include <iostream.h>
class CRectangle {
int width, height;
public:
CRectangle ();
CRectangle (int,int);
int area (void) {return (width*height);}
};
CRectangle::CRectangle () {
width = 5;
height = 5;
}
CRectangle::CRectangle (int a, int b) {
width = a;
height = b;
}
int main () {
CRectangle rect (3,4);
CRectangle rectb;
cout << "rect area: " << rect.area() << endl;
cout << "rectb area: " << rectb.area() << endl;
}
```

Keluaran program diatas adalah sebagai berikut:

```
rect area: 12 rectb area: 25
```

pada contoh program diatas **rectb** dideklarasikan tanpa parameter, sehingga diinisialisasikan dengan constructor tanpa parameters, yang mendeklarasikan **width** dan **height** dengan nilai **5**. Perhatikan jika dideklarasikan objek baru dan tidak diberikan parameter maka tidak diperlukan tkita kurung **()**:

```
CRectangle rectb; // right
CRectangle rectb(); // wrong!
```

14.7. Menulis Class

Sebelum menulis *class*, langkah pertama pertimbangkan dimana letak atau programer akan menggunakan *class* dan bagaimana *class* tersebut akan digunakan. Pertimbangkan pula nama yang tepat dan tuliskan seluruh informasi atau properti yang ingin kita isi pada *class*. Jangan sampai terlupa untuk menuliskan secara urut *method*

yang akan Kita gunakan dalam *class*. Dalam pendefinisian *class*, dituliskan:

```
<modifier> class <name> {
<attributeDeclaration>*
<constructorDeclaration>*
<methodDeclaration>*
}
```

dimana: <modifier> adalah sebuah *access modifier*, yang dapat dikombinasikan dengan tipe *modifier* lain.

Pada bagian ini, kita akan membuat sebuah *class* yang berisi *record* dari siswa. Jika kita telah mengidentifikasikan tujuan dari pembuatan class, maka dapat dilakukan emberian nama yang sesuai. Nama yang tepat pada *class* ini adalah StudentRecord. Untuk mendefinisikan *class*, kita tuliskan :

```
public class StudentRecord
{
        //area penulisan kode
selanjutnya
}
```

Dimana,

Public	*Class* ini dapat diakses dari luar *package*
Class	*Keyword* yang digunakan untuk pembuatan *class (*Java)
StudentRecord	Identifier yang menjelaskan *class*

Dalam pendeklarasian atribut, kita tuliskan :

```
<modifier> <type> <name> [=
<default_value>];
```

Langkah selanjutnya adalah mengurutkan atribut yang akan diisikan pada *class.* Untuk setiap informasi, urutkan juga tipe data yang yang tepat untuk digunakan. Contohnya, Kita tidak mungkin menginginkan untuk menggunakan tipe data *integer* untuk nama siswa, atau tipe data *string* pada nilai siswa. Berikut ini adalah contoh informasi yang akan diisikan pada *class* StudentRecord :

```
name            - String
address         - String
age             - Int
math grade      - double
english grade   - double
science grade   - double
average grade   - double
```

Kita dapat menambahkan informasi lain jika diperlukan. Jika kita telah menuliskan seluruh atribut yang akan diisikan pada *class*, selanjutnya kita akan menuliskannya pada kode. Jika kita menginginkan bahwa atribut – atribut tersebut adalah unik untuk setiap *object* (dalam hal ini untuk setiap siswa), maka kita harus mendeklarasikannya sebagai *instance variable.* Perhatikan contoh dibawah ini:

```
public class StudentRecord
{
private String name;
private String address;
```

```
private int age;
private double mathGrade;
private double englishGrade;
private double scienceGrade;
private double average;
```

private disini menjelaskan bahwa variabel tersebut hanya dapat diakses oleh class itu sendiri. Object lain tidak dapat menggunakan variabel tersebut secara langsung. Kita akan membahas tentang kemampuan akses pada pembahasan selanjutnya. Disamping instance variable, kita juga dapat mendeklarasikan class variable atau variabel yang dimiliki class sepenuhnya. Nilai pada variabel ini sama pada semua object di class yang sama.

Anggaplah kita menginginkan jumlah dari siswa yang dimiliki dari seluruh class, kita dapat mendeklarasikan satu static variable yang akan menampung nilai tersebut. Kita beri nama variabel tersebut dengan nama studentCount. Berikut penulisan static variable:

```
public class StudentRecord
{

//area deklarasi instance variables
        private static int studentCount;
//area penulisan kode selanjutnya
}
```

Kita gunakan *keyword : 'static'* untuk mendeklarasikan bahwa variabel tersebut adalah *static*. Maka keseluruhan kode yang dibuat terlihat sebagai berikut :

```
public class StudentRecord
{
        private String name;
        private String address;
        private int age;
        private double mathGrade;
        private double englishGrade;
        private double scienceGrade;
        private double average;
        private static int studentCount;
        //area penulisan kode
selanjutnya
}
```

Sebelum kita membahas *method* apa yang akan dipakai pada *class*, mari kita erhatikan penulisan *method* secara umum. Dalam pendeklarasian *method*, kita tuliskan :

```
<modifier> <returnType>
<name>(<parameter>*) {
<statement>*
}
```

dimana,

```
<modifier> dapat menggunakan
beberapa modifier yang berbeda
<returnType> dapat berupa seluruh tipe
data, termasuk void
<name> identifier atas class
<parameter> ::= <tipe_parameter>
<nama_parameter>[,]
```

Untuk mengimplementasikan enkapsulasi, kita tidak menginginkan sembarang object dapat mengakses data kapan saja. Untuk itu, kita deklarasikan atribut dari class sebagai private. Namun, ada kalanya dimana kita menginginkan object lain untuk dapat mengakses data private. Dalam hal ini kita gunakan accessor methods.

Accessor Methods digunakan untuk membaca nilai variabel pada class, baik berupa instance maupun static. Sebuah accessor method umumnya dimulai dengan penulisan **get<namaInstanceVariable>**.
Method ini juga mempunyai sebuah return value. Sebagai contoh, kita ingin menggunakan accessor method untuk dapat membaca nama, alamat, nilai bahasa Inggris, Matematika, dan ilmu pasti dari siswa. Perhatikan salah satu contoh implementasi accessor method.

```
public class StudentRecord
{
        private String name;
                :
                :
        public String getName(){
        return name;
        }
}
```

dimana,

```
public   - Menjelaskan bahwa method tersebut dapat diakses dari object luar class
String   - Tipe data return value dari method tersebut bertipe String
getName - Nama dari method
() - Menjelaskan bahwa method tidak memiliki parameter apapun
```

Pernyataan berikut,

```
return name;
```

dalam program kita menkitakan akan ada pengembalian nilai dari nama instance variable ke pemanggilan method. Perhatikan bahwa return type dari method harus sama dengan tipe data seperti data pada pernyataan return. Kita akan mendapatkan pesan kesalahan sebagai berikut bila tipe data yang digunakan tidak sama :

```
StudentRecord.java:14: incompatible types
found : int
required: java.lang.String
        return age;
        ^
        1 error
```

Contoh lain dari penggunaan accessor method adalah getAverage,

```
public class StudentRecord
{
    private String name;
    :
    :
    public double getAverage(){
        double result = 0;
        result = (mathGrade+englishGrade+scienceGrade)/3;
        return result;
    }
}
```

Method **getAverage()** menghitung rata – rata dari 3 nilai siswa dan menghasilkan nilai return value dengan nama result. Bagaimana jika kita menghendaki object lain untuk mengubah data? Yang dapat kita lakukan adalah membuat method yang dapat memberi atau mengubah nilai variable dalam class, baik itu berupa instance maupun static.

Method semacam ini disebut dengan mutator methods. Sebuah mutator method umumnya tertulis **set<namaInstanceVariabel>**. Mari kita perhatikan salah satu dari implementasi mutator method :

```
public class StudentRecord
{
private String name;
        :
        :
        public void setName( String
temp ){
                name = temp;
        }
}
```

dimana,

public - Menjelaskan bahwa *method* ini dapat dipanggil *object* luar class
void - *Method* ini tidak menghasilkan *return value*
setName - Nama dari *method*
(String temp) - Parameter yang akan digunakan pada *method*

Pernyataan berikut :

```
name = temp;
```

mengidentifikasi nilai dari temp sama dengan name dan mengubah data pada instance variable name. Perlu diingat bahwa mutator methods tidak menghasilkan return value. Namun berisi beberapa argumen dari program yang akan digunakan oleh method. Kita dapat mempunyai banyak return values pada sebuah method selama mereka tidak pada blok program yang sama. Kita juga dapat menggunakan konstanta disamping variabel sebagai return value. Sebagai contoh, perhatikan method berikut ini:

```
public String getNumberInWords( int
num ){
        String defaultNum = "zero";
        if( num == 1 ){
                return "one";
        }
        else if( num == 2){
                return "two";
        }
        return defaultNum;
}
```

Kita menggunakan static method untuk mengakses static variable studentCount.

```
public class StudentRecord
{
        private static int studentCount;
        public static int
getStudentCount(){
        return studentCount;
        }
}
```

dimana,

public - Menjelaskan bahwa *method* ini dapat diakses dari *object* luar class
static - *Method* ini adalah *static* dan pemanggilannya menggunakan [namaclass].[namaMethod]. misalnya:studentRecord.getStudentCount
Int - Tipe *return* dari *method.* Mengindikasikan *method* tersebut harus mempunyai *return value* berupa integer
public - Menjelaskan bahwa *method* ini dapat diakses dari *object* luar class
getStudentCount - Nama dari *method*
() - *Method* ini tidak memiliki parameter apapun

Pada deklarasi di atas, method getStudentCount() akan selalu menghasilkan return value 0 jika kita tidak mengubah apapun pada kode program untuk mengatur nilainya. Kita akan membahas pengubahan nilai dari studentCount pada pembahasan constructor. Berikut ini adalah kode untuk class StudentRecord :

Program 14.13

```
public class StudentRecord
{
        private String name;
        private String address;
        private int age;
        private double mathGrade;
        private double englishGrade;
        private double scienceGrade;
        private double average;
        private static int studentCount;
        /**
        * Menghasilkan nama dari Siswa
        */
        public String getName(){
                return name;
        }
        /**
        * Mengubah nama siswa
        */
        public void setName( String temp ){
                name = temp;
        }
        // area penulisan kode lain
        /**
        * Menghitung rata–rata nilai Matematik, * * Ilmu          Pasti
        */
        public double getAverage(){
                double result = 0;
                result = ( mathGrade+englishGrade+scienceGrade )/3;
                return result;
        }
        /**
        * Menghasilkan jumlah instance StudentRecord
        */
```

```
        public static int getStudentCount(){
                return studentCount;
        }
}
```

14.8. Reference *this*

Reference *this* digunakan untuk mengakses instance variable yang dibiaskan oleh parameter. Untuk pemahaman lebih lanjut, mari kita perhatikan contoh pada m*ethod* setAge. Dimisalkan kita mempunyai kode deklarasi berikut pada *method* setAge.

```
public void setAge( int age ){
        age = age; //SALAH!!!
}
```

Nama parameter pada deklarasi ini adalah age, yang memiliki penamaan yang sama dengan *instance variable* age. Parameter age adalah deklarasi terdekat dari *method*, sehingga nilai dari parameter tersebut akan digunakan. Maka pada pernyataan :

```
age = age;
```

kita telah menentukan nilai dari parameter age kepada parameter itu sendiri. Hal ini sangat tidak kita kehendaki pada kode program kita. Untuk menghindari kesalahan semacam ini, kita gunakan metode referensi ***this***. Untuk menggunakan tipe referensi ini, kita tuliskan :

```
this.<namaInstanceVariable>
```

Sebagai contoh, kita dapat menulis ulang kode hingga tampak sebagai berikut:

```
public void setAge( int age ){
        this.age = age;
}
```

Method ini akan mereferensikan nilai dari parameter age kepada instance variable dari object StudentRecord.

14.9. Overloading Method

Dalam class yang kita buat, kadangkala kita menginginkan untuk membuat method dengan nama yang sama namun mempunyai fungsi yang berbeda menurut parameter yang digunakan. Kemampuan ini dimungkinkan dalam pemrograman Java, dan dikenal sebagai overloading method. Overloading method mengijinkan sebuah method dengan nama yang sama namun memiliki parameter yang berbeda sehingga mempunyai implementasi dan return value yang berbeda pula.

Daripada memberikan nama yang berbeda pada setiap pembuatan method, overloading method dapat digunakan pada operasi yang sama namun berbeda dalam implementasinya. Sebagai

contoh, pada class StudentRecord kita menginginkan sebuah method yang akan menampilkan informasi tentang siswa. Namun kita juga menginginkan operasi penampilan data tersebut menghasilkan output yang berbeda menurut parameter yang digunakan. Jika pada saat kita memberikan sebuah parameter berupa string, hasil yang ditampilkan adalah nama, alamat dan umur dari siswa, sedang pada saat kita memberikan 3 nilai dengan tipe double, kita menginginkan method tersebut untuk menampilkan nama dan nilai dari siswa. Untuk mendapatkan hasil yang sesuai, kita gunakan overloading method di dalam deklarasi class StudentRecord.

Program 14.14

```
public void print( String temp ){
      System.out.println("Name:" + name);
      System.out.println("Address:" + address);
      System.out.println("Age:" + age);
      }
      public void print(double eGrade, double mGrade, double sGrade)
      System.out.println("Name:" + name);
      System.out.println("Math Grade:" + mGrade);
      System.out.println("English Grade:" + eGrade);
      System.out.println("Science Grade:" + sGrade);
}
```

Jika kita panggil pada *method* utama *(main)* :

Program 14.15

```
public static void main( String[] args )
{
      StudentRecord annaRecord = new StudentRecord();
      annaRecord.setName("Anna");
      annaRecord.setAddress("Philippines");
      annaRecord.setAge(15);
      annaRecord.setMathGrade(80);
      annaRecord.setEnglishGrade(95.5);
      annaRecord.setScienceGrade(100);
      //overloaded methods
      annaRecord.print( annaRecord.getName() );
      annaRecord.print( annaRecord.getEnglishGrade(),
      annaRecord.getMathGrade(),
      annaRecord.getScienceGrade());
}
```

Kita akan mendapatkan *output* pada panggilan pertama sebagai berikut :

```
Name:Anna
Address:Philippines
Age:15
```

Kemudian akan dihasilkan *output* sebagai berikut pada panggilan kedua :

```
Name:Anna
Math Grade:80.0
English Grade:95.5
Science Grade:100.0
```

Jangan lupakan bahwa *overloaded method* memiliki *property* sebagai berikut :

1. Nama yang sama
2. Parameter yang berbeda
3. Nilai kembalian (*return*) bisa sama ataupun berbeda

Telah tersirat pada pembahasan sebelumnya, Constructor sangatlah penting pada pembentukan sebuah *object*. Constructor adalah *method* dimana seluruh inisialisasi *object* ditempatkan. Berikut ini adalah *property* dari Constructor :

1. Constructor memiliki nama yang sama dengan *class*
2. Sebuah Constructor mirip dengan *method* pada umumnya, namun hanya informasi - informasi berikut yang dapat ditempatkan pada *header* sebuah constructor, *scope* atau identifikasi pengaksesan (misal: public), nama dari konstuktor dan parameter.
3. Constructor tidak memiliki *return value*
4. Constructor tidak dapat dipanggil secara langsung, namun harus dipanggil dengan menggunakan operator ***new*** pada pembentukan sebuah *class*.

Untuk mendeklarasikan constructor, kita tulis,

```
<modifier> <className> (<parameter>*)
{
<statement>*
}
```

Setiap class memiliki default constructor. Sebuah default constructor adalah constructor yang tidak memiliki parameter apapun. Jika sebuah *class* tidak memiliki constructor apapun, maka sebuah default constructor akan dibentuk secara implisit: Sebagai contoh, pada *class* StudentRecord, bentuk default constructor akan terlihat seperti dibawah ini :

```
public StudentRecord()
{
        //area penulisan kode
}
```

Seperti telah kita bahas sebelumnya, sebuah constructor juga dapat dibentuk menjadi ***overloaded***. Dapat dilihat pada contoh sebagai berikut :

Program 14.16

```
public StudentRecord(){
```

```
//area inisialisasi kode;
}
        public StudentRecord(String temp){
        this.name = temp;
}
        public StudentRecord(String name, String address){
        this.name = name;
        this.address = address;
}
public StudentRecord(double mGrade, double eGrade, double sGrade){
        mathGrade = mGrade;
        englishGrade = eGrade;
        scienceGrade = sGrade;
}
```

Untuk menggunakan constructor, kita gunakan kode – kode sebagai berikut :

```
public static void main( String[] args )
{
        //membuat 3 objek
        StudentRecord annaRecord=new StudentRecord("Anna");
        StudentRecord beahRecord=new StudentRecord("Beah","Philippines");
        StudentRecord crisRecord=new StudentRecord(80,90,100);
        //area penulisan kode selanjtunya
}
```

Sebelum kita lanjutkan, mari kita perhatikan kembali deklarasi variabel static studentCount yang telah dibuat sebelumnya. Tujuan deklarasi studentCount adalah untuk menghitung jumlah *object* yang dibentuk pada *class* StudentRecord. Jadi, apa yang akan kita lakukan selanjutnya adalah menambahkan nilai dari studentCount setiap kali setiap pembentukan *object* pada *class* StudentRecord. Lokasi yang tepat untuk memodifikasi dan menambahkan nilai studentCount terletak pada constructor-nya, karena selalu dipanggil setiap kali objek terbentuk. Perhatikan contoh sebagai berikut:

Program 14.17

```
public StudentRecord(){
        //letak kode inisialisasi
        studentCount++; //menambah student
}
public StudentRecord(String temp){
        this.name = temp;
        studentCount++; //menambah student
```

```
}
public StudentRecord(String name, String address){
        this.name = name;
        this.address = address;
        studentCount++; //menambah student
}
public StudentRecord(double mGrade, double eGrade, double sGrade){
        mathGrade = mGrade;
        englishGrade = eGrade;
        scienceGrade = sGrade;
        studentCount++; //menambah student
}
```

Pemanggilan constructor dapat dilakukan secara berangkai, dalam arti Kita dapat memanggil constructor di dalam constructor lain. Pemanggilan dapat dilakukan dengan referensi ***this()***. Perhatikan contoh kode sebagai berikut :

Program 14.18

```
public StudentRecord(){
this("some string");

}

public StudentRecord(String temp){
this.name = temp;
}

public static void main( String[] args )
{

StudentRecord annaRecord = new StudentRecord();
}
```

Dari contoh kode diatas, pada saat baris ke 13 dipanggil akan memanggil constructor dasar pada baris pertama. Pada saat baris kedua dijalankan, baris tersebut akan menjalankan constructor yang memiliki parameter String pada baris ke-6. Beberapa hal yang patut diperhatikan pada penggunaan *this()*:

1. Harus dituliskan pada baris pertama pada sebuah constructor
2. Hanya dapat digunakan pada satu definisi constructor. Kemudian metode ini dapat diikuti dengan kode – kode berikutnya yang relevan

14.10. Access Modifier

Pada saat membuat, mengatur *properties* dan *class methods*, kita ingin untuk mengimplementasikan beberapa macam larangan untuk mengakses data. Sebagai contoh, jika Kita ingin beberapa atribut hanya dapat diubah hanya dengan *method* tertentu, tentu Kita ingin menyembunyikannya dari *object* lain pada *class*. Di JAVA, implementasi tersebut disebut dengan ***access modifiers.*** Terdapat 4 macam *access modifiers*, yaitu: *public, private, protected* dan *default*. 3 tipe akses pertama tertulis secara ekplisit pada kode untuk mengindikasikan tipe akses, sedangkan yang keempat yang merupakan tipe *default*, tidak diperlukan penulisan *keyword* atas tipe.

14.10.1. Akses Default (Package Accessibility)

Tipe ini mensyaratkan bahwa hanya *class* dalam *package* yang sama yang memiliki hak akses terhadap variabel dan *methods* dalam *class*. Tidak terdapat *keyword* pada tipe ini.Sebagai contoh :

```
public class StudentRecord
{
        //akses dasar terhadap variabel
        int name;
        //akses dasar terhadap metode
        String getName(){
                return name;
        }
}
```

Pada contoh diatas, variabel nama dan *method* getName() dapat diakses dari *object* lain selama *object* tersebut berada pada *package* yang sama dengan letak dari file StudentRecord.

14.10.2. Akses Public

Tipe ini mengijinkan seluruh *class member* untuk diakses baik dari dalam dan luar *class*. *Object* apapun yang memiliki interaksi pada *class* memiliki akses penuh terhadap *member* dari tipe ini. Sebagai contoh:

```
public class StudentRecord
{
        //akses dasar terhadap variabel
        public int name;
        //akses dasar terhadap metode
        public String getName(){
                return name;
        }
}
```

Dalam contoh ini, variabel name dan *method* getName() dapat diakses dari *object* lain.

14.10.3. Akses Protected

Tipe ini hanya mengijinkan *class member* untuk diakses oleh *method* dalam *class* tersebut dan elemen – elemen *subclass*. Sebagai contoh :

```
public class StudentRecord
{
        //akses pada variabel
        protected int name;
        //akses pada metode
        protected String getName(){
                return name;
        }
}
```

Pada contoh diatas, variabel name dan *method* getName() hanya dapat diakses oleh *method* internal *class* dan *subclass* dari *class* StudentRecord. Definisi *subclass* akan dibahas pada bab selanjutnya.

14.10.4. Akses Private

Tipe ini mengijinkan pengaksesan *class* hanya dapat diakses oleh *class* dimana tipe ini dibuat. Sebagai contoh :

```
public class StudentRecord
{
        //akses dasar terhadap variabel
        private int name;
        //akses dasar terhadap metode
        private String getName(){
                return name;
        }
}
```

Pada contoh diatas, variabel name dan *method* getName() hanya dapat diakses oleh
method internal *class* tersebut.

14.11. Contoh Program Class

Kelas merupakan struktur dari obyek. Deklarasi class sebagai berikut:

```
class  nama_class
{   private;
        ------------
        ------------
     protected:
     ------------
     ------------
   public:
    void  nama_fungsi();
        ------------
};
```

menciptakan variable obyek

```
 void main()

 {  nama_class  variable_obyek;
        }
     void nama_class::nama_fungsi()
 {  definisi fungsi }
```

private ; data/fungsi yang tidak dapat diakses diluar kelas. protected: data/fungsi yang dapat diakses pada kelas tersebut dan friend class. public: data/fungsi yang dapat diakses diluar kelas.

Program 14.19

```
#include<iostream>
#include<conio.h>

using namespace std;

class Pegawai
        { public:
        char nip[35];
        char nama[25];
```

```
        int umur;
        };

void main()
{
        Pegawai produksi,akuntansi;
        strcpy(produksi.nip, "200322");
        trcpy(produksi.nama, "Wijaya");
        produksi.umur= 35;
        cout<<"================================="<<endl;
         cout<<"NIP      = ", cout<<produksi.nip<<endl;
         cout<<"NAMA  = ", cout<<produksi.nama<<endl;
         cout<<"UMUR  = ", cout<<produksi.umur<<endl;
         cout<<"================================="<<endl;
        akuntansi=produksi;
        cout<<"NIP      = ", cout<<akuntansi.nip<<endl;
        cout<<"NAMA  = ", cout<<akuntansi.nama<<endl;
         cout<<"UMUR  = ", cout<<akuntansi.umur<<endl;
         cout<<"================================="<<endl;
}
```

Program 14.20

```
#include<iostream>
#include<conio.h>

using namespace std;

class Pegawai
{ public:
                char nip[6];
                char nama[25];
                int umur;

                void isidata(char*nip, char *nama, int umur)
                { strcpy(nip, nip);
                  strcpy(nama,nama);
                  umur= umur;  }

                void tampil_biodata();
                {
                  cout<<endl;
                  cout<<"============================"<<endl;
```

```
                cout<< "NIP             : "<<nip<<endl;
                cout<< "NAMA            : "<<nama<<endl;
                cout<< "UMUR            : "<<umur<<endl;
                cout<<"============================"<<endl; }
        };

 void main()
 {
                Pegawai Akuntansi
                cout<<"INPUTKAN DATA KARYAWAN"<<endl;
                cout<<"NIP(6)  = ", cin>>Akuntansi.nip;
                cout<<"NAMA(20)= ", cin>>Akuntansi.nama;
                cout<<"UMUR(2) = ", cin>>Akuntansi.umur;
                Akuntansi.isidata(Akuntansi.nip,Akuntansi.nama,Akuntansi.umur);
                Akuntansi.tampil_biodata();
 }
```

Program 14.21

```
#include<iostream>
#include<conio.h>

using namespace std;

class contoh
{  public:
                int x;
                int y;
                int tambah()
                {
                        cout<< ( x + y)<<endl;;
                        return 0; }
                        int kali()
                {
                cout<< ( x * y)<<endl;
                }
};

void main()
{
        contoh hitung;
        cout<<" Masukan x ? ", cin>>hitung.x;
        cout<<" Masukan y ? ", cin>>hitung.y;
        cout<<" Hasil x + y = ",
```

```
        tambah();
        cout<<" Hasil x * y = ",
        hitung.kali();
  getch()
}
```

14.12. Soal Latihan

Jawablah soal latihan dibawah ini dengan baik dan benar.

1. Apakah fungsi tipe kelas dalam bahasa C
2. Sebutkan tujuan dari mendeklarasikan class
3. Sebutkan yang digunakan untuk menentukan batasan akses terhadap member
4. Apa yang dimaksud dengan konstruktor dan destruktor
5. Jelaskan apa yang dimaksud dengan overloading konstruktor
6. Jelaskan bagaimana cara menulis class
7. Apa yang dimaksud dengan overloading method
8. Apa yang dimakksud dengan akses public, private dan protected

BAB 15
PEMROGRAMAN BERORIENTASI OBJEK

15.1. Pemrograman Object-Oriented dan Prosedural

Dalam beberapa forum diskusi yang membahas programming, salah satu thread yang selalu hangat dan menarik di bicarakan adalah thread mengenai Object oriented Programming (OOP). OOP itu sendiri sebetulnya adalah konsep lama, tapi topik ini seperti Design pattern, OOAD dan lain sebagainya sepertinya baru hangat dibicarakan di forum diskusi beberapa tahun belakangan ini. Entah mengapa, mungkin karena bersamaan dengan bermunculan framework yang banyak diantaranya menganut konsep OOP, C++ yangg sekarang sudah full-blown OOP, framework MS .NET yang juga sudah full-blown OOP, dan banyak lagi. Akhirnya banyak orang-orang di sini yang mulai bertanya-tanya "apa sih OOP tuh?", "apa bedanya dengan prosedural?", "apa untungnya kalau memakai OOP?.

Sekarang kita tidak perlu membicarakan "Apa itu OOP" dan bagaimana membuat / belajar OOP. kita hanya akan memberi contoh konkrit perbedaan dari OOP dan Procedural Code, dan kemudian mengambil kesimpulan tentang keuntungan apa yang bisa kita dapat dari OOP. Dalam buku ini banyak dijelaskan apa itu OOP, penggunaan Design Patternnya dan bagaimana mengimplementasikannya dalam kode C++.

15.2. Perbedaan Object-Oriented dan Prosedural

Bagaimana desain object oriented berbeda dengan kode prosedural yang lebih tradisional? Akan sangat mudah untuk mengatakan bahwa perbedaan utamanya adalah kode berorientasi obyek memiliki object di dalamnya. Ini tidak betul. Dalam PHP kita akan sering menemukan kode prosedural menggunakan obyek. Kita juga mungkin sering melihat class-class yang mengandung kode prosedural di dalamnya. Keberadaan class tidak menjamin adanya desain berorientasi obyek, walau dalam bahasa pemrogramman Java sekalipun, yang notabene "memaksa" kita untuk melakukan apa pun di dalam class.

Satu perbedaan utama antara kode berorientasi obyek dengan kode prosedural dapat dilihat dari bagaimana tanggung jawab (responsibility) didistribusikan. Kode prosedural akan berupa urut-urutan perintah dan method calls. Kode pemanggil (controlling code) cenderung bertanggung jawab untuk menghandle pengecekan kondisi (if, then, else dst). Kontrol top-down ini – dalam pengembangan aplikasi– dapat menghasilkan duplikasi kode dan saling ketergantungan lintas (antar) proyek. Kode berorientasi obyek mencoba untuk meminimalisasi ketergantungan ini dengan cara memindahakan responsibility untuk meng-handle tugas-tugas dari client code ke dalam obyek-obyek di dalam system.

15.3. Pemrograman berorientasi objek

Objek adalah kumpulan variabel dan method (fungsi) yang saling berkaitan. Mungkin kita pernah menggunakan struct dalam C atau record dalam Pascal/Delphi? Dalam struct dan record, kita bisa mengumpulkan tipe-tipe variabel yang berlainan. Nah, objek adalah perkembangan dari struct. Dalam objek, kita bisa mengumpulkan variabel-2 dengan tipe yang boleh berbeda-2 dan fungsi-2 yang disebut method.

Objek sebenarnya meniru konsep alam. Ilustrasinya seperti ini. Sebutlah suatu objek, misalnya anjing. Anjing memiliki kondisi: warna, tinggi, lapar, sakit, dll. Selain itu anjing juga memiliki hal-hal yang biasa dilakukan, seperti: menggonggong, menggigit, goyang ekor, dan lain-lain. Jadi kondisi pada alam (misal anjing) ditiru dengan Variabel sedangkan hal-hal yang biasa dilakukan ditiru dengan Method. penggabungan antara Variabel dan Method itulah yang disebut Enkapsulasi.

Hal penting berikutnya dalam OOP adalah Message. Antara objek satu dengan objek yang lain berkomunikasi melalui Message. Ketika objek A ingin objek B melakukan sesuatu, maka objek A mengirim Message ke objek B.

Contoh: Misalnya objek A adalah ICT. Objek B adalah Sepeda.

Jika ICT ingin memindahkan gigi sepedanya ke gigi 5, maka ICT akan mengirim Message ke Sepeda

diketahui sepeda memiliki kemampuan/Method pindahGigi) berbunyi pindahGigi(5). Lengkapnya dapat ditulis: Sepeda.pindahGigi(5). Message itu akan dilaksanakan oleh Sepeda. Kemampuan pindahGigi memerlukan informasi gigi yang diinginkan, dalam hal ini 5. Maka sekarang Variabel (nantinya kita sebut atribut) gigi bernilai 5. Bicara tentang Class, kita bayangkan dalam contoh sebelumnya bahwa Sepeda milik ICT adalah sepeda gunung (kita sebut SepedaGunung). Tentunya selain SepedaGunung terdapat sepeda lain seperti SepedaOnta, SepedaTandem, dll. Tapi yang jelas, jenis-jenis tadi termasuk Sepeda.

Jadi Sepeda Gunung adalah Instance dari Sepeda. SepedaGunung memiliki atribut-atribut(seperti gigi, roda, panjang, pedal, dan lain-lain) juga kemampuan(seperti pindahGigi(), belok(), dll) Sepeda. Ada perbedaan antara Object dan Class.

Class adalah blue print atau rancangan dari Object. Jadi dengan satu Class, kita dapat membuat berapapun Object. Inheritance adalah pewarisan sifat. Sebuah Subclass akan mewarisi sifat dari Superclass-nya. Dalam contoh di atas, Sepeda Gunung akan memiliki atribut gigi, roda dan panjang dan juga kemampuannya yaitu pindah Gigi dan belok. Tetapi Subclass tidak harus hanya memiliki sifat dan kemampuan yang sama persis dengan Superclassnya, tetapi biasanya ditambahkan sifat dan kemampuan lain yang membedakannya dengan Subclass lain dengan Superclass yang sama. Itulah yang membuat SepedaGunung dan SepedaOnta berbeda.

Subclass juga dapat melakukan Override terhadap sifat dan kemampuan warisan Superclassnya. Misalnya yang dilakukan oleh SepedaTandem. SepedaTandem memiliki lebih dari dua roda dan juga pedal sehingga perlu memiliki kemampuan untuk mengaturnya. Sehingga suatu kemampuan warisan dari class Sepeda dapat diganti atau ditambahkan.

Interface membuat satu class dapat berinteraksi dengan class lain yang benar-benar berbeda. Misalnya dalam Show Room yang menjual Mobil, Motor juga Sepeda. Tentu saja ManajerShowRoom harus bisa mengatur sepeda-sepeda bersama dengan Mobil-2 dan Motor-2. Masalahnya adalah perbedaan class yang sangat mencolok. Interface mengatasinya dengan cara:

- Mencari persamaan antara class-2 yang berbeda tanpa memaksa class tersebut saling berhubungan.
- Membuat method yang ingin diterapkan pada class-class.

Istilah-istilah yang digunakan dalam pemrograman berorientasi objek

- Class adalah blueprint atau prototype dari object-objec tertentu yang memiliki kesamaan variable dan method.
- Object atau insance of class merupakan software yang mengemas variable-variable dan methodmethod menjadi satu kesatuan.
- Attribute adalah suatu bentuk karakteristik atau status dari suatu object.
- Method atau behavior dari object adalah action yang bisa di kerjakan oleh suatu object

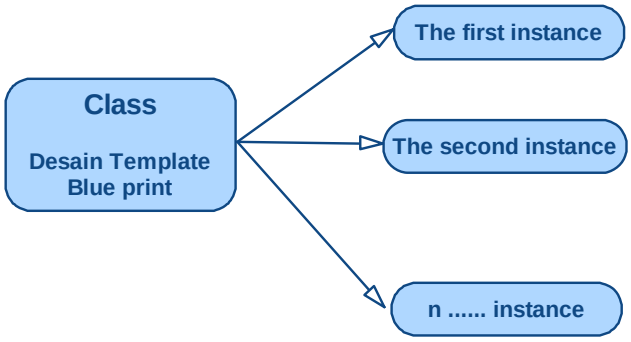
- Constructor merupakan method khusus yang berfungsi untuk inisialisi atau menciptakan suatu object dari class

Pemrograman Berorientasi Objek sebenarnya bukanlah bahasa pemrograman baru, tetapi jalan baru untuk berpikir dan merancang aplikasi yang dapat membantu memecahkan suatu persoalan mengenai pengembangan perangkat lunak. Pemrograman berorientasi objek disusun dan dipahami oleh ilmuwan yang memkitang dunia sebagai populasi objek yang berinteraksi antara satu yang satu dengan objek yang lain.

Prosedur yang digunakan dalam objek dipkitang sebagai kepentingan kedua karena tergantung pada objek itu sendiri. Tentunya hal tersebut berbeda dengan pemrograman terstruktur. Pemrograman terstruktur mempunyai sifat pemisahan data dengan kode yang mengolahnya. Pemrograman Berorientasi Objek (PBO) adalah metode pemrograman yang meniru cara kita memperlakukan sesuatu (benda).

Dalam pemrograman berorientasi obyek, class merupakan sebuah design, template, atau blue-print. Dari suatu class dapat dibuat banyak objek/instance. Analoginya adalah seperti ini, bayangkan jika kita seorang arsitek, sebelum membangun rumah tentunya akan merancang dulu di kertas, jumlah dan letak pintu, jendela, dan semuanya, itulah class. Kemudian dari gambar rumah tersebut kita dapat membangun banyak rumah sesuai rancangan yang sudah dibuat.

Masing-masing rumah jadi tersebut adalah yang disebut dengan objek/instance.



Gambar 15.1. Class Dalam Pemrograman Berorientasi Obyek

Ada empat ciri-ciri atau karakteristik bahasa Pemrograman Berorientasi Objek, antara lain:

15.3.1. Abstraksi

Untuk memproses sesuatu dari dunia nyata dengan menggunakan komputer kita harus menentukan ciri-ciri / sifat-sifat yang penting yang perlu direpresentasikan ke dalam sistem komputer. Ciri-ciri yang dipilih bergantung dari apa yang akan kita lakukan di dalam sistem tersebut dan tergantung dari sudut pkitang yang melihatnya. Misalnya pada sebuah

kasus untuk menentukan attribute suatu class mobil pada :

Tabel 15.1. Attribute suatu class mobil

Kantor Polisi	**Bengkel Mobil**
No mesin	No registrasi Pemilik
Pemilik	Waktu Ganti Oli
Besar Pajak	Deskripsi servis

15.3.2. Enkapsulasi

Penggabungan data dan method ke dalam suatu class atau dengan kata lain mengkombinasikan sebuah struktur dengan fungsi yang memanipulasinya untuk membentuk tipe data baru yaitu kelas (class).

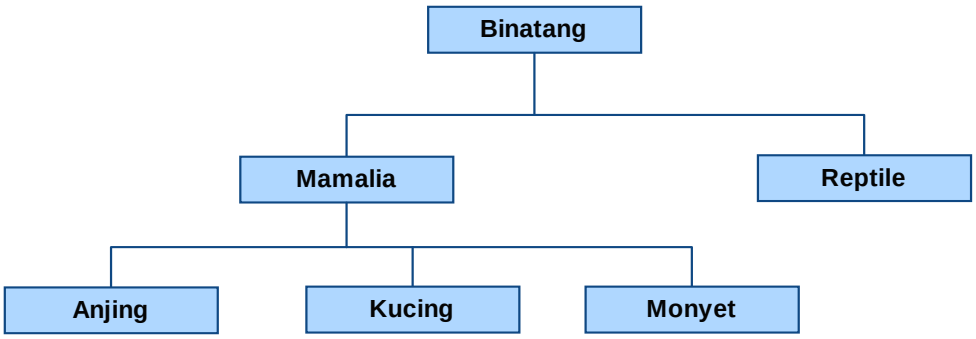
Enkapsulasi merupakan suatu mekanisme untuk menyembunyikan atau memproteksi suatu proses dari kemungkinan interferensi atau penyalahgunaan dari luar sistem dan sekaligus menyederhanakan penggunaan sistem itu sendiri. Akses ke internal sistem diatur sedemikian rupa melalui seperangkat interface.

Contoh sebuah kasus yang ada dilapangan misalnya pada sebuah sepeda motor, dimana pada sistem pemindahan gigi transmisi, maka pengendara tidak perlu tahu detail dari bagaimana proses pemindahan gigi itu dilakukan oleh mesin, cukup tahu bagaimana cara menekan gigi transmisi itu. Pedal gigi transmisi yang diinjak pengendara itu merupakan interface (antar muka) pengendara dengan sistem transmisi sepeda motor.

15.3.3. Pewarisan

warisan sifat dari class induk kepada class anaknya atau mendefinisikan suatu kelas dan kemudian menggunakannya untuk membangun hirarki kelas turunan, yang mana masing-masing turunan mewarisi semua akses kode maupun data kelas dasarnya.

Sebagai manusia kita sebenarnya terbiasa untuk melihat objek yang berada disekitar kita tersusun secara hierarki berdasarkan classnya masing-masing. Dari sini kemudian timbul suatu konsep tentang pewarisan yang merupakan suatu proses dimana suatu class diturunkan dari class lainnya sehingga ia mendapatkkan ciri atau sifat dari class tersebut. Perhatikan contoh hirarki berikut ini:



Gambar 15.2. Hirarki Pewarisan

Dari hirarki diatas dapat dilihat bahwa, semakin kebawah, class akan semakin bersifat spesifik. Class mamalia memiliki seluruh sifat yang dimiliki oleh binatang, demikian halnya juga Anjing, kucing dan Monyet memiliki seluruh sifat yang diturunkan dari class mamalia. Dengan konsep ini, karakteristik yang dimiliki oleh class binatang cukup didefinisikan didefinisikan dalam class binatang saja.

Class mamalia tidak perlu mendefinisikan ulang apa yang telah dimiliki oleh class binatang, karena sebagai class turunannya, ia akan mendapatkan karakteristik dari class binatang secara otomatis. Demikian juga dengan class anjing, kucing dan monyet, hanya perlu mendefinisikan karakteristik yang spesifik dimiliki oleh classnya masing-masing. Dengan memanfaatkan konsep pewarisan ini dalam pemrograman, maka hanya perlu mendefinisikan karakteristik yang lebih umum akan didapatkan dari class darimana ia diturunkan.

15.4. Immutable obyek

Dalam pemrograman berorientasi objek, immutable obyek adalah obyek dengan kondisi yang tidak dapat dimodifikasi setelah dibuat. Hal ini berbeda dengan mutable objek, dimana obyek tersebut dapat dimodifikasi setelah dibuat. Obyek dapat immutable secara keseluruhan atau hanya beberapa atribut objek yang ditentukan saja, misalnya menggunakan atribut anggota const data pada bahasa pemrograman C++.

Dalam beberapa kasus, obyek dianggap immutable walaupun ada beberapa hal perubahan internal yang digunakan oleh atribut tetapi kondisi objek nampaknya tidak bisa diubah secara ekternal dari beberapa sudut pkitang. Sebagai contoh, sebuah obyek yang menggunakan memoization pada cache hasil komputasi yang baik ini tetap dapat tetap dianggap sebagai objek. Awal dari sebuah kondisi tetap objek biasanya diatur pada posisi berdiri, tetapi dapat juga ditetapkan sebelum menggunakan obyek tersebut.

Immutable objek sering berguna karena beberapa biaya operasional seperti menyalin dan membandingkan dapat diabaikan, menyederhanakan kode program dan mempercepat eksekusi. Namun, membuat obyek immutable biasanya tidak tepat jika objek berisi sejumlah besar data yang berubah-ubah. Karena itu, banyak bahasa yang mempunyai fasilita kedua objek tersebut baik mutable dan immutable obyek.

Dalam kebanyakan bahasa berorientasi objek, objek dapat disebut dengan referensi. Beberapa contoh seperti bahasa Java, C++, C#, VB.NET dan banyak bahasa scrip seperti Ruby dan Python. Dalam hal ini, hal-hal apakah kondisi obyek dapat berbeda bila benda dibagi melalui referensi.

Jika obyek yang diketahui adalah immutable, maka dapat disalin cukup dengan membuat salinan referensi, bukan untuk menyalin seluruh objek. Karena referensi (biasanya hanya ukuran sebuah pointer) biasanya jauh lebih kecil daripada obyek itu sendiri, sehingga lebih mudah dalam penyimpanan di memori dan meningkatkan kecepatan dalam eksekusi.

Teknik penyalinan Referensi jauh lebih sulit digunakan pada objek mutable, karena jika ada pengguna referensi ke sebuah obyek mutable yang merubahnya, semua pengguna lain akan melihat perubahannya. Jika ini bukan efek dimaksudkan, maka akan lebih sulit untuk memberitahukan ke pengguna yang lain untuk meminta mereka merespon dengan benar.

Dalam situasi ini, defensif menyalin dari seluruh obyek daripada referensi biasanya lebih mudah namun solusi mahal. Para pengamat menyatakan bahwa teknik alternatif untuk menangani perubahan objek mutabel. Immutable objects dapat berguna dalam aplikasi multi-threaded. Multiple threads dapat bertindak berdasarkan data yang diwakili oleh immutable objek tanpa memperhatikan data yang diubah oleh thread lainnya. immutable obyek mempunyai pertimbangan atau dianggap lebih aman daripada mutable object atau obyek yang berubah.

Dalam Praktiknya selalu menggunakan referensi pada saat melakukan copy atau sering dikenal sebagai objek interning. Jika interning digunakan, dua benda dianggap sama jika dan hanya jika mereka referensi, biasanya digambarkan sebagai integers yang keuanya sama. Beberapa bahasa secara otomatis: misalnya, secara otomatis Python interns string dan lain-lain.

Jika algoritma yang menerapkan interning dijamin, maka untuk melakukan setiap kasus yang mungkin membandingkan dengan obyek untuk mengurangi pointer, dan yang penting dalam mendapatkan kecepatan maksimal dalam aplikasi. (Bahkan jika algoritma tidak dijamin secara lengkap, masih ada kemungkinan jalan yang cepat untuk meningkatan kasus ketika obyek yang sama dan menggunakan referensi yang sama.) Interning umumnya hanya berguna untuk immutable benda.

Kadang-kadang kalau bicara tertentu mengenai satu bidang obyek yang immutable. Ini berarti bahwa tidak ada cara untuk mengubah kondisi bagian objek, walaupun

bagian lain dari objek mungkin berubah-ubah. Hal ini dapat, misalnya, untuk membantu menegakkan tertentu secara eksplisit invariants tentang beberapa data dalam obyek yang sama tinggal melalui masa objek. Dalam beberapa bahasa, hal ini dilakukan dengan kata kunci (misalnya "const" di C++, "final" di Java) yang mana dirancang pada field dari obyek yang immutable.

Keimmutablean tidak menyiratkan bahwa sebagai objek yang disimpan dalam memori komputer *unwriteable*. Sebaliknya, keimmutablean adalah kompilasi waktu membangun dan menunjukkan apa yang harus dilakukan oleh programmer.

Dalam bahasa C++ implementasi pada sebuah cart akan memungkinkan pengguna menyatakan sebuah kasus kelas baru sebagai salah satu const (immutable) atau mutable, seperti yang dikehendaki, dengan menyediakan dua versi yang berbeda dari getItems () method. (Perhatikan bahwa dalam C + + itu tidak diperlukan dan bahkan mustahil - untuk menyediakan constructor khusus untuk kasus const.) Template <typename T>. perhatikan potongan program dibawah ini:

Program 15.1

```
class Cart {
 private:
      std::vector<T> items;

 public:
      Cart(const std::vector<T>& v): items(v) { }

      std::vector<T>& getItems() { return items; }
      const std::vector<T>& getItems()
      const { return items; }
      int total()
      const { /* return sum of the prices */ }
};
```

15.5. Modularitas dan Abstraksi Data

Modularitas dan abstraksi data dapat didefinisikan

- **Modul** atau sebuah unit dari organisasi sistem perangkat lunak yang membundel sekumpulan entitas (data dan metoda operasi) dan dengan secara teliti mengontrol apa "user eksternal" bisa lihat atau gunakan.
- **Penyembunyian informasi** yang dilakukan modul untuk melindungi informasi dari "user external" dalam penggunaannya (misalnya software error, security) dan merupakan basis dari *abstract data type* (ragam data abstraks).
- ***Abstract data type*** **(ADT)** atau Kumpulan obyek dan metoda operasi yang mempresentasikan

sifat-sifat abstraks bagi "user" dengan menyembunyikan bagaimana semua itu direpresentasikan dalam representasi data level yang lebih rendah.

- ***Interface* dari modul** atau Spesifikasi bagaimana modul tersebut dilihat atau digunakan oleh "user" sementara bagian yang private tetap tersembunyi, perhatikan contoh dibawah ini:
  - Sebuah microwave oven sebagai suatu modul. Pemakai dapat menggunakannya dengan tersedianya interface panel tombol. Pemakai hanya perlu tahu untuk apa tombol-tombol tersebut dan bagaimana mengoperasikannya. Pemakai tidak perlu tahu mekanisme kerja oven tersebut secara intenal seperti kerja tabung magnetron, mikrokontroller mekanik penggerak dan lain sebagainya.).
  - Window menyediakan sejumlah button untuk dapat menutup, memperbesar, memperkecil, dan sebagainya.

15.5.1. Pemrograman Modular

Bahasa-bahasa pemrograman modern menyediakan cara-cara untuk dapat membuat sistem perangkat lunak secara modular demikian, antara lain:

- Cara untuk mengelompokkan bersama data dan metoda operasi
- Cara untuk mendefinisikan interface yang "clean"
- Cara untuk menyembunyikan rincian internal dari data dan metoda-metoda operasi
- Cara untuk kompilasi secara terpisah misalnya:
  - Turbo Pascal menyediakan fasilitas pembuatan TPU dengan definisi interface.
  - Bahasa C memungkinkan pembuatan object module (*.obj) dan library (*.a) yang dapat dilink ke modul program utama melalui interface header file (*.h).

Dalam bahasa Java memungkinkan konsep pemrograman modular, enkapsulasi, dan abstraksi data dengan disediakannya skema-skema class, interface, dan packages. Enkapsulasi dimungkinkan dengan pendefinisian **private** bagi data/metoda yang disemebunyikan dan **public** bagi data/metoda yang berperan sebagai interface.

Dalam Java terdapat keyword **interface** dengan pengertian yang lain. Keword ini digunakan sebagai suatu definisi teknis yang menyatakan sekumpulan metoda abstraks yang bisa diimplementasikan dalam pendefinisian kelas-kelas obyek. Manfaat definisi interface ini agar obyek-obyek dari sejumlah kelas yang mengimplementasikan interface ini dapat saling direfer oleh variabel beragam interface ini. Perhatikan contoh dibawah ini:

Program 15.2

```
interface Xclass {
  public void whataMethod();
}
public class Aclass implements Xclass {
  public void whataMethod() {
    ...// diimplementasikan disini
  }
}
public class Bclass implements Xclass {
  public void whataMethod() {
    ...// diimplementasikan disini
  }
}
class Mainclass {
  Xclass a,b;
  static public void main(String [] a) {
    a = new Aclass();
    b = new Bclass();
    a.whataMethod();
    b.whataMethod();
  }
}
```

Contoh lain adalah pada masalah pencarian data (*searching*). Key data bisa muncul dalam berbagai tipe data: **int**, **float**, **String**, dsb. Fasiltas interface ini dapat memungkinkan pedefinisan algoritma umum compareTo() dalam interface ComparisonKey untuk setiap tipe data. Selanjutnya kelas-kelas yang mengimplementasikan interface tersebut dapat didefinisikan sesuai denga tipe data yang dikehendaki.

Dengan fasilitas-fasilitas ini kita bisa mengembangkan sistem perangkat lunak besar dengan komponen-komponennya yang bersifat umum demikian dan selanjutnya dapat di-plug-and-play implementasi dalam berbagai tipe data. Fasilitas Java lain dalam pendefinisian modularitas dan ADT adalah keyword **package**. Keyword ini membundle sekumpulan kelas yang satu dengan yang lainnya saling memiliki relasi khusus. Pendefinisian package ini memungkinkan kompilasi secara terpisah dan diimport sebagai library. Sebagai contoh kasus untuk pembahasan modularitas dan ADT adalah suatu kelas Priority Queue.

15.5.2. Priority Queue

Priority queue (PQ) adalah suatu struktur data abstraks yang berperilaku sebagai berikut. PQ adalah kumpulan item data yang masing-masing memiliki tingkat prioritas yang bisa berbeda. Apabila item data diambil dari kumpulan tersebut maka item data yang

berprioritas paling tinggi lah yang diperoleh.

Dalam pengembangannya pertama kali adalah memformulasikan konsep abstraksnya tanpa harus memikirkan tipe datanya, apalagi implementasi rincinya. Implementasinya sendiri dapat dilakukan dalam dua representasi level bawah yang berbeda: array atau linked-list yang dapat belakangan kita buat.

Banyak ditemukan dalam masalah-masalah penjadwalan, misalnya untuk sistem antrian untuk pelayanan pasien dalam ruang praktek dokter, penjadwalan pembayaran tagihan, dsb. Prioritas bisa didefinisikan dalam beberapa aspek. PQ dapat digunakan juga untuk pengurutan data kengan key data sebagai harga prioritas: sejumlah data tak terurut dimasukkan ke dalam PQ lalu setelah semua masuk satu demi satu data diambil dari PQ. Pada pengurutan menaik maka data yang diambil ditaruh dari kanan ke kiri elemen-elemen array. Pada pengurutan menurun harga key bisa kebalikan dari harga prioritas, alterntifnya data yang di ambil satu demi satu di taruh dari kiri ke kanan elemen-elemen array.

Terdapat sejumlah metoda operasi yang menjadi 'interface': bagi pemakai kelas obyek ini.

- mengkonstruksikan PQ yang diinisialisasi kosong
  `PQ = new PriorityQueue();`
- mendapatkan ukuran (jumlah item data) dalam PQ,
  `n = PQ.size();`
- menyisipkan item baru dalam PQ,
  `PQ.insert(X);`
- bila PQ tidak kosong, menghapus item dengan prioritas tertinggi dalam PQ
  `X = PQ.remove();`

Secara sepintas bisa diketahui bahwa terdapat suatu proses internal (yang tidak perlu diketahui oleh user) di dalam skema ini yaitu pengurutan item data. Implementasi bagaimana pun user "tidak perduli". Contoh penggunaan dalam metoda pengurutan data berdasarkan Priority Queue:

Program 15.3

```
void priorityQueueSort(ComparisonKey[] A) {
  int n = A.length;
  PriorityQueue PQ = new PriorityQueue();  // create & inisialisasi PQ
  for (int i = 0; i < n; i++) PQ.insert(A[i]);  // mengisi PQ
  for (int i = n-1; i >= 0; i--) A[i] = PQ.remove(); // mengambil satu/satu
}
```

Selanjutnya contoh penggunaan PQ dengan tipe key integer ini dapat dilihat dalam program lebih lengkapnya. Pada saat ini definisi kelas PriorityQueue belum diimplementasikan. Sebelum mengimplementasikannya maka key data item yang sedang kerjakan ini dibungkus dalam tipe data yang generik yaitu: ComparisonKey sebagai suatu interface.

```
interface ComparisonKey {
  int compareTo(ComparisonKey value);
  String toString();
}
```

Selanjutnya ComparisonKey ini diimplementasikan sesuai dengan tipe data yang sedang digunakan, dalam hal ini misalnya kita menggunakan integer (nanti kita lihat bila key bertipe String).

Program 15.4

```
public class PQItem implements ComparisonKey {
  private int key;
  PQItem(int value) {
    key = value;
  }
  public String toString() {
    return Integer.toString(key);
  }
  public int compareTo(ComparisonKey value) {
    int a = this.key;
    int b = ((PQItem) value).key;
    return ((a==b)? 0:( (a > b)?1:-1));
  }
}
```

Pada level yang lebih rendah kelas priority queue ini dapat diimplementasikan baik dengan array atau dengan linked list, dimana hal tersebut dijelaskan sebagai berikut:

- Versi Implementasi Unsorted Array (berukuran dinamis)
  PQ menyimpan item data dalam suatu array yang apabila penuh panjangnya dapat diperbesar secara otomatis. Item data masuk begitu saja di bagian belakang. Saat di-*remove* maka item data dengan key terbesar dicari (dilakukan *sequential search*). Key yang diremove tempatnya digantikan oleh key yang paling belakang (tidak melakukan pergeseran!)
- Versi Implementasi Sorted Linked-list
  Saat item data masuk dalam PQ maka langsung ditempatkan pada posisi yang sesuai dalam urutan menurun (dilakukan pencarian posisi terlebih dulu lalu dilakukan insert), dan saat di-*remove* maka cukup dilakukan *delete-first* (node pertama yang diambil.

Kedua implementasi tersebut berfungsi sama tapi dengan implementasi yang berbeda namun sekali lagi user melihatnya sesuai dengan abstraksi spesifikasi PQ yang ditentukan. Dalam prakteknya maka kedua versi itu bisa dipilih tanpa mengubah modul-modul pemakainya selama spesifikasinya ditaati.

Selain implementasinya yang bisa berbeda tanpa mengubah rancangan modul tingkat di atasnya, representasi key data pun dapat diimplementasikan secara berbeda

karena ComparisonKey yang didefinisikan abstraks tersebut. Dengan abstraksi ComparisonKey di atas maka key bertipe lain dapat menggantikan tipe key integer dalam contoh di atas.ComparisonKey ini memiliki konsep Representation Independent Module. Berikut ini representasi String untuk key.

Program 15.5

```
class PQItem implements ComparisonKey {
  private  String  key;
  PQItem(String value) {
    key = value;
  }
  public String toString() {
    return key;
  }
  public int compareTo(ComparisonKey value) {
    String a = this.key;
    String b = ((PQItem)value).key;
    return a.compareTo(b);       // metoda compareTo dalam kelas String
  }
}
```

Dalam program aplikasinya perubahan terjadi pada bagian data, sementara sejumlah metoda tetap sama.

15.6. Modularitas dan Penyebunyian Informasi

Saat program menjadi semakin besar terjadi situasi berbahaya: rumit, sulit melakukan debug, dan hampir tidak mungkin di modifikasi kecuali dikembangkan dengan taat dan secara terstruktur. Penulisan program komputer dilakukan melalui medium yang memungkinkan bervariasinya cara penulisan ekspresi-ekspresi yang potensial dapat menyebabkan masalah yang besar. Dekomposisi ke dalam komponen-komponennya pun dapat bervariasi, bisa saling kait-mengkait secara kompleks atau teratur dalam layer-layer.

Abstraksi prosedural: menyatakan fungsi "apa" (*what*) yang dilakukan suatu prosedur. Kita perlu mendefinisikan "apa" dari suatu prosedur terpisah dari "bagaimana" (*how*) cara yang dilakukan prosedur untuk mencapai fungsi tersebut. Setelah terdefinisi dengan baik abstraksi prosedur ini kemudian kita boleh menuliskan rincian "bagaimana" tersebut, bahkan suatu saat misalnya untuk peningkatan efisiensi maka metodologi/algoritmanya dapat disubstitusikan tanpa mengubah abstraksi proseduralnya.

Jadi pemisahan tersebut memberi manfaat:

- Memudahkan penggunaan -- hanya perlu tahu apa yang dilakukan oleh prosedur P
- Memudahkan modifikasi -- rincian penulisan P dapat diubah tabpa mengganggu program secara keseluruhan (di mana saja pada pemanggilan-pemanggian P).

Secara umum pendefinisian abstraksi dengan baik pada perancangan program dapat mengendalikan kompleksitas program serta meningkatkan modifiabilitas. Penyembunyian informasi (enkapsulasi) baik variabel ataupun metoda dalam modul-modul bermanfaat dalam:

- Menghindari interferensi entitas lokal dalam modul terhadap entitas bernama sama di luar modul
- Menghindari interferensi entitas diluar modul terhadap entitas lokal dalam modul terhadap entitas bernama sama

Bahasa berorientasi obyek semacam C++ atau Java mengakomodasikan enkapsulasi ini dengan baik dengan deklarasi private. Dalam Java terdapat tingkatan lebih lanjut sepain private, yaitu: package, protected, baru kemudian public. Istilah package dalam Java memiliki konsep yang sama dengan istilah unit atau module dalam bahasa pemrograman lain.

15.7. Interface

Interface adalah jenis khusus dari blok yang hanya berisi method signature(atau constant). Interface mendefinisikan sebuah(signature) dari sebuah kumpulan method tanpa tubuh. Interface mendefinisikan sebuah cara stkitar dan umum dalam menetapkan sifat-sifat dari class-class. Mereka menyediakan class-class, tanpa memperhatikan lokasinya dalam hirarki class, untuk mengimplementasikan sifat-sifat yang umum. Dengan catatan bahwa interface-interface juga menunjukkan polimorfisme, dikarenakan program dapat memanggil method interface dan versi yang tepat dari method yang akan dieksekusi tergantung dari tipe object yang melewati pemanggil method interface.

Kita akan menggunakan interface jika kita ingin class yang tidak berhubungan mengimplementasikan method yang sama. Melalui interface-interface, kita dapat menangkap kemiripan diantara class yang tidak berhubungan tanpa membuatnya seolah-olah class yang berhubungan.Mari kita ambil contoh class Line dimana berisi method yang menghitung panjang dari garis dan membandingkan object Line ke object dari class yang sama. Sekarang, misalkan kita punya class yang lain yaitu MyInteger dimana berisi method yang membandingkan object MyInteger ke object dari class yang sama.

Seperti yang kita lihat disini, kedua class-class mempunyai method yang mirip dimana membandingkan mereka dari object lain dalam tipe yang sama, tetapi mereka tidak berhubungan sama sekali. Supaya dapat menjalankan cara untuk memastikan bahwa dua

class-class ini mengimplementasikan beberapa method dengan tkita yang sama, kita dapat menggunakan sebuah interface untuk hal ini. Kita dapat membuat sebuah class interface, katakanlah interface Relation dimana mempunyai deklarasi method pembanding. Relasi interface dapat dideklarasikan sebagai,

```
public interface Relation
{
public boolean isGreater( Object a, Object b);
public boolean isLess( Object a, Object b);
public boolean isEqual( Object a, Object b);
}
```

Alasan lain dalam menggunakan interface pemrograman object adalah untuk menyatakan sebuah interface pemrograman object tanpa menyatakan classnya. Seperti yang dapat kita lihat nanti dalam bagian Interface vs class, kita dapat benar-benar menggunakan interface sebagai tipe data. Pada akhirnya, kita perlu menggunakan interface untuk pewarisan model jamak dimana menyediakan class untuk mempunyai lebih dari satu superclass. Pewarisan jamak tidak ditunjukkan di Java, tetapi ditunjukkan di bahasa berorientasi object lain seperti C++.

Perbedaan utama antara sebuah interface dan sebuah class abstract: method interface tidak punya tubuh, sebuah interface hanya dapat mendefinisikan konstanta dan interface tidak langsung mewariskan hubungan dengan class istimewa lainnya, mereka didefinisikan secara independent.

15.8. Interface dan Class

Satu ciri umum dari sebuah interface dan class adalah pada tipe mereka berdua. Ini artinya bahwa sebuah interface dapat digunakan dalam tempat-tempat dimana sebuah class dapat digunakan. Sebagai contoh, diberikan class Person dan interface PersonInterface, berikut deklarasi yang benar:

```
PersonInterface pi = new Person();
Person pc = new Person();
```

Bagaimanapun, Kita tidak dapat membuat instance dari sebuah interface. Perhatikan contoh potongan program sebagai berikut:

```
PersonInterface pi = new
PersonInterface();
```

```
//COMPILE
/ERROR!!!
```

Ciri umum lain baik interface maupun class dapat mendefinisikan method. Bagaimanapun, sebuah interface tidak punya sebuah kode implementasi sedangkan pada class memiliki salah satunya.

Untuk membuat interface, kita tulis sebagai berikut:

```
public interface [InterfaceName]
{
    //beberapa method tanpa isi
}
```

Sebagai contoh, mari kita membuat sebuah interface yang digunakan untuk mendefinisikan hubungan

antara dua object menurut urutan asli dari object.

```
public interface Relation
{
    public boolean isGreater( Object a, Object b);
    public boolean isLess( Object a, Object b);
    public boolean isEqual( Object a, Object b);
}
```

Sekarang, penggunaan interface, kita gunakan kata kunci **implements**. Perhatikan contoh program dibawah ini:

Program 15.6

```
// Class ini mendefinisikan segmen garis
public class Line implements Relation
{
    private double x1;
    private double x2;
    private double y1;
    private double y2;
    public Line(double x1, double x2, double y1, double y2){
    this.x1 = x1;
    this.x2 = x2;
    this.y1 = y1;
    this.y2 = y2;
    }

    public double getLength(){
        double length = Math.sqrt((x2-x1)*(x2-x1) +
        (y2-y1)* (y2-y1));
        return length;
    }

    public boolean isGreater( Object a, Object b){
        double aLen = ((Line)a).getLength();
        double bLen = ((Line)b).getLength();
        return (aLen > bLen);
    }

    public boolean isLess( Object a, Object b){
        double aLen = ((Line)a).getLength();
        double bLen = ((Line)b).getLength();
        return (aLen < bLen);
    }
```

```
    public boolean isEqual( Object a, Object b){
        double aLen = ((Line)a).getLength();
        double bLen = ((Line)b).getLength();
        return (aLen == bLen);
    }
}
```

Ketika class kita mencoba mengimplementasikan sebuah interface, selalu pastikan bahwa Kita mengimplementasikan semua method dari interface, jika tidak, Kita akan menemukan kesalahan ini,

```
Line.java:4: Line is not abstract and does not override abstract
method isGreater(java.lang.Object,java.lang.Object) in
Relation
public class Line implements Relation
^
1 error
```

15.9. Hubungan dari Interface ke Class

Sebuah class dapat mengimplementasikan sebuah interface selama kode implementasi untuk semua method yang didefinisikan dalam interface tersedia. Hal lain yang perlu dicatat tentang hubungan antara interface ke class-class yaitu, class hanya dapat mengextend satu superclass, tetapi dapat mengimplementasikan banyak interface.

Sebuah contoh dari sebuah class yang mengimplementasikan interface adalah sebagai berikut:

```
public class Person implements
PersonInterface,
LivingThing,
WhateverInterface {
//beberapa kode di sini
}
```

Contoh lain dari class yang meng-extend satu superclass dan mengimplementasikan sebuah interface adalah,

```
public class ComputerScienceStudent
extends Student
implements PersonInterface,
LivingThing {
//beberapa kode di sini
}
```

Catatan bahwa sebuah interface bukan bagian dari hirarki pewarisan class. Class yang tidak berhubungan dapat mengimplementasikan interface yang sama.

15.10. Pewarisan Antar Interface

Interface bukan bagian dari hirarki class. sedangkan interface dapat mempunyai hubungan pewarisan antara mereka sendiri. misalnya, ketika kita punya dua interface **StudentInterface** dan interface **PersonInterface**. Jika StudentInterface meng- extend PersonInterface, maka ia akan mewariskan semua deklarasi method dalam PersonInterface.

```
public interface PersonInterface {
. . .
}
public interface StudentInterface
extends PersonInterface {
. . .
}
```

15.13.Soal Latihan

Jawablah soal latihan dibawah ini dengan baik dan benar.

1. Apa yang dimaksud dengan pemrograman Object oriented dan prosedural
2. Sebutkan perbedaan antara kedua program tersebut
3. Berilah gambaran yang jelas mengenai pemroraman OOP
4. Apa yang dimaksud dengan abstraksi
5. Apa yang dimaksud dengan enkapsulasi
6. Apa yang dimaksud dengan pewarisan
7. Apa yang dimaksud dengan imutable dan mutable objek
8. Apa yang dimaksud dengan bahasa pemrograman modular dan buatlah program sederhana
9. Apa yang dimaksud dengan Priority queue (PQ)

BAB 16
INHERITANCE, FRIENDS, POLYMORPHISM DAN OVERLOADING

16.1. Menggunakan Obyek dan Class

Untuk memahami tentang Class ada baiknya kita bahas mengenai Struct yang merupakan suatu perintah dalam bahasa C++ sebagai pembentuk tipe data baru. Sebuah contoh misal tentang kartu pelajar. Di dalam kartu pelajar terdapat keterangan tentang Identitas Sekolah, Nama Siswa, Nomor Induk Siswa, Alamat Siswa, Jurusan/Kelas. Pengertian data-data tersebut harus menjadi satu tidak terpisah-pisah (Identitas Sekolah sendiri, Nama Siswa sendiri, dan sebagainya).

Jadi seakan-akan persis seperti kalau kita melihat kartu pelajar di atas, dalam satu kartu terdapat beberapa informasi sekaligus. Untuk mewujudkan hal tersebut bahasa C++ menyediakan *keyword* yang disebut Struct. Contoh penggunaan sebagai berikut:

Program 16.1

```
#include <cstdlib>
#include <iostream>

using namespace std;

struct KartuPelajar
```

```
{
   char Sekolah[20];
   char Nama[25];
   char NIS[12];
   char Alamat[25];
   char Kelas[10];
};

int main(int argc, char *argv[])
{
   KartuPelajar data;

   cout<<"Pengisian Data Kartu Pelajar"<<endl;
   cout<<"Asal Sekolah : ";
   cin.getline(data.Sekolah, sizeof(data.Sekolah));
   cout<<"Nama Siswa : ";
   cin.getline(data.Nama, sizeof(data.Nama));
   cout<<"N I S : ";
   cin.getline(data.NIS, sizeof(data.NIS));
   cout<<"Alamat : ";
   cin.getline(data.Alamat, sizeof(data.Alamat));
   cout<<"Jurusan/Kelas : ";
   cin.getline(data.Kelas, sizeof(data.Kelas));
   cout<<endl<<endl;

   cout<<"Hasil Pengisian Data "<<endl;
   cout<<"Asal Sekolah : "<<data.Sekolah<<endl;
   cout<<"Nama Siswa : "<<data.Nama<<endl;
   cout<<"N I S : "<<data.NIS<<endl;
   cout<<"Alamat : "<<data.Alamat<<endl;
   cout<<"Jurusan/Kelas : "<<data.Kelas<<endl;
   system("PAUSE");
   return EXIT_SUCCESS;
}
```

Keluaran dari program diatas adalah sebagai berikut:

```
Pengisian Data Kartu Pelajar
Asal Sekolah : SMA N 1 Yogyakarta
Nama Siswa  : Badu Mahir Banget
N I S : 1234
Alamat : Jl. Antah Berantah 2009
Jurusan / Kelas : X IPA 1
```

Hasil Pengisian Data

```
Pengisian Data Kartu Pelajar
Asal Sekolah : SMA N 1 Yogyakarta
Nama Siswa  : Badu Mahir Banget
N I S : 1234
Alamat : Jl. Antah Berantah 2009
Jurusan / Kelas : X IPA 1
```

Lebih lanjut, struct dalam program di atas merupakan tipe data baru yang bernama KartuPelajar, ini sama dengan char, int, float dan sebagainya. Hal itu lebih diyakinkan pada main() ada perintah KartuPelajar data; dapat dijelaskan bahwa data merupakan variable dengan tipe KartuPelajar. *Keyword* struct dalam bahasa C++ dapat digantikan dengan class seperti contoh di bawah ini:

```
class KartuPelajar
{
  public:
    char Sekolah[20];
    char Nama[25];
    char NIS[12];
    char Alamat[25];
    char Kelas[10];
};
```

Potongan program di atas merupakan pembuatan tipe data baru juga namun menggunakan class, bukan struct. Program selengkapnya seperti berikut:

Program 16.2

```
#include <cstdlib>
#include <iostream>

using namespace std;

class KartuPelajar
{
  public:
   char Sekolah[20];
   char Nama[25];
   char NIS[12];
   char Alamat[25];
   char Kelas[10];
};

int main(int argc, char *argv[])
{
   KartuPelajar data;
```

```
  cout<<"Pengisian Data Kartu Pelajar"<<endl;
  cout<<"Asal Sekolah : ";
  cin.getline(data.Sekolah, sizeof(data.Sekolah));
  cout<<"Nama Siswa : ";
  cin.getline(data.Nama, sizeof(data.Nama));
  cout<<"N I S : ";
  cin.getline(data.NIS, sizeof(data.NIS));
  cout<<"Alamat : ";
  cin.getline(data.Alamat, sizeof(data.Alamat));
  cout<<"Jurusan/Kelas : ";
  cin.getline(data.Kelas, sizeof(data.Kelas));
  cout<<endl<<endl;

  cout<<"Hasil Pengisian Data"<<endl;
  cout<<"Asal Sekolah : "<<data.Sekolah<<endl;
  cout<<"Nama Siswa : "<<data.Nama<<endl;
  cout<<"N I S : "<<data.NIS<<endl;
  cout<<"Alamat : "<<data.Alamat<<endl;
  cout<<"Jurusan/Kelas : "<<data.Kelas<<endl;
  system("PAUSE");
  return EXIT_SUCCESS;
}
```

Keluaran program di atas:

```
Pengisian Data Kartu Pelajar
Asal Sekolah : SMA N 3 Yogyakarta
Nama Siswa  : Badu Pakar Banget
N I S : 1500
Alamat : Jl. Kemana-mana 007
Jurusan / Kelas : X IPA 3
```

Hasil Pengisian Data

```
Pengisian Data Kartu Pelajar
Asal Sekolah : SMA N 3 Yogyakarta
Nama Siswa  : Badu Pakar Banget
N I S : 1500
Alamat : Jl. Kemana-mana 007
Jurusan / Kelas : X IPA 3
```

Lalu apa bedanya selain *keyword* struct diganti dengan class dan ditambah public: dalam class tersebut ? Di sinilah konsep *object oriented programming* (pemrograman berorientasi obyek) seharusnya

berbeda. Dalam pemrograman berorientasi obyek antara data atau variable dan fungsinya menjadi satu kesatuan tidak terpisah seperti struct di atas. Sehingga fungsi pengisian data dan pencetakan data menjadi satu dalam class, contoh lengkap sebagai berikut:

Program 16.3

```
#include <cstdlib>
#include <iostream>

using namespace std;

class KartuPelajar
{
  private:
    char Sekolah[20];
    char Nama[25];
    char NIS[12];
    char Alamat[25];
    char Kelas[10];

  public:
    void pengisian()
    {
      cout<<"Pengisian Data Kartu Pelajar"<<endl;
      cout<<"Asal Sekolah : ";
      cin.getline(Sekolah, sizeof(Sekolah));
      cout<<"Nama Siswa : ";
      cin.getline(Nama, sizeof(Nama));
      cout<<"N I S : ";
      cin.getline(NIS, sizeof(NIS));
      cout<<"Alamat : ";
      cin.getline(Alamat, sizeof(Alamat));
      cout<<"Jurusan/Kelas : ";
      cin.getline(Kelas, sizeof(Kelas));
      cout<<endl<<endl;
    };
    void cetak()
    {
      cout<<"Hasil Pengisian Data"<<endl;
      cout<<"Asal Sekolah : "<<Sekolah<<endl;
      cout<<"Nama Siswa : "<<Nama<<endl;
      cout<<"N I S : "<<NIS<<endl;
      cout<<"Alamat : "<<Alamat<<endl;
```

```
      cout<<"Jurusan/Kelas : "<<Kelas<<endl;
    };
};

int main(int argc, char *argv[])
{
   KartuPelajar data;

   data.pengisian();
   data.cetak();
   system("PAUSE");
   return EXIT_SUCCESS;
}
```

Dalam program di atas dapat dilihat bahwa antara variable dan *method* menjadi satu dalam class. Selain itu ada perubahan dari contoh sebelumnya bahwa variable berubah sifat dari public menjadi private. Hal ini dimaksudkan untuk penyembunyian data (*information hiding*) dari pengguna, sehingga pengguna tidak dapa mengubah-ubah data secara langsung tetapi harus melalui pemanggilan class atau penurunan (*inheritance*) class. Sedangkan *method* pada class bersifat public, tentunya dimaksudkan agar dapat dipakai secara umum dari class yang lain atau class turunannya.

Pada contoh program di atas konten dari *method* ditulis langsung dalam class, itu merupakan cara pertama. Ada cara kedua penulisan konten *method* di luar class supaya lebih leluasa apabila kontennya panjang. Program lengkapnya seperti berikut:

Program 16.4

```
#include <cstdlib>
#include <iostream>

using namespace std;

class KartuPelajar
{
  private:
    char Sekolah[20];
    char Nama[25];
    char NIS[12];
    char Alamat[25];
    char Kelas[10];

  public:
```

```
    void pengisian();
    void cetak();
};

void KartuPelajar::pengisian()
{
  cout<<"Pengisian Data Kartu Pelajar"<<endl;
  cout<<"Asal Sekolah : ";
  cin.getline(Sekolah, sizeof(Sekolah));
  cout<<"Nama Siswa : ";
  cin.getline(Nama, sizeof(Nama));
  cout<<"N I S : ";
  cin.getline(NIS, sizeof(NIS));
  cout<<"Alamat : ";
  cin.getline(Alamat, sizeof(Alamat));
  cout<<"Jurusan/Kelas : ";
  cin.getline(Kelas, sizeof(Kelas));
  cout<<endl<<endl;
};

void KartuPelajar::cetak()
{
  cout<<"Hasil Pengisian Data"<<endl;
  cout<<"Asal Sekolah : "<<Sekolah<<endl;
  cout<<"Nama Siswa : "<<Nama<<endl;
  cout<<"N I S : "<<NIS<<endl;
  cout<<"Alamat : "<<Alamat<<endl;
  cout<<"Jurusan/Kelas : "<<Kelas<<endl;
};

int main(int argc, char *argv[])
{
   KartuPelajar data;

   data.pengisian();
   data.cetak();

   system("PAUSE");
   return EXIT_SUCCESS;
}
```

Penulisan dalam pemrograman berorientasi obyek lebih lazim contoh terakhir, walaupun cara pertama tidak salah. Dalam contoh cara kedua lebih menunjukkan ciri-ciri pemrograman beririentasi obyeknya dibanding cara pertama.

16.2. Realisasi Prosedur dan Fungsi dalam Class

Sebuah class telah disinggung di uraian sebelumnya bahwa prosedur maupun fungsi (*method*) merupakan satu kesatuan dengan datanya, sering disebut dengan *encapsulated*.

Prosedur dan fungsi ini merupakan dasar untuk komunikasi sebuah class dengan class lainnya, dan merupakan nyawanya class tersebut karena pengaturan proses dari suatu class melalui komponen ini. Untuk lebih jelasnya lihat contoh di bawah ini:

Program 16.5

```
#include <cstdlib>
#include <iostream>

using namespace std;

class Titik
{
    int Absis;
    int Ordinat;

  public:
    void SetAbsis(int x);
    void SetOrdinat(int y);
    int AmbilAbsis();
    int AmbilOrdinat();
};

void Titik::SetAbsis(int x)
{
   Absis = x;
}

void Titik::SetOrdinat(int y)
{
   Ordinat = y;
}
```

```
int Titik::AmbilAbsis()
{
    return Absis;
}

int Titik::AmbilOrdinat()
{
    return Ordinat;
}

int main(int argc, char *argv[])
{
   Titik A;
   A.SetAbsis(4);
   A.SetOrdinat(6);
   cout<<"Absis titik A = "<<A.AmbilAbsis()<<"\n";
   cout<<"Ordinat titik A = "<<A.AmbilOrdinat()<<"\n";
   Titik* B = &A;
   cout<<"Absis titik B = "<<B->AmbilAbsis()<<"\n";
   cout<<"Ordinat titik B = "<<B->AmbilOrdinat()<<"\n";
   A.SetAbsis(2);
   A.SetOrdinat(3);
   Titik& C = A;
   cout<<"Absis titik C = "<<C.AmbilAbsis()<<"\n";
   cout<<"Ordinat titik C = "<<C.AmbilOrdinat()<<"\n";
   system("PAUSE");
   return EXIT_SUCCESS;
}
```

Program di atas merupakan program untuk mengolah tentang titik dalam suatu bidang 2 dimensi. Sehingga mempunyai variable x dan y. Untuk dapat mengolah data-data absis serta ordinat dalam bidang 2 dimensi tersebut diperlukan prosedur untuk set absis dan prosedur untuk set ordinat. Hal itu dapat dilihat dalam prosedur SetAbsis(int x) dan prosedur SetOrdinat(int y); Sedangkan untuk mengambil data baik absis maupun ordinatnya menggunakan fungsi AmbilAbsis() dan fungsi AmbilOrdinat() guna pengolahan lebih lanjut, misal untuk mencari jarak antara dua titik dalam suatu bidang 2 dimensi. Dalam program di atas juga dimunculkan cara pemanggilan obyek dengan berbagai cara :

Titik A;	pemanggilan obyek secara biasa
Titik* B = &A;	pemanggilan obyek secara pointer
Titik& C = A;	pemanggilan obyek secara reference

Pemanggilan anggota dari obyek juga ditunjukkan contoh pada pemanggilan biasa dan reference berbeda dengan apabila pemanggilan obyek secara pointer. Lebih jelasnya seperti berikut:

Titik * B = &A; untuk pemanggilan fungsi atau prosedur anggota obyek tersebut tidak dapat menggunakan (.) titik tetapi harus menggunakan (->) panah. Seperti program di atas untuk memanggil fungsi AmbilAbsis(), maka perintahnya B->AmbilAbsis(). Begitu pula untuk prosedur SetAbsis(x), maka perintahnya B->SetAbsis(x). Contoh lain program untuk menentukan kelulusan siswa.

Program 16.6

```
#include <cstdlib>
#include <iostream>

using namespace std;

class Siswa
{
    char Nama[25];
    char NIS[12];
    double Nilai;
  public:
    void InpNama();
    void InpNIS();
    void InpNilai();
    void output();
    char* cek(double n);
};

void Siswa::InpNama()
{
    cout<<"Nama Siswa = ";
    cin.getline(Nama, sizeof(Nama));
}

void Siswa::InpNIS()
{
    cout<<"NIS Siswa = ";
    cin.getline(NIS, sizeof(NIS));
}

void Siswa::InpNilai()
```

```
{
    cout<<"Nilai Siswa = ";
    cin>>Nilai;
}

void Siswa::output()
{
    cout<<endl<<"Hasil Ujian Siswa"<<endl;
    cout<<"Nama Siswa = "<<Nama<<endl;
    cout<<"NIS Siswa  = "<<NIS<<endl;
    cout<<"Keterangan = "<<cek(Nilai)<<endl;
}

char* Siswa::cek(double n)
{
    if (n>=75)
      return "Lulus";
    else
      return "Tak Lulus";
}

int main(int argc, char *argv[])
{
   Siswa Data;
   cout<<"Masukkan Data Siswa"<<endl;
   Data.InpNama();
   Data.InpNIS();
   Data.InpNilai();
   Data.output();
   system("PAUSE");
   return EXIT_SUCCESS;
}
```

Dalam program di atas ada satu fungsi yang mengembalikan nilai string yaitu fungsi cek. Fungsi ini akan mengembalikan keterangan sebagai status siswa “Lulus” ataupun “Tak Lulus”, sehingga harus mengembalikan nilai string. Keluaran program tersebut dapat dilihat seperti di bawah ini:

```
Masukkan Data Siswa
Nama Siswa = aku
NIS Siswa = 1234
Nilai Siswa = 75

Hasil Ujian Siswa
Nama Siswa = aku
NIS Siswa    = 1234
Keterangan   = Lulus
```

16.3. Class Private , Class Public, dan Class Protected

Penggunaan private maupun public biasanya digunakan bersama-sama atau dikombinasikan. Pada dasarnya suatu class apabila tidak dituliskan secara *default* adalah private jadi pada prinsipnya class tidak dapat diakses oleh class lain dan hanya dapat diakses oleh class itu sendiri. Untuk lebih memahami kita kembali ke contoh aplikasi class yang pertama kali.

```
class Titik
{
    int Absis;
    int Ordinat;
};
```

Variabel Absis dan Ordinat merupakan variabel private walaupun tidak ditulis secara eksplisit. Apabila ditulispun hasilnya sama saja, karena secara *default* komponen dalam suatu class adalah private. Dalam hal ini variabel Absis dan Ordinat tidak bisa diakses secara langsung.

Sebagai gambaran dilihat contoh di bawah ini:

Program 16.7

```
#include <cstdlib>
#include <iostream>

using namespace std;

class Titik
{
    int X;
    int Y;
};

int main(int argc, char *argv[])
{
   Titik A;
   A.X = 4;
   A.Y = 6;
   cout<<"Absis titik A = "<<A. X<<"\n";
   cout<<"Ordinat titik A = "<<A.Y<<"\n";
   system("PAUSE");
   return EXIT_SUCCESS;
}
```

Program di atas apabila di *compile* akan terjadi kesalahan dianggap kelas Titik tidak mempunyai member bernama X dan Y. Jadi X dan Y tidak dianggap sebagai member hal ini karena private. Lain halnya jika diganti menjadi public seperti berikut:

Program 16.8

```
#include <cstdlib>
#include <iostream>

using namespace std;

class Titik
{
   public:
     int X;
     int Y;
};

int main(int argc, char *argv[])
{
   Titik A;
   A.X = 4;
   A.Y = 6;
   cout<<"Absis titik A = "<<A. X<<"\n";
   cout<<"Ordinat titik A = "<<A.Y<<"\n";
   system("PAUSE");
   return EXIT_SUCCESS;
}
```

Program di atas jikalau dijalankan tidak akan terjadi kesalahan, karena variabel X dan Y public jadi tidak ada penyembunyian informasi (*information hiding*). Keluaran dari program tersebut seperti berikut:

```
Absis titik A = 4
Ordinat titik A = 6
```

Tapi kalau bisa diubah dari luar maka data itu terlalu berisiko ibarat sebuah rumah bisa dimasuki orang sekehendak hati, hal ini tidak boleh terjadi. Oleh karena itu variabel tersebut harus tetap private tetapi diberi jalan masuk apabila diperlukan untuk diganti atau diubah nilainya. Program secara lengkapnya menjadi #include <cstdlib>

Program 16.9

```
#include <iostream>

using namespace std;

class Titik
{
```

```
 private:
    int X;
    int Y;
 public:
    void SetAbsis(int x);
    void SetOrdinat(int y);
    int AmbilAbsis();
    int AmbilOrdinat();
};

void Titik::SetAbsis(int x)
{
    X = x;
}

void Titik::SetOrdinat(int y)
{
    Y = y;
}

int Titik::AmbilAbsis()
{
    return X;
}

int Titik::AmbilOrdinat()
{
    return Y;
}

int main(int argc, char *argv[])
{
   Titik A;
   A.SetAbsis(4);
   A.SetOrdinat(6);
   cout<<"Absis titik A = "<<A.AmbilAbsis()<<"\n";
   cout<<"Ordinat titik A = "<<A.AmbilOrdinat()<<"\n";
   system("PAUSE");
   return EXIT_SUCCESS;
}
```

Jadi ada dua pasang *method* yang ditambahkan, satu untuk memasukkan data baik X maupun Y yaitu lewat SetAbsis maupun SetOrdinat. Yang kedua untuk menampilkan atau melihat variabel X ataupun Y melalui perintah AmbilAbsis dan AmbilOrdinat. Disini terlihat bahwa pintu masuk dan keluar melalui jalur tertentu tidak bisa langsung seperti apabila variabel di set public. Ibarat super market disitu ada pintu masuk dan pintu keluar, untuk mengambil barang-barang yang ada harus melalui prosedur tidak bisa begitu saja. Dapat disimpulkan untuk mengakses data dalam suatu class yang bersifat private bisa dengan membuat *method* yang bersifat public sehingga bisa memanipulasi data/variabel yang ada didalamnya. Contoh lain sebagai berikut:

Program 16.10

```
#include <cstdlib>
#include <iostream>

using namespace std;

class A
{
 private:
   int privat;
 public:
   int umum;
   int AmbilPrivat()
   { return privat; };
   void SetPrivat(int data)
   { privat = data; };

   void TesA();
};

void A::TesA()
{
    cout<<"Keluaran dari TesA"<<endl;
    cout<<"Umum = "<<umum<<endl;
    cout<<"Private = "<<privat<<endl;
}

class B: public A
{
 public:
   void TesB();
```

```
};

void B::TesB()
{
    cout<<"Keluaran dari TesB"<<endl;
    cout<<"Umum = "<<umum<<endl;
    cout<<"Private = "<<AmbilPrivat()<<endl;
}
int main(int argc, char *argv[])
{
   A ContohA;

   ContohA.SetPrivat(4);
   ContohA.umum = 20;
   ContohA.TesA();

   cout<<endl<<"Keluaran dari Main()"<<endl;
   cout<<"Umum = "<<ContohA.umum<<endl;
   cout<<"Private = "<<ContohA.AmbilPrivat()<<endl<<endl;

   B ContohB;

   ContohB.SetPrivat(40);
   ContohB.umum = 200;
   ContohB.TesB();

   cout<<endl<<"Keluaran dari Main()"<<endl;
   cout<<"Umum = "<<ContohB.umum<<endl;
   cout<<"Private = "<<ContohB.AmbilPrivat()<<endl<<endl;

   system("PAUSE");
   return EXIT_SUCCESS;
}
```

Dari contoh di atas dapat dilihat perbedaan pemakaian public dan private dalam memanipulasi data di dalam class, baik dari class itu sendiri maupun dari class turunannya. Dari class sendiri yaitu class A semua variabel maupun method bisa diakses lansung, contoh *method* TesA().

Tetapi pemanggilan dari class turunannya yaitu class B harus melalui method public-nya, contohnya pada *method* TesB() untuk menampilkan data harus pakai AmbilPrivat tidak langsung variabel privat seperti TesA(). Tetapi untuk variabel umum bisa dipanggil langsung karena sifatnya public, jadi

tidak ada beda cara pemanggilan baik TesA() maupun TesB().

Sedangkan untuk pemanggilan di main() baik ContohA yang merupakan class baru dari class A, dan Contoh B yang merupakan class baru dari class B tidak ada perbedaan cara pemanggilan. Bisa dimengerti karena class B merupaka turunan public dari class A. Sebenarnya masih ada satu lagi yaitu sifat protected. Suatu class yang memproteksi data-datanya dengan sifat protected hanya bisa diakses dari dirinya sendiri atau dari class turunannya. Lihat contoh berikut dan bandingkan dengan sifat private yang telah dibahas sebelumnya.

Program 16.11

```
#include <cstdlib>
#include <iostream>

using namespace std;

class A
{
 protected:
   int proteksi;
 public:
   int umum;
   int AmbilProteksi()
   { return proteksi; };
   void SetProteksi(int data)
   { proteksi = data; };

   void TesA();
};

void A::TesA()
{
    cout<<"Keluaran dari TesA"<<endl;
    cout<<"Umum = "<<umum<<endl;
    cout<<"Private = "<<proteksi<<endl;
}

class B: public A
{
 public:
   void TesB();
};
```

```
void B::TesB()
{
    cout<<"Keluaran dari TesB"<<endl;
    cout<<"Umum = "<<umum<<endl;
    cout<<"Protected = "<<proteksi<<endl;
}
int main(int argc, char *argv[])
{
   A ContohA;

   ContohA.SetProteksi(4);
   ContohA.umum = 20;
   ContohA.TesA();

   cout<<endl<<"Keluaran dari Main()"<<endl;
   cout<<"Umum = "<<ContohA.umum<<endl;
   cout<<"Private = "<<ContohA.AmbilProteksi()<<endl<<endl;

   B ContohB;

   ContohB.SetProteksi(40);
   ContohB.umum = 200;
   ContohB.TesB();

   cout<<endl<<"Keluaran dari Main()"<<endl;
   cout<<"Umum = "<<ContohB.umum<<endl;
   cout<<"Private = "<<ContohB.AmbilProteksi()<<endl<<endl;

   system("PAUSE");
   return EXIT_SUCCESS;
}
```

Coba lihat *method* TesA() dan TesB() cara mengakses data class sama persis (variabel maupun *method*) contoh variabel umum dan variabel proteksi. Jadi class yang bersifat protected bisa diakses langsung persis seperti class tersebut melalui turunannya. Bandingkan dengan contoh sebelumnya yaitu class yang bersifat private tidak dapat diakses langsung dari turunannya. Tetapi harus melalui *method* yang bersifat public. Namun perlu dicermati cara macam ini mempunyai kelemahan karena data tersebut seolah-olah menjadi bersifat public meskipun dideklarasikan secara private atau protected; Untuk mengatasi masalah semacam itu, C++ menyediakan satu *keyword*,

yaitu friend. Sebuah fungsi atau class yang dideklarasikan dengan sifat friend dapat mengakses semua anggota class lain, baik bersifat private, protected, maupun public.

16.4. Friend

Sebelumnya sudah di bahas bahwa ada tiga macam proteksi bagi member class, yakni: public, protected dan private. Member protected dan private tidak bisa diakses dari luar class dimana dia dideklarasikan. Walaupun demikian aturan ini dilanggar dengan adanya friend di dalam class, sehingga kita dapat membuat fungsi eksternal untuk mengakses member private dan protected.

Untuk itu kita mendeklarasikan prototype fungsi eksternal yang dimaksud yang akan menambah akses sebelumnya dengan instruksi friend di dalam deklarasi class yang dapat dishare oleh semua member. Perhatikan contoh berikut dimana kita mendeklarasikan friend function duplicate:

Program 16.12. Friend functions

```
#include <iostream.h>

class CRectangle {
   int width, height;
  public:
   void set_values (int, int);
   int area (void) {return (width * height);}
   friend CRectangle duplicate (CRectangle);
};

void CRectangle::set_values (int a, int b) {
  width = a;
  height = b;
}

CRectangle duplicate (CRectangle rectparam)
{
  CRectangle rectres;
  rectres.width = rectparam.width*2;
  rectres.height = rectparam.height*2;
  return (rectres);
}

int main () {
```

```
 CRectangle rect, rectb;
 rect.set_values (2,3);
 rectb = duplicate (rect);
 cout << "Hasil nya = "<<rectb.area();
}
```

Dari fungsi duplicate yang merupakan friend dari CRectangle, kita dapat mengakses member width dan height dari objek yang berbeda dalam tipe CRectangle. Perhatikan bahwa fungsi duplicate() bukan merupakan anggota dari class CRectangle

Dengan friend function fungsi yang bukan merupakan anggota class (diliar class) dapat mengakses member class yang sifatnya privateprotected. Secara umum penggunaan fungsi friend kelihatannya keluar dari metodologi OOP, sehingga selama masih memungkinkan lebih baik menggunakan member yang classnya sama untuk masalah ini. Seperti pada contoh sebelumnya, akan lebih sederhana jika duplicate() diletakkan dalam class CRectangle.

Program 16.13

```
#include <iostream.h>
#include <conio.h>
class contoh {
   int x;
   void fcontoh();
public:
   void fcontoh1 ();
   void fcontoh2 ();
   friend void fcontoh3 ();
};

void contoh::fcontoh()
{ cout<<"\nFungsi fcontoh anggota class contoh, bersifat private";
}

void contoh::fcontoh1()
{ x=3;
 cout<<"Fungsi contoh 1 anggota class contoh";
 fcontoh();
 cout<<"\nNilai x = "<<x;}

void contoh::fcontoh2()
{ cout<<"\n\nFungsi contoh 2 anggota class contoh";
 cout<<"\nNilai x = "<<x;}
```

```
void fcontoh3()
{ contoh b;
  b.x=10;
  cout<<"\n\nFungsi contoh 3 bukan anggota class contoh";
  cout<<"\ntetapi merupakan fungsi friend dari class contoh";
  cout<<"\nsehingga dapat mengakses x yang sifatnya private";
  cout<<"\ndan mengakses fcontoh yang juga bersifat private";
  b.fcontoh();
  cout<<"\nNilai x = "<<b.x;}

main () {
  contoh a;
  a.fcontoh1();
  a.fcontoh2();
  fcontoh3();
  getch();
}
```

Telah diketahui bahwa tujuan utama dari pemrograman berorientasi obyek adalah enkapsulasi data dan fungsi yang mengakses data di dalam class. Pengkapsulan memberikan keuntungan terutama memudahkan dalam pemeliharaan program dan juga dalam debugging. Namun pengkapsulan untuk obyek yang kompleks terkadang sulit dilakukan, sehingga adakalanya aturan terpaksa harus dilanggar demi tujuan yang harus dicapai. Seperti jika punya rumah ada kamar yang sifatnya pribadi, sehingga tidak dimasuki oleh orang lain. Tetapi kadang justru seorang teman yang dianggap sahabat boleh memasuki kamar tersebut, itulah analogi dari friend.

Friend pada C++ memang kontradiksi. Disatu pihak dapat menyederhanakan kompleksitas suatu obyek, tetapi juga dapat meruntuhkan konsep enkapsulasi, tentu saja jika terlalu banyak digunakan. Hal ini disebabkan karena friend memungkinkan pihak di luar class dapat mengakses anggota class termasuk yang bersifat private. Itulah sebabnya friend harus digunakan seperlunya dan juga digunakan secara tepat. Apabila ternyata kita terlalu banyak menggunakan friend maka sebaiknya merombak rancangan program class yang telah dibuat. Sebaiknya class-class yang menggunakan friend dipecah menjadi beberapa class lain. Secara umum, friend berguna kalau ada suatu operasi yang hendak mengakses data dari dua class yang tidak ada kaitannya. Friend juga memungkinkan untuk menampilkan isi obyek lain.

Salah satu fungsi friend adalah fungsi bukan anggota class yang dapat mengakses anggota class.

Fungsi ini dapat dipakai untuk mengakses anggota class yang bersifat private maupun protected. Gambaran fungsi friend dapat dilihat pada program berikut:

Program 16.14

```
#include <cstdlib>
#include <iostream>

using namespace std;

#include <cstdlib>
#include <iostream>

using namespace std;

class Siswa
{
   private:
      char *Nama;
      char *Kelas;
      int NIS;
   public:
      Siswa();
      void inisialisasi(int Nomor, char *Nama, char *Kelas);
      friend void tampil_data(Siswa swa);
};

Siswa::Siswa()
{
   NIS = 0;
   Nama = "";
   Kelas = "";
}

void Siswa::inisialisasi(int Nomor,char *Nama, char *Kelas)
{
    Siswa::NIS = Nomor;
    Siswa::Nama = Nama;
    Siswa::Kelas = Kelas;
}

void tampil_data(Siswa swa)
{
```

```
    cout<<"Nomor Siswa = "<<swa.NIS<<endl;
    cout<<"Nama Siswa  = "<<swa.Nama<<endl;
    cout<<"Kelas Siswa = "<<swa.Kelas<<endl;
}

int main(int argc, char *argv[])
{
   Siswa sis;
   sis.inisialisasi(1234,"Badu","X IPS");
   tampil_data(sis);
   system("PAUSE");
   return EXIT_SUCCESS;
}
```

Keluaran dari program di atas adalah sebagai berikut:

```
Nomor Siswa    = 1234
Nama Siswa     = Badu
Kelas Siswa    = X IPS
```

Pada contoh di atas, fungsi tampil_data() pada class Siswa dideklarasikan sebagai fungsi friend. Dengan demikian, fungsi ini dapat mengakses data seperti NIS. Perhatikan saat fungsi ini didefinisikan, terlihat bahwa fungsi ini seperti fungsi biasa bukan fungsi dari anggota class. Tidak didahului dengan:

```
Siswa::
```

di depan nama fungsi.
Oleh karena fungsi tampil_data() berkedudukan sebagai fungsi biasa maka cara pemanggilannya juga mengikuti kaidah fungsi biasa, seperti:

```
tampil_data(sis);
```

tidak diperbolehkan menyebut nama class di depan fungsi tersebut seperti contoh berikut:

```
sis.tampil_data();
```

seperti layaknya memanggil fungsi sebuah class. Jadi fungsi friend dianggap sebagai argument biasa bukan anggota suatu class. Sebuah contoh lain mengenai fungsi friend ditunjukkan pada program berikut. Contoh ini menggambarkan fungsi friend yang dapat mengakses tiga buah obyek sekaligus.

Program 16.15

```
#include <cstdlib>
#include <iostream>

using namespace std;
```

```
class HasilUjian
{
   private:
      int NIS;
      double Nilai;
   public:
      HasilUjian(int Nomor, double Nil);
      friend double nilai_max(HasilUjian a, HasilUjian b, HasilUjian c);
};

HasilUjian::HasilUjian(int Nomor, double Nil) : NIS(Nomor), Nilai(Nil)
{
}

double nilai_max(HasilUjian a, HasilUjian b, HasilUjian c)
{
    double maks = a.Nilai;
    maks = (b.Nilai) > maks ? b.Nilai : maks;
    maks = (c.Nilai) > maks ? c.Nilai : maks;

    return maks;
}

int main(int argc, char *argv[])
{
   HasilUjian   amir(1234, 78.6);
   HasilUjian  endah(1237, 88.5);
   HasilUjian   siti(1224, 58.6);

   cout<<"Nilai ujian terbesar : "<<nilai_max(amir,endah,siti)<<endl;
   system("PAUSE");
   return EXIT_SUCCESS;
}
```

Keluaran program di atas adalah sebagai berikut:

```
Nilai ujian terbesar : 88.5
```

Pada program di atas,

```
friend double nilai_max(HasilUjian a,
HasilUjian b,HasilUjian c)
```

berfungsi mencari nilai terbesar dengan fungsi nilai_max() dari tiga obyek a, b, dan c yang merupakan class HasilUjian. Bagaimana untuk pemakaian friend yang lebih dari satu

class ?. Program berikut suatu contoh pemakaian fungsi friend untuk mengakses dua buah class. Ada class PemainWanita dan PemainPria, kemudian ada fungsi friend info_pemain_campuran(). Program selengkapnya sebagai berikut:

Program 16.16

```
#include <cstdlib>
#include <iostream>

using namespace std;

class PemainPria;

class PemainWanita
{
  private:
    char *nama;
    char *asal;
  public:
    PemainWanita(char *nam, char *asl);
    friend void info_pemain_campuran(PemainWanita x, PemainPria y);
};

PemainWanita::PemainWanita(char *nam, char *asl)
{
    nama = nam;
    asal = asl;
}

class PemainPria
{
  private:
    char *nama;
    char *asal;
  public:
    PemainPria(char *nam, char *asl);
    friend void info_pemain_campuran(PemainWanita x, PemainPria y);
};

PemainPria::PemainPria(char *nam, char *asl)
{
    nama = nam;
    asal = asl;
```

```
}

void info_pemain_campuran(PemainWanita x, PemainPria y)
{
    cout<<"Pemain ganda campuran"<<endl;
    cout<<x.nama<<" < "<<x.asal<<" > "<<endl;
    cout<<y.nama<<" < "<<y.asal<<" > "<<endl;
}

int main(int argc, char *argv[])
{
   PemainWanita p_wanita("Anna Kournikova", "Rusia");
   PemainPria p_pria("Rafael Nadal", "Spanyol");

   info_pemain_campuran(p_wanita, p_pria);
   system("PAUSE");
   return EXIT_SUCCESS;
}
```

Keluaran program seperti berikut:

```
Pemain ganda campuran
Anna Kournikova < Rusia >
Rafael Nadal < Spanyol >
```

Pada program di atas ada baris yang berisi:

```
class PemainPria;
```

Pernyataan ini dikenal dengan istilah refernsi di depan. Hal seperti ini diperlukan mengingat pengenal class PemainPria akan digunakan pada pendeklarasian class PemainWanita, yaitu pada baris yang berisi:

```
friend void
info_pemain_campuran(PemainWanita
x, PemainPria y);
```

Deklarasi lengkap PemainPria ini dapat diletakkan di belakang. Hal yang perlu diperhatikan dalam membuat fungsi friend yang melibatkan dua buah class yang berbeda atau lebih adalah mendeklarasikan fungsi tersebut pada class-class yang hendak diakses. Bentuk Pendeklarasiannya persis sama, seperti contoh baris di atas terdapat di kedua class baik PemainWanita maupun PemainPria.

Para pemrogram C++ biasa melewatkan obyek pada fungsi friend dengan referensi. Hal seperti ini sangat bermanfaat untuk obyek berukuran besar. Dengan melewatkan secara referensi, prosesnya menjadi lebih cepat. Deklarasi fungsi friend di atas bisa diuah menjadi seperti berikut ini :

```
friend void
info_pemain_campuran(const
PemainWanita &x, const PemainPria
&y);
```

Keyword const di atas menunjukkan bahwa anggota obyek tidak dimaksudkan untuk diubah. Sedangkan symbol & di depan nama argument x dan y menyatakan obyek tersebut dilewatkan berdasarkan referensi. Selain itu perlu juga disesuaikan pada bagian definisi fungsi, perubahannya sebagai berikut:

```
void info_pemain_campuran(const
PemainWanita &x, const PemainPria &y)
{
   …..
}
```

Secara lengkap sebagai berikut:

Program 16.17

```
#include <cstdlib>
#include <iostream>

using namespace std;

class PemainPria;

class PemainWanita
{
  private:
    char *nama;
    char *asal;
  public:
    PemainWanita(char *nam, char *asl);
    friend void info_pemain_campuran(const PemainWanita &x, const PemainPria
&y);
};

PemainWanita::PemainWanita(char *nam, char *asl)
{
    nama = nam;
    asal = asl;
}

class PemainPria
{
  private:
    char *nama;
    char *asal;
```

```
  public:
    PemainPria(char *nam, char *asl);
    friend void info_pemain_campuran(const PemainWanita &x, const PemainPria
&y);
};

PemainPria::PemainPria(char *nam, char *asl)
{
    nama = nam;
    asal = asl;
}

void info_pemain_campuran(const PemainWanita &x, const PemainPria &y)
{
    cout<<"Pemain ganda campuran"<<endl;
    cout<<x.nama<<" < "<<x.asal<<" > "<<endl;
    cout<<y.nama<<" < "<<y.asal<<" > "<<endl;
}

int main(int argc, char *argv[])
{
   PemainWanita p_wanita("Anna Kournikova", "Rusia");
   PemainPria p_pria("Rafael Nadal", "Spanyol");

   info_pemain_campuran(p_wanita, p_pria);
   system("PAUSE");
   return EXIT_SUCCESS;
}
```

Keluaran program tetap sama dengan program sebelumnya. Suatu fungsi anggota kelas dapat berkedudukan sebagai fungsi friend. Fungsi seperti ini (biasa disebut fungsi anggota friend) dapat mengakses anggota class yang mendeklarasikannya. Sebagai contoh program di atas dapat dimodifikasi agar fungsi info_pemain_campuran() menjadi fungsi anggota class PemainWanita dan dideklarasikan sebagai frien pada class PemainPria. Mula-mula class PemainWanita perlu dideklarasikan sebagai berikut:

Program 16.18

```
class PemainWanita
{
  private:
    char *nama;
```

```
    char *asal;
  public:
    PemainWanita(char *nam, char *asl);
    void info_pemain_campuran(const PemainPria &x);
};
```

Nampak sekarang bahwa fungsi info_pemain_campuran() berkedudukan sebagai fungsi anggota dari class PemainWanita. Argumen dari fungsi ini berupa referensi dari obyek ber-clas PemainPria. Penyebutan class PemainPria mengharuskan adanya pemberitahuan kepada compiler bahwa PemainPria adalah class. Oleh karena itu, sebelum dideklarasikan PemainPria perlu diberikan referensi dimuka seperti contoh program sebelumnya. Kemudian bagaimana pendeklarasian pada class PemainPria ? Pada class ini fungsi info_pemain_campuran() dideklarasikan sebagai fungsi friend:

Program 16.19

```
class PemainPria
{
  private:
    char *nama;
    char *asal;
    friend void PemainWanita::info_pemain_campuran(const PemainPria &x);
  public:
    PemainPria(char *nam, char *asl);
};
```

Perhatikan ada perubaha pada fungsi info_pemain_campuran(), pada class PemainPria menjadi bersifat private dan memakai *keyword* friend. Selain itu juga ditambahkan nama class yang menjadi friend yaitu PemainWanita. Program selengkapnya sebagai berikut:

Program 16.20

```
#include <cstdlib>
#include <iostream>

using namespace std;

class PemainPria;

class PemainWanita
{
```

```
  private:
    char *nama;
    char *asal;
  public:
    PemainWanita(char *nam, char *asl);
    void info_pemain_campuran(const PemainPria &x);
};

PemainWanita::PemainWanita(char *nam, char *asl)
{
    nama = nam;
    asal = asl;
}

class PemainPria
{
  private:
    char *nama;
    char *asal;
    friend void PemainWanita::info_pemain_campuran(const PemainPria &x);
  public:
    PemainPria(char *nam, char *asl);
};

PemainPria::PemainPria(char *nam, char *asl)
{
    nama = nam;
    asal = asl;
}

void PemainWanita::info_pemain_campuran(const PemainPria &x)
{
    cout<<"Pemain ganda campuran"<<endl;
    cout<<nama<<" < "<<asal<<" > "<<endl;
    cout<<x.nama<<" < "<<x.asal<<" > "<<endl;
}

int main(int argc, char *argv[])
{
   PemainWanita p_wanita("Anna Kournikova", "Rusia");
   PemainPria p_pria("Rafael Nadal", "Spanyol");

   p_wanita.info_pemain_campuran(p_pria);
```

```
   system("PAUSE");
   return EXIT_SUCCESS;
}
```

Apa yang dimaksud dengan class friend ? Sesungguhnya suatu class dapat dijadikan sebagai friend dari class yang lain. Hal seperti ini diperlukan apabila ada dua class dan dibutuhkan mengakses bagian private maupun protected dari class-class tersebut. Kita lihat contoh berikut. Misal ada dua kelas yaitu D3 dan S1, keduanya mempunyai data tentang banyaknya ruang atau kebutuhan ruang kelas. Serta ruangan D3 dapat dipakai oleh S1, maka pendeklarasian masing-masing class sebagai berikut:

Program 16.21

```
class D3
{
  private:
    int ruangD3;
  public:
    D3()
    { ruangD3 = 10; }
    friend class S1;
};

class S1
{
  private:
    int ruangS1;
  public:
    S1()
    { ruangS1 = 6; }
    void info_ruang(D3 x);
};
```

Pada contoh ini, x adalah obyek dengan class D3 yang menjadi argument fungsi info_ruang() pada class S1. Fungsi dimaksud untuk mengakses data ruang_D3 milik obyek x(yang ber-class D3). Masalahnya adalah bahwa data ruangD3 bersifat private, sehingga class S1 tidak bisa mengaksesnya. Persoalan yang sama tetap muncul sekiranya *keyword* private pada class D3 diganti dengan protected. Sebab seperti diketahui, anggota yang bersifat protected hanya bisa diakses kalau class ini diwariskan ke class turunan. Solusinya dengan class friend seperti yang terlihat pada baris dibawah konstructor D3(). Dan dengan pernyataan ini dapat diletakkan dimana saja pada class

D3, private, protected, maupun public. Keluaran program di atas sebagai berikut:

```
Ruang D3        = 10
Ruang S1        = 6
Total Ruang S1  = 16
```

16.5. Friend class

Sebagaimana kita dapat mendefinisikan friend function, kita juga dapat mendefinisikan class sebagai friend dari class lain, yang mengijinkan satu class mengakses member private dan protected dari class lainnya.

Program 16.22

```
#include <cstdlib>
#include <iostream>

using namespace std;
class CSquare;

class CRectangle {
   int width, height;
  public:
   int area (void)
     {return (width * height);}
   void convert (CSquare a);
};

class CSquare {
  private:
   int side;
  public:
   void set_side (int a)
     {side=a;}
   friend class CRectangle;
};

void CRectangle::convert (CSquare a) {
  width = a.side;
  height = a.side;
}

int main () {
  CSquare sqr;
  CRectangle rect;
```

```
 sqr.set_side(4);
 rect.convert(sqr);
 cout << "Hasilnya = "<<rect.area()<<"\n";
 system("PAUSE");
 return EXIT_SUCCESS;
}
```

Keluaran program diatas adalah sebagai berikut:

```
Hasilnya = 16
```

Dalam contoh ini kita mendeklarasikan CRectangle sebagai friend dari CSquare, sehingga CRectangle dapat mengakses member private dan protected yang dimiliki class CSquare, yakni CSquare::side, yang mendefinisikan lebar dari segiempat. Kita juga melihat adanya instruksi baru di awal program, yakni

```
class CSquare;
```

yang merupakan prototype dari class CSquare. Hal ini diperlukan karena dalam deklarasi CRectangle kita mengacu kepada CSquare (sebagai parameter dalam fungsi convert()). Definisi CSquare disertakan di bagian akhir, sehingga jika kita tidak mengikutkan prototype CSquare diawal class instruksi tersebut akan dianggap salah (tidak dikenali).

Perhatikan bahwa penggunaan friend tidak akan berfungsi jika kita tidak menuliskannya secara eksplisit. Dalam contoh CSquare, CRectangle tidak dianggap sebagai suatu friend class, sehingga CRectangle dapat mengakses member private dan protected milik CSquare tetapi tidak sebaliknya. Agar dua buah class dapat saling mengakses member private dan protected nya maka keduanya harus dideklarasikan sebagai friend class dari yang lain seperti pada contoh berikut:

Program 16.23

```
#include <iostream.h>
#include <conio.h>

class contoh {
   int x;
public:
   void fcontoh1 ();
   void fcontoh2 ();
   friend class cth;
};

class cth {
   int y;
```

```
   void fcth();
public:
   void fcth1 ();
   friend class contoh;
};

void cth::fcth()
{ contoh d;
 d.x=5;
 cout<<"\nFungsi cth anggota class cth (private)";
 cout<<"\nNilai x = "<<d.x;
 }

void contoh::fcontoh1()
{ x=3;
 cout<<"Fungsi contoh 1 anggota class contoh";
 cout<<"\nNilai x = "<<x;}

void contoh::fcontoh2()
{ cth m;
 m.y=5;
 cout<<"\n\nFungsi contoh 2 anggota class contoh";
 cout<<"\nNilai x = "<<x;
 cout<<"\nNilai y = "<<m.y;
 m.fcth();}

void cth::fcth1()
{
 y=3;
 cout<<"\n\nFungsi cth 1 anggota class cth";
 cout<<"\nNilai y = "<<y;
 fcth();
 }

main () {
 contoh a;
 cth b;
 a.fcontoh1();
 a.fcontoh2();
 b.fcth1();
 getch();
}
```

Keluaran program diatas adalah sebagai berikut:

```
Fungsi contoh 1 anggota class contoh
Nilai x = 3

Fungsi contoh 2 anggota class contoh
Nilai x = 3
Nilai y = 5
Fungsi cth anggota class cth (private)
Nilai x = 5

Fungsi cth 1 anggota class cth
Nilai y = 3
Fungsi cth anggota class cth (private)
Nilai x = 5
```

16.6. Inheritance

Salah satu hal penting dalam class adalah inheritance. Inheritance mengijinkan kita untuk membuat objek yang diturunkan dari objek lain, sehingga dimungkinkan didalamnya terdapat member lain selain membernya sendiri.

Sebagai contoh, misalnya kita ingin mendeklarasikan sederetan class yang mendeskripsikan polygon seperti CRectangle atau CTriangle. Keduanya dapat dideskripsikan dengan dua atribut yakni : alas dan tinggi. Hal ini dapat direpresentasikan dengan class CPolygon dan dari class tersebut diturunkan dua class yakni CRectangle dan CTriangle.

Class CPolygon berisi member yang umum pada semua polygon, dalam contoh ini adalah panjang dan lebar (width dan height). CRectangle dan CTriangle diturunkan dari class tersebut. Class yang diturunkan dari class lain mewarisi semua member yang ada dalam class dasarnya. Hal ini berarti bahwa jika class dasarnya memiliki member A dan kita menurunkannya ke class lain yang memiliki member B, maka class turunan akan terdiri dari A dan B. Untuk menurunkan class dari yang lain, kita menggunakan operator : (colon) dalam deklarasi class turunan dengan cara sebagai berikut:

```
class derived_class_name: public base_class_name;
```

dimana derived_class_name adalah nama class turunan dan base_class_name adalah nama class yang menjadi dasar. Public dapat diganti dengan akses lain misalnya protected atau private, dan menjelaskan akses untuk member yang diturunkan, seperti kita dapat lihat pada contoh berikut:

Program 16.24

```
#include <cstdlib>
#include <iostream.h>

class CPolygon {
 protected:
  int width, height;
 public:
  void set_values (int a, int b)
    { width=a; height=b;}
 };

class CRectangle: public CPolygon {
 public:
  int area (void)
    { return (width * height); }
 };

class CTriangle: public CPolygon {
 public:
  int area (void)
    { return (width * height / 2); }
 };

int main () {
 CRectangle rect;
 CTriangle trgl;
 rect.set_values (4,5);
 trgl.set_values (4,5);
 cout<<"\nHasil pemanggilan fungsi area oleh rect= "<<rect.area();
 cout <<"\nHasil pemanggilan fungsi area oleh trgl = "<<trgl.area();
 system("PAUSE");
 return EXIT_SUCCESS;
}
```

Keluaran program diatas adalah sebagai berikut:

```
Hasil pemanggilan fungsi area oleh rect= 20
Hasil pemanggilan fungsi area oleh trgl = 10
```

Sebagaimana kita lihat objek class CRectangle dan CTriangle masing-masing berisi member dari class CPolygon yakni : width, height dan set_values(). Protected mirip dengan private, perbedaan hanya terjadi pada class turunan. Ketika kita menurunkan class, member protected dari class dasar dapat digunakan oleh member lain dalam

class turunan, namun member private tidak demikian.

Karena kita ingin width dan height dapat diakses oleh member dalam class turunan CRectangle dan CTriangle dan tidak hanya oleh member CPolygon, kita pilih akses protected daripada private. Kita dapat mebuat ringkasan perbedaan tipe akses berdasarkan siapa yang dapat mengakses, sebagai berikut:

Access	public	protected	private
Anggota dari kelas yang sama	yes	yes	yes
Anggota dari turunan kelas	yes	yes	no
Bukan anggota	yes	no	no

Dimana "bukan anggota" merepresentasikan referensi dari luar class, seperti dari main(), dari class lain atau dari fungsi lain baik global ataupun local. Pada contoh diatas, anggota yang diturunkan kepada CRectangle dan CTriangle diikuti dengan hak akses yang sama dengan class asalnya, CPolygon:

```
CPolygon::width          //protected access
CRectangle::width        //protected access
CPolygon::set_values()   // public access
CRectangle::set_values() // public access
```

Ini dikarenakan kita menurunkan class dari class lain sebagai public, yang perlu diperhatikan adalah:

```
class CRectangle: public CPolygon;
```

public menunjukkan level minimum proteksi yang berarti member warisan class dasar (CPolygon) harus diikutkan dalam class baru (CRectangle). Level akses minimum untuk member yang diturunkan dapat diubah menjadi protected atau private selain public. Sebagai contoh, daughter adalah class yang diturunkan dari mother yang didefinisikan sebagai berikut:

```
class daughter: protected mother;
```

yang menggunakan protected sebagai minimum level akses untuk member daughter yang diwarisi dari mother. Karena itu, semua member yang sifatnya public pada mother akan menjadi protected pada daughter, yang merupakan level minimum dimana ia dapat diwariskan. Tentunya, ini tidak akan menjadi halangan bagi daughter untuk mempunyai member public. Minimum level hanya akan digunakan untuk member yang diwariskan mother.

Penggunaan umum level inheritance selain public adalah private, yang menyediakan encapsulasi sempurna dalam class dasar, kecuali jika tidak ada yang beranggapan classnya akan dapat diakses member class dasar dari turunannya. Sehingga dalam banyak kasus class diturunkan dalam bentuk public. Jika tidak ada level akses yang ditulis maka dalam class akan dianggap private, dan dalam tipe struct dianggap public.

Ada empat konversi baku dari suatu class turunan ke class basis publiknya, yaitu:

1. Obyek class turunan akan dikonversi secara implisit ke obyek class basis publiknya.
2. Reference class turunan akan dikonversi secara implisit ke reference class basis publiknya.
3. Pointer class turunan akan dikonversi secara implisit ke pointer class basis publiknya.

Pointer ke suatu member dari suatu class basis akan dikonversi secara implisit menjadi pointer ke suatu member dari suatu publicly-derived-class. Konversi-konversi dengan arah sebaliknya memerlukan *cast* secara eksplisit, walaupun perlu diingat bahwa hal ini tidak aman dalam pemrograman

C++ memungkinkan suatu class mewarisi data ataupun fungsi anggota class lain. Sifat ini disebut inheritance. Class yang mewarisi sifat class lain disebut class turunan (*derived class*), sedangkan class yang mewariskan sifat ke class lain disebut class dasar(*base class*).

Konsep inheritance pada C++ ini sebenarnya diilhami oleh sifat pewarisan dalam kehidupan nyata. Misal, mata si Upik mirip dengan mata ayahnya dan hidungnya mancung mewarisi ibunya. Keuntungan utama dengan adanya inheritance yaitu memungkinkan suatu kode yang telah ditulis mudah sekali untuk digunakan kembali. Contoh jika anda mempunyai class yang telah dipakai dan teruji. Suatu saat anda menemukan permasalahan yang sama dengan class tersebut maka class tesebut dapat dipergunakan dengan cara menurunkannya sehingga sifat-sifat yang sama bisa dimanfaatkan.

Dengan cara seperti ini pengembangan program menjadi lebih efisien dan efektif. Juga apabila diperlukan turunannya dapat menambahkan sifat-sifat baru sesuai dengan kebutuhan permasalahan permograman yang sedang dihadapi, bahkan kalau perlu bisa mengganti dengan sifat yang berbeda dengan class dasarnya.

Untuk memahami tentang konsep inheritance, marilah kita coba contoh berikut :

Program 16.25

```
#include <cstdlib>
#include <iostream>

using namespace std;

class basis
{
  private:
    int A;
    int B;
  public:
    void info_basis()
```

```
    { cout<<"Info_basis() dijalankan..."<<endl; }
};

class turunan : public basis
{
  public:
    void info_turunan()
    { cout<<"Info_turunan() dijalankan..."<<endl; }
};

int main(int argc, char *argv[])
{
   turunan anak;
   anak.info_basis();
   anak.info_turunan();
   system("PAUSE");
   return EXIT_SUCCESS;
}
```

Keluaran program diatas adalah sebagai berikut:

```
Info_basis() dijalankan...
Info_turunan() dijalankan...
```

Dari hasil keluaran terlihat bahwa class turunan dapat menjalanka fungsi info_basis() yang ada pada class dasar. Hal ini bisa terjadi karena pernyataan tambahan pada deklarasi class turunan, lebih jelasnya lihat pernyataan berikut:

```
class turunan : public basis
{
   .....
}
```

Terlihat ada tambahan :public basis, itulah yang menyebabkan bahwa class turunan adalah turunan dari class basis. Dengan demikian maka tentu saja dapat mengakses semua anggota class basis yang bersifat public. Akan tetapi anggota yang bersifat private tetap tidak dapat diakses. Jadi yang diwariskan hanya yang bersifat public. Namun demikian apabila hendak mengakses bagian private dari class basis dibuatkan fungsi pada class turunan untuk mengaksesnya, fungsi inilah nanti yang dipanggil untuk menjalankan atau memanipulasi anggota private class dasar. Jadi harus dengan perantara tidak dapat langsung.

Pada prinsipnya apabila class turunan bersifat private maka sebenarnya sifat-sifat class dasar diturunkan di bagian private, sehingga tidak dapat diakses langsung. Dan apabila class turunan bersifat public maka sifat-sifat class dasar diturunkan di bagian public, jadi dapat diakses langsung.

```
class dasar
{
    private:
        A;
    protected:
        B;
        C;
    public:
        void info_dasar();
};
class turunan:private dasar
{
    private:
        local;
        B;
        C;
        void info_dasar();
    protected:
        D;
    public:
        void info_turunan();
};
```

Bandingkan dengan program berikut ini:

```
class dasar
{
    private:
        A;
    protected:
        B;
        C;
    public:
        void info_dasar();
};
class turunan:public dasar
{
    private:
        local;
     protected:
        B;
        C;
```

```
        D;
    public:
        void info_dasar();
        void info_turunan();
};
```

Dari gambaran di atas jelas kiranya bagaimana dan dimana sifat-sifat class dasar diturunkan di class turunan. Sekali lagi bahwa anggota class dasar yang bersifat private tetap tidak diwariskan ke class turunan. Akan tetapi apabila menghendaki beberapa data anggota class dasar dapat diturunkan maka buatlah bersifat protected. Karena anggota yang bersifat protected ikut diwariskan pada class turunan (lihat gambaran di atas). *Keyword* protected ini juga dapat diterapkan untuk class turunan sebagaimana private dan public. Apabila class turunan bersifat protected maka semua anggota protected dan public class dasar akan menjadi anggota protected pada class turunan.

Untuk lebih meyakinkan bagaimana dan dimana sifat-sifat dari class dasar diwariskan, analisa program berikut yang menggunakan konstruktor dan destruktor.

Program 16.26

```
#include <cstdlib>
#include <iostream>
using namespace std;

class kendaraan
{
 private:
   char *nama;
 public:
   kendaraan(char *nm = "XXX")
   {
     nama = nm;
     cout<<"Hidupkan mesin anda ....."<<endl;
   }
   ~kendaraan()
   {
     cout<<"Matikan mesin anda ....."<<endl;
   }
   void info_kendaraan()
   {
     cout<<nama<<" sedang berjalan ....."<<endl;
   }
};
```

```
class truk : public kendaraan
{
 public:
   truk(char *nmTr) : kendaraan(nmTr)
   {
     cout<<"Hidupkan mesin truk anda ....."<<endl;
   }
   ~truk()
   {
     cout<<"Matikan mesin truk anda ....."<<endl;
   }
};

int main(int argc, char *argv[])
{
  truk fuso("TRUK FUSO");
  fuso.info_kendaraan();
  system("PAUSE");
  return EXIT_SUCCESS;
}
```

Keluaran program diatas adalah sebagai berikut:

```
Hidupkan mesin anda .....
Hidupkan mesin truk anda .....
TRUK FUSO sedang berjalan .....
Press any key to continue . . .[Enter]
Matikan mesin truk anda .....
Matikan mesin anda .....
```

Hasil eksekusi program menunjukkan konstruktor kedua class dijalankan semua, ini bisa dipahami karena konstruktor terletak pada bagian public. Begitu pula dengan destructor semua class dijalankan baik class dasar maupun class turunan. Namun nama kendaraan diisi dari masukkan nama truk bukan kendaraan. Kenapa bisa padahal nama kendaraan bersifat private. Karena pemasukan data nama melalui inisialisasi konstruktor bukan langsung pada main().

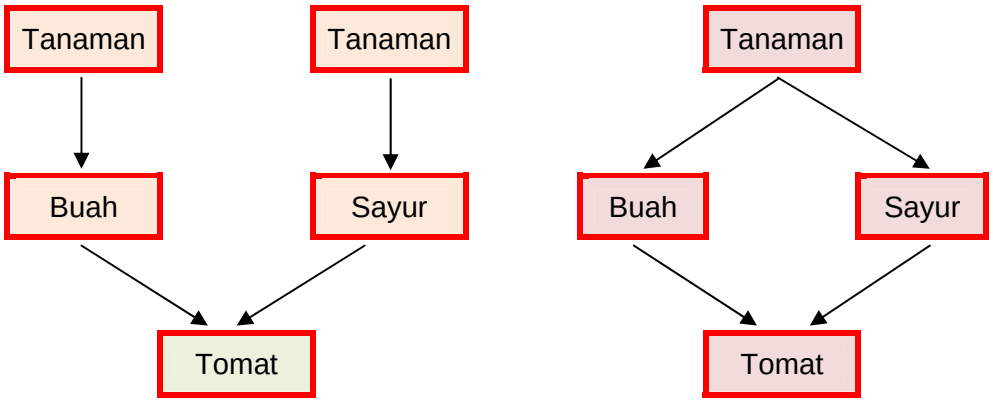
16.7. Class Basis Virtual

Penggunaan class basis virtual adalah untuk mengatasi mekanisme pewarisan default sehingga perancang/pembuat class dapat menentukan suatu shared-base-class. Tak peduli berapa kali suatu class basis virtual muncul dalam hirarkhi derivasi, hanya suatu instance dibuat dan suatu class basis virtual dispesifikasikan dengan menggunakan kata kunci virtual dalam deklarasinya. Perhatikan contoh dibawah ini:

```
class D : public virtual B {.....}
```

pada potongan program diatas menunjukkan bahwa B adalah suatu class basis virtual untuk class D. Jika suatu class basis virtual mendefinisikan konstruktor, class basis virtual itu harus mendefinisikan suatu konstrutkor yang tak memerlukan argumen. Biasanya suatu class turunan dapat menginit secara eksplisit hanya class-class basis yang langsung di atasnya. Class-class basis virtual merupakan perkecualian.

Class basis virtual di-init oleh class turunan terbawah/terakhir. perhatikan contoh berikut ini:



Gambar 16.1. Class Basis Virtual

Program 16.27

```
#include<iostream>

using namespace std;

class tanaman {
public :
```

```
tanaman() : i(0), f(0) {cout << "tanaman().\n"; }
tanaman (int init_i,float init-f):i(init-i),f(init-f)
{cout << "tanaman(int,float).\n"; }
~tanaman() {cout << "~tanaman().\n"; }

void print();
        private :
        int i;
        float f;
};

void tanaman :: print() {
        cout << "i= " << i << endl;
        cout << "f= " << f << endl;
}

class buah : virtual public tanaman {
public :
        buah (int init_j, float init_g) : j(init_j), g(init_g)
        {cout << "buah (int,float).\n"; }
        ~buah() { cout << "~buah().\n"; }

void print();
        private :
        int j;
        float g;
};

void buah :: print() {
        tanaman :: print();
        cout << "j= "<< j<< endl;
        cout << "g="<< g<< endl;
}

class sayur : public virtual tanaman {
public :
sayur (int init_k,float init_h) : k(init_k), h(init-h)
{cout << "sayur(int,float).\n"; }
~sayur() {cout << "~sayur().\n"; }

void print();
private :
int k;
```

```
float h;
};

void sayur :: print() {
        tanaman :: print() ;
        cout << "k= " << k << endl;
        cout << "h=" << h << endl;
}

class tomat : public buah, public sayur {
        public :
        tomat (int init_l, float init_m) : buah(init_l, init_m),      sayur(int init_l,
init_m),
        l(init_l+2), m(init_m +2.1)
        {cout << "tomat (int,float).\n"; }
        ~tomat() { cout << "~tomat().\n"; }

void print();
        private:
        int l;
        float m;
};

void tomat :: print() {
        tanaman :: print();
        buah :: print();
        sayur :: print();
        cout << "l= " << l << endl;
        cout <<"m= "<< m << endl;
}

main() {
        tanaman *pt = new tanaman (2, 3.1);
        cout << "pt ? print(): \n";
        pt? print();
        delete pt;
        cout << endl;
        buah *pb = new buah (2, 3.1);
        cout << "pb? print() :\n";
        pb?print();
        delete pb;
        cout << "\n";
        sayur *ps = new sayur (2, 3.1);
```

```
        cout << "ps ?print() :\n";
        ps ? print();
        delete ps;
        cout << endl;
        tomat *pto = new tomat (2, 3.1);
        cout << "pto ?print(): \n";
        pto ? print();
        delete pto;
        return 0;
}
```

Keluaran program diatas adalah sebagai berikut:

```
tanaman (int,float).
pt→print():
i=2
f=3.1
~tanaman().
tanaman().

buah(int,float).
pb→print():
i=0
f=0
j=2
g=3.1
~buah().
~tanaman().
tanaman().
sayur(int,float).
ps→print():
i=0
f=0
k=2
h=3.1
~sayur().
~tanaman().
tanaman().
buah(int,float).
sayur(int,float).
tomat(int,float).
pto→print():
i=0
```

```
f=0
i=0
f=0
j=2
g=3.1
i=0
f=0
k=2
h=3.1
l=4
m=5.2
~tomat().
~sayur().
~buah().
~tanaman().
```

Konstruktor class basis virtual selalu dijalankan sebelum konstruktor class basis nonvirtual, tak peduli posisinya dalam derivation-list. Sedangkan urutan destruktor sebaliknya, dan jika suatu class turunan melibatkan sekaligus instance public dan instance private dari suatu class basis virtual,maka yang menang adalah instance public.

Program16.28

```
class tanaman {
        public :
        void habitat();
        protected:
        short tinggi;
};
        class buah : public virtual tanaman {...};
        calss sayur : private virtual tanaman {...};
        class nangka : public buah, public sayur {...};
```

16.8. Inheritance between class

Inheritance memungkinkan kita untuk membuat objek dari objek sebelumnya, sehingga memungkinan untuk menyertakan beberapa anggota objek sebelumnya ditambah dengan anggota objeknya sendiri. Contoh, membuat class untuk mengetahui apakah segi empat (CRectangle), atau (CTriangle). Masing-masing mempunyai hal yang sama yaitu, dasar dan tinggi. Dapat direpresentasikan dengan class CPolygon kemudian diturunkan menjadi CRectangle dan CTriangle.

Class CPolygon dapat berisi anggota yang dipakai untuk setiap polygon, dalam hal ini width dan height, dan CRectangle dan

CTriangle adalah class turunannya. Class turunan akan menurunkan seluruh anggota yang dimiliki oleh class dasar(parent)nya. Jadi jika class paret mempunyai anggota A dan diturunkan pada class lain dengan anggota B,maka clas turunan ini akan memiliki A dan B.
Untuk menurunkan class, menggunakan operator : (colon) pada saat deklarasi, syntax :

class *derived_class_name*: **public** *base_class_name*;

Dimana derived_class_name adalah nama dari derived class dan base_class_name adalah nama dari class asal. **public** dapat digantikan dengan tipe akses lainnya : **protected** atau **private**, Perhatikan contoh program dibawah ini :

Program 16.29

```
#include <iostream.h>

class CPolygon {
protected:
int width, height;
public:
void set_values (int a, int b)
        { width=a; height=b;}
};

class CRectangle: public CPolygon {
public:
        int area (void)
        { return (width * height); }
};

class CTriangle: public CPolygon {
        public:
        int area (void)
        { return (width * height / 2); }
};

int main () {
        CRectangle rect;
        CTriangle trgl;
        rect.set_values (4,5);
        trgl.set_values (4,5);
        cout << rect.area() << endl;
        cout << trgl.area() << endl;
  system("PAUSE");
```

```
  return EXIT_SUCCESS;
}
```

Keluaran program diatas adalah :

```
20 10
```

Class CRectangle dan CTriangle masing-masing mengandung anggota dari CPolygon, yaitu : width, height dan set_values().

16.9. Multiple inheritance

Dalam C++ memungkinkan untuk menurunkan field atau method dari satu atau lebih class dengan menggunakan operator koma dalam deklarasi class turunan. Contoh, akan dibuat class untuk menampilkan dilayar (COutput) dan akan diturunkan ke class CRectangle and CTriangle maka dapat dituliskan :

```
class CRectangle: public CPolygon, public
COutput { class CTriangle: public
CPolygon, public COutput {
```

perhatikan contoh program multiple inheritance dibawah ini:

Program 16.30

```
#include <iostream.h>

class CPolygon {
protected:
int width, height;

public:
void set_values (int a, int b)
{ width=a; height=b;}
};

class COutput {
        public:
        void output (int i); };
        void COutput::output (int i) {
        cout << i << endl; }

class CRectangle: public CPolygon, public COutput {
public:
        int area (void)
        { return (width * height); }
```

```
};

class CTriangle: public CPolygon, public COutput {
        public:
        int area (void)
        { return (width * height / 2); }
};

int main () {
        CRectangle rect;
        CTriangle trgl;
        rect.set_values (4,5);
        trgl.set_values (4,5);
        rect.output (rect.area());
        trgl.output (trgl.area());
  system("PAUSE");
  return EXIT_SUCCESS;
}
```

Keluaran program diatas adalah :

```
20  10
```

16.10.Polymorphism

Untuk mempelajari materi polimorfisme kita harus paham dulu terhadap penggunaan pointer dan inheritance. Coba anda pahami instruksi berikut, jika anda masih belum paham pelajari kembali materi terkait:

```
int a::b(c) {};          // Materi Class
a->b                     // Materi pointer
class a: public b;       // Materi Relationships between classes
```

16.10.1. Pointers to Base Class

Salah satu keuntungan terbesar dalam menurunkan class adalah bahwa pointer ke class turunan merupakan tipe yang kompatibel dengan pointer ke class dasar. Bagian ini akan membahas kelebihan C++ dalam hal tersebut. Sebagai contoh kita akan menuliskan kembali program tentang persegi panjang dan segitiga dalam contoh sebelumnya dengan deklarasi pointer. Perhatikan contoh dibawah ini:

Program 16.31

```
#include <iostream.h>

class CPolygon {
```

```
 protected:
   int width, height;
 public:
   void set_values (int a, int b)
     { width=a; height=b; }
 };

class CRectangle: public CPolygon {
 public:
   int area (void)
     { return (width * height); }
 };

class CTriangle: public CPolygon {
 public:
   int area (void)
     { return (width * height / 2); }
 };

int main () {
 CRectangle rect;
 CTriangle trgl;
 CPolygon * ppoly1 = &rect;
 CPolygon * ppoly2 = &trgl;
 ppoly1->set_values (4,5);
 ppoly2->set_values (4,5);
 cout << "Rectangle area = "<<rect.area() << endl;
 cout << "Triangle area = "<<trgl.area() << endl;
 system("PAUSE");
 return EXIT_SUCCESS;
}
```

Keluaran program diatas adalah sebagai berikut:

```
Rectangle area = 20
Triangle area = 10
```

Fungsi main membuat dua pointer yang menunjuk ke objek dari class CPolygon, yakni *ppoly1 dan *ppoly2. Kedua pointer tersebut diinisialisasi dengan alamat objek rect dan trgl, dan karena pointer tersebut merupakan objek class turunan dari Cpolygon, maka inisialisasi ini benar. Pointer *ppoly1 dan *ppoly2 dideklarasikan dari base class. Inisialisasi dengan rect dan trgl valid, namun pointer *ppoly1 dan *ppoly2 tetap tidak dapat mengakses member CRectangle dan CTriangle.

Untuk alasan ini, ketika area() dipanggil kita tidak mungkin menggunakan pointer *ppoly1 dan *ppoly2. Untuk membuat agar *ppoly1 dan *ppoly2 dapat memanggil area(), maka harus dideklarasikan pada base classnya dan tidak hanya di class turunannya. Cara mendeklarasikannya dapat kita lihat di pembahasan materi Virtual members.

16.10.2. Virtual member

Untuk mendeklarasikan elemen sebuah kelas yang akan kita redefinisika di kelas turunan kita harus mendauluinya dengan keyword **virtual** maka pointer ke obyek yang menunjuk klas tersebut dapat digunakan dengan baik.

Program 16.32

```
#include <iostream.h>

class CSegibanyak {
 protected:
  int lebar, tinggi;
 public:
  void set_nilai (int a, int b)
    { lebar=a; tinggi=b; }
  virtual int luas (void)
    { return (0); }
 };

class CSegiempat: public CSegibanyak {
 public:
  int luas (void)
    { return (lebar * tinggi); }
 };

class CSegitiga: public CSegibanyak {
 public:
  int luas (void)
    { return (lebar * tinggi / 2); }
 };

int main () {
 CSegiempat rect;
 CSegitiga trgl;
 CSegibanyak poly;
 CSegibanyak * ppoly1 = &rect;
 CSegibanyak * ppoly2 = &trgl;
```

```
 CSegibanyak * ppoly3 = &poly;
 ppoly1->set_nilai (4,5);
 ppoly2->set_nilai (4,5);
 ppoly3->set_nilai (4,5);
 cout << ppoly1->luas() << endl;
 cout << ppoly2->luas() << endl;
 cout << ppoly3->luas() << endl;
 system("PAUSE");
 return EXIT_SUCCESS;
}
```

Keluaran program diatas adalah sebagai berikut:

```
20
10
0
```

Sekarang kita punya 3 kelas (CSegibanyak, CSegiempat dan CSegitiga) yang memiliki anggota yang sama: lebar, tinggi, set_nilai() dan luas(). Luas() telah didefinisikan sebagai virtual karena itu akan didefinisikan kembali pada kelas turunan. Anda dapat melakukan verifikasi jika anda ingin melakukannya dengan menghilangkan keyword virtual dari kode program diatas dan kemudian anda eksekusi program tersebut, hasil akan menhadi 0 untuk ketiga segibanyak bukannya 20,10,0. ini disebabkan karena pemanggilan terhadap obyek (CSegiempat::luas(), CSegitiga::luas() dan CSegibanyak::luas(), akan mengacu ke CSegibanyak::luas() dan akan memanggil fungsi yang sama tersebut (CSegibanyak::luas()) untuk semua pemanggilan melalui obyek kelas turunan karena pemanggilan-nya menggunakan pointer dengan tipe CSegibanyak. Ini menunjukan kepada anda tentang kegunaan dari keyword virtual, yang mengijinkan member/anggota dari kelas turunan dengan nama yang sama dengan member dari kelas dasar (base class) untuk dipanggil dari pointer dengan tipe kelas dasar (bas e class) tetapi melaksanakan fungsi tersebut yang didefinisikan pada kelas turunan.

Untuk memahami polymorphism terlebih lanjut harus mengenal fungsi virtual karena fungsi ini merupakan dasarnya. Fungsi anggota dari suatu class dapat dijadikan fungsi virtual. Jika fungsi ini dideklarasikan kembali pada turunan dan suatu pointer yang menunjuk class dasar diciptakan, pointer dapat memilih obyek yang tepat sekiranya fungsi anggota tersebut dipanggil via pointer. Untuk lebih memahami lihat contoh berikut:

Program 16.33

```
#include <cstdlib>
#include <iostream>
```

```
using namespace std;

class makhluk
{
  public:
    void informasi()
    {
        cout<<"Informasi pada makhluk ....."<<endl;
    }
    virtual void keterangan()
    {
        cout<<"Keterangan pada makhluk ....."<<endl;
    }
};

class mamalia : public makhluk
{
  public:
    void informasi()
    {
        cout<<"Informasi pada mamalia ....."<<endl;
    }
    virtual void keterangan()
    {
        cout<<"Keterangan pada mamalia ....."<<endl;
    }
};

class sapi : public mamalia
{
  public:
    void informasi()
    {
        cout<<"Informasi pada sapi ....."<<endl;
    }
    virtual void keterangan()
    {
        cout<<"Keterangan pada sapi ....."<<endl;
    }
};

int main(int argc, char *argv[])
```

```
{
   mamalia mamal;
   sapi lembu;
   makhluk *binatang;

   binatang = &mamal;
   binatang->informasi();
   binatang->keterangan();
   cout<<"----------------------------"<<endl;
   binatang = &lembu;
   binatang->informasi();
   binatang->keterangan();
   system("PAUSE");
   return EXIT_SUCCESS;
}
```

Keluaran program hasil eksekusi:

```
Informasi pada makhluk .....
Keterangan pada mamalia ......
-------------------------
Informasi pada makhluk ......
Keterangan pada sapi .....
Press any key to continue ...
```

Perhatikan hasil dari program di atas dan fungsi-fungsi anggota class yang dibuat yaitu makhluk, mamalia, sapi. Class mamalia mewarisi class makhluk dan class sapi mewaris class mamalia. Kemudian pada deklarasi class makhluk, perbedaan terletak pada fungsi anggota informasi() dan keterangan(), fungsi anggota informasi() tidak ada *keyword* virtual didepannya, sedangkan fungsi anggota keterangan() ada tambahan virtual. Perbedaan class mamalia dan class sapi hamper tidak ada perbedaan kecuali narasinya.

Sekarang kita lihat pada bagian main(). Mula-mula penciptaan obyek baru yaitu mamal ber-class mamalia, lembu ber-class sapi, dan binatang merupakan pointer ke class makhluk. Kemudian pointer (binatang) ini menunjuk ke obyek mamal yang ber-class mamalia. Saat fungsi anggota informasi() dipanggil dari pointer, ternyata yang ditampilkan adalah fungsi anggota dari class makhluk bukan fungsi anggota class mamalia. Tetapi tidakdemikian saat pointer memanggil fungsi anggota keterangan(), yang ditampilkan adalah fungsi anggota mamalia bukan dari fungsi anggota makhluk. Mengapa demikian ? Disinilah peranan *keyword* virtual, karena fungsi anggota keterangan() pada class makhluk diperlakukan sebagai fungsi virtual. Jadi jika perintah :

```
binatang->keterangan();
```

dijalankan akan menyebabkan fungsi anggota keterangan() milik mamalia yang diakses karena pointer sebelumnya sudah diarahkan ke obyek mamal yang ber-class mamalia. Hal serupa juga terjadi pada obyek lembu yang ber-class sapi.

Fungsi anggota yang dibuat virtual perlu dideklarasikan ulang pada setiap class turunan. Bentuk deklarasinya harus sama baik nilai balik maupun argument-argumen yang dipakai. Namun *keyword* virtual cukup dinyatakan pada class dasar, dengan sendirinya otomatis fungsi anggota tersebut pada class turunan juga bersifat virtual. Bila fungsi pada class turunan ternyata berbeda maka fungsi tersebut tidak lagi sebagai fungsi virtual, dan pada saat dijalankan kompiler akan member peringatan duplikasi nama fungsi.

Polymorphism sesuai dengan asal-usul kata pembentuknya berarti "mempunyai banyak bentuk". Dalam wujudnya, polymorphism dapat beroperasi pada dua aras, yaitu saat kompilasi dan saat eksekusi. *Overloading* terhadap fungsi dan operator merupakan bentuk polymorphism saat kompilasi. Namun orang lebih banyak mengenal polymorphism saat eksekusi atau sering disebut *late binding* atau *dynamic binding*. Hal ini menunjukkan kemampuan untuk menangani dua atau lebih bentuk obyek yang berlainan pada saat eksekusi berlangsung, dengan menyesuaikan lingkungan obyek bersangkutan. Contoh pada program sebelumnya pada fungsi informasi() dan keterangan().

Kalau diperhatikan program di atas tidak ada penciptaan obyek class makhluk, yang ada penciptaan pointer ke obyek ber-class makhluk. Pada keadaan seperti ini berarti fungsi anggota keterangan() class makhluk tidak pernah digunakan. Kalau memang tidak hendak digunakan lebih baik disingkirkan dan sebagai penggantinya ditambahkan angka 0 atau dibuat =0; lihat keterangan berikut:

```
Virtual void keterangan () = 0;
```

Fungsi inilah yang disebut dengan istilah fungsi virtual murni. Fungsi virtual murni ini sering dipakai sebagai class abstrak, yaitu class yang dideklarasikan tidak untuk menciptakan obyek. Class ini mempunyai cirri paling tidak mempunyai sebuah fungsi virtual murni. Jika class makhluk hendak dibuat class abstrak maka perubahannya sebagai berikut:

```
class makhluk
{
  public:
    void informasi()
    {
        cout<<"Informasi pada makhluk .."
       <<endl;
    }
    virtual void keterangan() = 0;
}
```

16.11.Overloading

Keistimewaan program C++ lainnya adalah adanya fasilitas *operator overloading*, yaitu penamaan fungsi tidak seperti pada umumnya, melainkan nama fungsi berupa kata kunci operator diikuti dengan lambang operator yang digunakan. Misalnya nama fungsi operator= yang dapat dipanggil dengan hanya menuliskan operator = seperti pada operasi penugasan biasa, sehingga dapat dibuat fungsi penugasan versi sendiri untuk tipe-tipe data tertentu, seperti kita dapat menyatakan Time t2 = t1, dan sebagainya. Jika kita mempunyai ekspresi dibawah ini :

```
int a, b, c;
a = b + c;
```

adalah benar, selama variable-variabel tersebut merupakan tipe data dasar. Namun tidak demikian halnya dengan contoh berikut:

```
struct {
      char product [50];
      float price; } a, b, c;
      a = b + c;
```

Tetapi C++ memungkinkan hal seperti diatas dengan menggunakan overload operator. Dibawah ini adalah operator-operator yang dapat dioverload:

```
+  -  *  /  =  <  >  +=  -=  *=  /=  <<  >>
<  <=  >  >=  ==  !=  <=  >=  ++  --  %  &  ^  !  |
~  &=  ^=  |=  &&  ||  %=  []  ()  new  delete
```

Untuk mengoverload operator kita cukup menuliskan satu fungsi anggota class dengan nama **operator** diikuti dengan operator apa yang akan dioverload. Syntaxnya sebagai berikut:

```
type operator sign (parameters);
```

Misalnya ketika akan menjumlahkan vector 2 dimensi a(3,1) and b(1,2). Hasilnya adalah sbb: (3+1,1+2) = (4,3). Untuk keperluan ini maka kita dapat mengoverload operator + seperti dalam contoh berikut :

Program 16.34

```
#include <iostream.h>

class CVector {
  public:
    int x,y;
    CVector () {};
    CVector (int,int);
    CVector operator + (CVector);
};

CVector::CVector (int a, int b) {
  x = a;
  y = b;
```

```
}

CVector CVector::operator+ (CVector param) {
 CVector temp;
 temp.x = x + param.x;
 temp.y = y + param.y;
 return (temp);
}

int main () {
 CVector a (3,1);
 CVector b (1,2);
 CVector c;
 c = a + b;
 cout << c.x << "," << c.y;
 system("PAUSE");
 return EXIT_SUCCESS;
}
```

Keluaran program diatas adalah:

```
4 , 3
```

Jika kita tidak jelas dengan banyaknya pernyataan CVector yg dipakai, perhatikan bahwa beberapa mengacu pada class CVector dan beberapa merupakan fungsi.

```
CVector (int, int);
// fungsi CVector (constructor)
CVector operator+ (CVector);
// fungsi operator+ yang return value-nya bertipe CVector
```

Fungsi operator+ dalam class CVector yang bertipe CVector tersebut digunakan untuk mengoverload operator + , sehingga pada program untuk menjumlahkan vector a ,b dan hasilnya disimpan di vector c kita dapat menggunakan salah satu instruksi berikut:

```
c = a + b;
```

```
c = a.operator+ (b);
```

Perhatikan bahwa kita juga harus mengikutkan empty constructor (tanpa parameter) yang kita definisikan dengan blok tanpa instruksi.

```
CVector () { };
```

Ini penting, karena kita mempunyai deklarasi

```
CVector temp;
CVector c:
```

Sehingga jika empty constructor tersebut dihilangkan maka dua deklarasi tersebut menjadi tidak valid. Empty constructor dapat juga dideklarasikan seperti dibawah ini :

```
CVector () { x=0; y=0; };
```

dan untuk menyederhanakan penulisan maka tidak disertakan.

Program 16.35

```
#include <cstdlib>
#include <iostream>

using namespace std;


class Time {
        friend ostream& operator << (ostream&, Time&);
                                                // untuk fungsi cout << obyek
public:
        Time(int h=0, int m=0, int s=0);        // default constructor
        Time& operator++();                     // prefix increment
        Time& operator+=(const Time&);          // operator +=
        Time& operator+=(const int);
        Time& operator+(const Time&);           // operator +
        Time& operator+(const int);
        ~Time() {cout << "Program selesai" << endl;} // destructor

private:
void format();                                  // memformat nilai obyek
        int hour;                               // agar sesuai, misalnya:
        int minute;                             // 12:65:70 ? 13:06:10
        int second;
        };
        Time::Time(int h=0, int m=0, int s=0) {
        hour = h;
        minute = m;
        second = s;
}

void Time::format() {
        int tm = minute, ts = second;
        if (second >>= 60) {
        second %= 60;
        minute += (int) (ts/60);
        tm = minute;
}
if (minute >>= 60) {
        minute %= 60;
```

```
            hour += (int) (tm/60);
}
if (hour >>= 24) hour %= 24;
}
            Time& Time::operator+= (const Time& t) {
            second = second + t.second;
            minute = minute + t.minute;
            hour = hour + t.hour;
            format;
            return *this;
}
            Time& Time::operator+= (const int t) {
            second = second + t;
            format;
            return *this;
}

Time& Time::operator++ () {
            second++;
            format;
            return *this;
}

Time& Time::operator+ (const Time& t) {
            second = second + t.second;
            minute = minute + t.minute;
            hour = hour + t.hour;
            format(hour, minute, second);
            return *this;
}

Time& Time::operator+ (const int t) {
            second = second + t;
            format(hour, minute, second);
            return *this;
}

ostream& operator << (ostream& ostr, Time& t) {
            return ostr << (t.hour < 10 ? "0" : "") << t.hour << ":"
            << (t.minute < 10 ? "0" : "") << t.minute << ":"
            << (t.second < 10 ? "0" : "") << t.second;
}
```

Untuk lebih memberikan gambaran lagi tentang penanganan suatu obyek, berikut didefenisikan suatu tipe kelas intArray untuk pengelolaan array bilangan bulat yang lebih fleksibel dari yang biasanya.

Program 16.36

```
#include <iostream.h>

const int arraySize = 20;                    //ukuran default

class intArray {

public :
        intArray (int sz = ArraySize );
        intArray ( const int * , int );
        intArray ( const IntArray & );
        ~intArray ( ) { delete [ ] ia ; }
        intArray & operator = ( const IntArray & );
        int & operator [ ] ( int );
        int getSize ( ) { return size ; }

protected :
        init ( const int *, int );
        int size;
        int *ia;
}
```

16.12. Soal Latihan

Jawablah soal latihan dibawah ini dengan baik dan benar.

1. Apa yang dimaksud dengan obyek dalam pemrograman
2. Apa yang dimaksud dengan kelas
3. Apa yang dimaksud dengan enkasulasi
4. Apa perbedaan antara class private dan public
5. Apa yang dimaksud dengan inheritance
6. Buatlah program sederhana dengan inheritance
7. Apa yang dimaksud dengan friend
8. Apa yang dimaksud dengan operator overloading
9. Apa fungsi dari class basis virtual

Lampiran 1. Daftar Kode ASCII

Decimal	Octal	Hex	Binary	Value	Comment
000	000	000	00000000	NUL	Null char
001	001	001	00000001	SOH	Start of Header
002	002	002	00000010	STX	Start of Text
003	003	003	00000011	ETX	End of Text
004	004	004	00000100	EOT	End of Transmission
005	005	005	00000101	ENQ	Enquiry
006	006	006	00000110	ACK	Acknowledgment
007	007	007	00000111	BEL	Bell
008	010	008	00001000	BS	Backspace
009	011	009	00001001	HT	Horizontal Tab
010	012	00A	00001010	LF	Line Feed
011	013	00B	00001011	VT	Vertical Tab
012	014	00C	00001100	FF	Form Feed
013	015	00D	00001101	CR	Carriage Return
014	016	00E	00001110	SO	Shift Out
015	017	00F	00001111	SI	Shift In
016	020	010	00010000	DLE	Data Link Escape
017	021	011	00010001	DC1	XON Device Control 1
018	022	012	00010010	DC2	Device Control 2
019	023	013	00010011	DC3	XOFF Device Control 3
020	024	014	00010100	DC4	Device Control 4
021	025	015	00010101	NAK	Negative Acknowledgement
022	026	016	00010110	SYN	Synchronous Idle
023	027	017	00010111	ETB	End of Trans. Block
024	030	018	00011000	CAN	Cancel
025	031	019	00011001	EM	End of Medium
026	032	01A	00011010	SUB	Substitute
027	033	01B	00011011	ESC	Escape
028	034	01C	00011100	FS	File Separator
029	035	01D	00011101	GS	Group Separator
030	036	01E	00011110	RS	Request to Send Record
031	037	01F	00011111	US	Unit Separator
032	040	020	00100000	SP	Space
033	041	021	00100001	!	
034	042	022	00100010	"	
035	043	023	00100011	#	
036	044	024	00100100	$	
037	045	025	00100101	%	
038	046	026	00100110	&	
039	047	027	00100111	'	
040	050	028	00101000	(	
041	051	029	00101001	)	

042	052	02A	00101010	*	
043	053	02B	00101011	+	
044	054	02C	00101100	,	
045	055	02D	00101101	-	
046	056	02E	00101110	.	
047	057	02F	00101111	/	
048	060	030	00110000	0	
049	061	031	00110001	1	
050	062	032	00110010	2	
051	063	033	00110011	3	
052	064	034	00110100	4	
053	065	035	00110101	5	
054	066	036	00110110	6	
055	067	037	00110111	7	
056	070	038	00111000	8	
057	071	039	00111001	9	
058	072	03A	00111010	:	
059	073	03B	00111011	;	
060	074	03C	00111100	<	
061	075	03D	00111101	=	
062	076	03E	00111110	>	
063	077	03F	00111111	?	
064	100	040	01000000	@	
065	101	041	01000001	A	
066	102	042	01000010	B	
067	103	043	01000011	C	
068	104	044	01000100	D	
069	105	045	01000101	E	
070	106	046	01000110	F	
071	107	047	01000111	G	
072	110	048	01001000	H	
073	111	049	01001001	I	
074	112	04A	01001010	J	
075	113	04B	01001011	K	
076	114	04C	01001100	L	
077	115	04D	01001101	M	
078	116	04E	01001110	N	
079	117	04F	01001111	O	
080	120	050	01010000	P	
081	121	051	01010001	Q	
082	122	052	01010010	R	
083	123	053	01010011	S	
084	124	054	01010100	T	
085	125	055	01010101	U	

086	126	056	01010110	V	
087	127	057	01010111	W	
088	130	058	01011000	X	
089	131	059	01011001	Y	
090	132	05A	01011010	Z	
091	133	05B	01011011	[	
092	134	05C	01011100	\	
093	135	05D	01011101	]	
094	136	05E	01011110	^	
095	137	05F	01011111	_	
096	140	060	01100000	`	
097	141	061	01100001	a	
098	142	062	01100010	b	
099	143	063	01100011	c	
100	144	064	01100100	d	
101	145	065	01100101	e	
102	146	066	01100110	f	
103	147	067	01100111	g	
104	150	068	01101000	h	
105	151	069	01101001	i	
106	152	06A	01101010	j	
107	153	06B	01101011	k	
108	154	06C	01101100	l	
109	155	06D	01101101	m	
110	156	06E	01101110	n	
111	157	06F	01101111	o	
112	160	070	01110000	p	
113	161	071	01110001	q	
114	162	072	01110010	r	
115	163	073	01110011	s	
116	164	074	01110100	t	
117	165	075	01110101	u	
118	166	076	01110110	v	
119	167	077	01110111	w	
120	170	078	01111000	x	
121	171	079	01111001	y	
122	172	07A	01111010	z	
123	173	07B	01111011	{	
124	174	07C	01111100	|	
125	175	07D	01111101	}	
126	176	07E	01111110	~	
127	177	07F	01111111	DEL	

Lampiran 2. Keyword C++

1	asm	insert an assembly instruction
2	auto	declare a local variable
3	bool	declare a boolean variable
4	break	break out of a loop
5	case	a block of code in a switch statement
6	catch	handles exceptions from throw
7	char	declare a character variable
8	class	declare a class
9	const	declare immutable data or functions that do not change data
10	const_cast	cast from const variables
11	continue	bypass iterations of a loop
12	default	default handler in a case statement
13	delete	make memory available
14	do	looping construct
15	double	declare a double precision floating-point variable
16	dynamic_cast	perform runtime casts
17	else	alternate case for an if statement
18	enum	create enumeration types
19	explicit	only use constructors when they exactly match
20	export	allows template definitions to be separated from their declara
21	extern	tell the compiler about variables defined elsewhere
22	false	the boolean value of false
23	float	declare a floating-point variable
24	for	looping construct
25	friend	grant non-member function access to private data
26	goto	jump to a different part of the program
27	if	execute code based off of the result of a test
28	inline	optimize calls to short functions
29	int	declare a integer variable
30	long	declare a long integer variable
31	mutable	override a const variable
32	namespace	partition the global namespace by defining a scope
33	new	allocate dynamic memory for a new variable
34	operator	create overloaded operator functions
35	private	declare private members of a class
36	protected	declare protected members of a class
37	public	declare public members of a class

38	register	request that a variable be optimized for speed
39	reinterpret_cast	change the type of a variable
40	return	return from a function
41	short	declare a short integer variable
42	signed	modify variable type declarations
43	sizeof	return the size of a variable or type
44	static	create permanent storage for a variable
45	static_cast	perform a nonpolymorphic cast
46	struct	define a new structure
47	switch	execute code based off of different possible values for a variable
48	template	create generic functions
49	this	a pointer to the current object
50	throw	throws an exception
51	true	the boolean value of true
52	try	execute code that can throw an exception
53	typedef	create a new type name from an existing type
54	typeid	describes an object
55	typename	declare a class or undefined type
56	union	a structure that assigns multiple variables to the same memory location
57	unsigned	declare an unsigned integer variable
58	using	import complete or partial namespaces into the current scope
59	virtual	create a function that can be overridden by a derived class
60	void	declare functions or data with no associated data type
61	volatile	warn the compiler about variables that can be modified unexpectedly
62	wchar_t	declare a wide-character variable
63	while	looping construct