Kata Pengantar

Bahasa pemrograman C/C++ merupakan bahasa yang popular didalam pengajaran pada computer sains maupun pada kalangan programmer yang mengembangkan system software maupun aplikasi.

Bahasa C/C++ sifatnya portable, karena compilernya tersedia hampir pada semua arsitektur computer maupun system operasi, sehingga investasi waktu dan tenaga yang anda lakukan dalam mempelajari bahasa pemrograman ini memiliki nilai strategis yang sangat menjanjikan.

Bahasa C/C++ merupakan bahasa yang sangat ketat dalam pemakaian type data maupun penulisannya yang case sensitif, hal ini berarti programmer di tuntut kedisiplinannya dalam penulisan program.

Sesuatu fasilitas yang tersedia dalam C/C++ yang tidak ditemukan pada bahasa pemrogaman lainnya adalah pointer, dengan pemanfaatan pointer programmer dapat melakukan manipulasi memori secara langsung.

Dewasa ini beberapa bahasa yang memiliki syntax penulisan yang menyerupai C/C++ adalah Java, Javascript dan PHP, yang artinya bahwa kemampuan pemrograman dengan C/C++ akan mempermudah anda untuk mempelajari bahasa modern seperti Java maupun C# (dibaca C sharp).

Akhirnya penulis mengucapkan selamat belajar dan semoga buku ini dapat memberi manfaat yang sebesarnya dalam pembelajaran mata kuliah C/C++ Programming.

Medan, 1 Maret 2004

Hendra, ST.

Website : **http://www.hendrasoewarno.com**
Email : **hendra.seowarno@gmail.com**

Daftar Isi

Section 0	History of programming language	3
Section 1	Software development and the sofware life cycle	11
Section 2	Structured programming	15
Section 3	Structure of a C++ program	24
Section 4	Comunication throught console	38
Section 5	Control structures	42
Section 6	Function (I)	53
Section 7	Function (II)	59
Section 8	Array	67
Section 9	Strings of characters	76
Section 10	Pointers	82
Section 11	Advanced pointers	89
Section 12	Dynamic memory	93
Section 13	Structures	100
Section 14	Input/output with files	109
Section 15	Classes	118
Section 16	C++ and Object oriented programming	130
Section 17	Exception handing	143
Section 18	User defined type	147

Appendix A	Preprocessor directives
Appendix B	C++ formatting and documentation
Appendix C	Programming patterns
Appendix D	Tips & Tricks

Section 0
History of programming language

Bahasa komputer telah melalui evolusi yang dramatis sejak komputer yang pertama dibuat untuk bantuan dalam kalkulasi sejak perang dunia ke dua. Awalnya programmer bekerja dengan instruksi komputer yang sangat primitif yaitu machine language (bahasa mesin).

Pada bahasa mesin instruksi-instruksi terdiri dari suatu kalimat yang panjang yang berupa tulisan 1 (satu) dan 0 (nol). Selanjutnya assembler digunakan untuk memetakan instruksi mesin ke mnemoniscs yang dapat dibaca dan ditangani oleh manusia seperti ADD dan MOV.

Kemudian, lahirlah bahasa tingkat tinggi seperti BASIC dan COBOL. Bahasa ini memungkinkan orang bekerja dengan sejumlah kata dan kalimat tertentu dalam seperti Let I=100. Instruksi ini akan diterjemahkan kembali menjadi bahasa mesin dengan mengunakan suatu interpreter ataupun compiler. Suatu interpreter bekerja dengan menterjemahkan baris program sesaat dibaca, mengubahnya menjadi bahasa mesin, dan menjalankannya.

Compiler bekerja dengan menterjemahkan program menjadi suatu bentuk intermediasi. Tahap ini disebut sebagai Compiling dan menghasilkan suatu file object. Kemudian compiler akan membangkitkan linker, yang mana akan mengubah file object menjadi suatu program yang executable (dapat dijalankan langsung).

Karena interpreter membaca kode sambil menjalankan baris per baris, maka interpreter terasa lebih mudah oleh programmer dalam menggunakannya. Dengan adanya tambahan langkah pada Compiler seperti tahap compiling dan lingking pada kode, sehingga terasa kurang nyaman.

Compiler akan menghasilkan suatu program yang sangat cepat pada saat dijalankan karena waktu untuk menterjemahkan program telah dihilangkan. Keuntungan dari suatu Compiler adalah menghasilkan program executable yang tidak tergantung kepada program sumber. Sedangkan pada interpreter anda harus menjalankannya dengan menggunakan program sumber.

Untuk beberapa tahun, prinsip utama dari programmer komputer adalah menulis suatu kode yang sependek mungkin, dan dapat dijalankan dengan cepat. Program tersebut harus berukuran kecil karena memori komputer sangat mahal, dan harus cepat karena tenaga pemrosesesan (CPU) sangat mahal.

Setelah komputer menjadi semakin kecil, murah dan cepat, dan harga dari memori telah jatuh. Prioritas ini telah berubah. Sekarang biaya tenaga programmer jauh lebih mahal dibandingkan dengan harga komputer yang digunakan dalam bisnis. Program yang ditulis dengan baik dan mudah ditangani adalah suatu yang bernilai tinggi. Mudah ditangani

artinya adalah pada saat kebutuhan bisnis berubah, program dengan mudah dapat dikembangkan tanpa adanya biaya yang sangat besar.

C programming language

C adalah bahasa pemrograman yang dikembangkan oleh Dennis Ritchie pada tahun 1970 untuk pemakaian pada sistem operasi UNIX, dan sampai sekarang bahasa ini telah dipergunakan secara praktis pada hampir semua sistem operasi. Bahasa C merupakan bahasa yang paling populer untuk menulis sistem software, dan aplikasi. serta banyak digunakan dalam pelajaran Komputer Sains.

Features

C adalah bahasa yang lebih low-level dibandingkan dengan bahasa pemrograman lainnya. Walaupun kadang-kadang disebutkan sebagai "high-level-language", sebenarnya ia hanya lebih high-level dibandingkan dengan bahasa assembly.

C memiliki dua keunggulan utama dibandingkan dengan assembly. Pertama, kodenya lebih mudah dibaca dan ditulis, terutama untuk program yang panjang. Kedua, kode assembly biasanya hanya bisa diterapkan pada arsitektur komputer yang tertentu saja, sedangkan program C dapat di pindahkan ke berbagai arsitektur dimana kalau compiler dan librarynya tersedia.

Sebaliknya efisiensi dari kode C adalah sangat bergantung pada kemampuan dari compiler untuk mengoptimisasi bahasa mesin yang dihasilkan, yang mana hal ini berada diluar kendali programmer.

Demikian juga keunggulan dan kelemahan antara C dengan bahasa high-level lainnya dimana efisiensi yang dihasilkan oleh kode C dapat lebih terkontrol, dan konsekuensinya adalah lebih sulit dibaca dan ditulis, tetapi perlu dicatat bahwa C adalah bahasa tingkat tinggi yang portabel, karena sampai saat ini hampir semua arsitektur komputer menyediakan compiler C dan librarynya.

Sesuatu fasilitas dari C yang perlu menjadi perhatian programmer adalah kemampuan dalam mengatur isi memori komputer. C Standar tidak menyediakan fasilitas array bounds checking yang dengan mudah akan menyebabkan bug dalam kaitannya dengan operasi memori, seperti buffer overflows, serta computer insecurity.

Beberapa fasilitas C adalah:

- Suatu bahasa dengan (kernel) inti yang sederhana, dimana fungsi-fungsi yang kurang penting tersedia sebagai kumpulan rutin-rutin pustaka (library) yang di standarisasi.
- Terfokus pada paradigma pemrograman procedural, dengan fasilitas pemrograman yang terstruktur.

- Memiliki suatu bahasa preprocessor
- Memiliki perfomance O(1) untuk semua operator.
- Akses secara Low-level pada memori komputer melalui pointer.
- Parameter selalu dilewatkan ke function secara by value, bukan by reference.

History

Early developments

Pengembangan C mula-mula dilakukan di laboratoriun AT&T Bell antara tahun 1969 dan 1973; menurut Ritchie, periode yang paling kreatif adalah 1972. Namanya C karena banyak fasilitasnya diambil dari bahasa sebelumnya yang disebut sebagai "B".

Pada tahun 1973, bahasa C menjadi cukup powerfull dimana biasanya kernel UNIX aslinya ditulis dengan menggunakan bahasa assembly PDP-11/20, telah ditulis kembali dengan menggunakan C. Ini merupakan pertama kali kernel suatu sistem operasi yang ditulis dengan bahasa selain assembly.

K&R C

Pada tahun 1978, Ritchie dan Brian Kerhighan mempubikasikan suatu buku edisi pertama dengan judul *The C Programming Language*. Buku ini dikenal oleh para programmer C sebagai "K&R".

K&R mengenalkan fasilitas berikut pada bahasa tersebut :

- Type data struktur (struc)
- Type data `long int`
- Type data `unsigned int`
- Operator `=+` diganti dengan `+=`, karena (`=+` membingungkan lexical analyzer Compiler C).

Pada tahun setelah publikasi K&R C, beberapa fasilitas tambahan seperti :

- `void` function dan `void *` data type
- fungsi pengembalian `struct` atau `union` type
- nama field `struct` dipisahkan dengan sebuah spasi setelah type struct.
- assignment untuk type data `struct`
- `const` qualifier untuk membuat suatu object read-only
- suatu standard library yang terdiri dari banyak fungsi yang di sediakan oleh berbagai vendor.
- enumerations
- type single-precision `float`

ANSI C and ISO C

Sejak tahun 1970, C mulai mengantikan BASIC sebagai pemimpin bahasa pemrograman mikrokomputer. Sejak 1980 mulai diadopsi pemakaiannya pada IBM PC, dan popularitasnya mulai bertambah secara nyata. Pada saat yang bersamaan Bjarne Stroustrup pada Bell Labs mulai bekerja untuk menambah konstruksi object-oriented pada C. Bahasa yang mereka produksi disebut sebagai C++, dan sekarang merupakan bahasa yang banyak dipergunakan pada sistem operasi Microsoft Windows; sedangkan C tetap merupakan bahasa yang poluler di Unix.

Pada tahun 1983, American National Standards Institute (ANSI) membentuk suatu komite, X3J11, untuk mengembangkan suatu spesifikasi standard untuk C. Melalui proses yang panjang, standard tersebut berhasil diselesaikan pada tahun 1989 (satu tahun setelah standard ANSI pertama untuk C++!) dan diratifikasikan sebagai ANSI X3.159-1989 "Programming Language C". Versi bahasa ini sering disebut sebagai ANSI C. Pada tahun 1990, ANSI C standard (dengan beberapa modifikasi kecil) telah diadopsi oleh International Standards Organization (ISO) sebagai ISO/IEC 9899:1990.

ANSI C didukung oleh kebanyakan compiler. Banyak kode C yang ditulis sekarang didasarkan pada ANSI C. Semua program yang ditulis dengan standard C dijamin akan berfungsi dengan baik pada platform lain yang memiliki C. Tetapi banyak juga program C yang hanya dapat di kompilasi pada platform tertentu dengan compiler tertentu sehubungan dengan library non standard, misalnya untuk graphic.

C99

Setelah proses standarisasi oleh ANSI, sepesikasi bahasa C masih relatif statis untuk beberapa saat, sedangkan C++ terus berevolusi. (Normalnya penyempurnaan 1 telah menghasilkan suatu versi C yang baru pada tahun 1995, tetapi versi yang ini jarang diketahui.) Sedangkan, revisi standard tahun 1990, mengawali publikasi sebagai ISO 9899:1999 pada tahun 1999. Standard ini disebut sebagai "C99" telah diadopsi sebagai ANSI standard pada 2000.

Kemampuan baru C99 meliputi:

- fungsi inline function
- membebaskan pembatasan terhadap tempat deklarasi variabel (seperti pada C++)
- menambah beberapa type data baru, termasuk `long long int` (untuk mengurangi kesulitan transisi 32-bit ke 64-bit), type data boolean, dan suatu yang baru untuk bilangan complex.
- array variable-length
- dukungan resmi terhadap one-line comment yang dimulai dengan `//`, dipinjam dari C++
- beberapa fungsi library baru, seperti `snprintf()`
- beberapa header file baru, seperti `stdint.h`

Dukungan terhadap C99 cukup beragam, dimana GCC dan beberapa compiler lainnya mendukung fasilitas C99, tetapi compiler yang dibuat oleh Microsoft dan Borland tidak.

"Hello, World!" in C

Berikut ini adalah aplikasi sederhana untuk mencetak "Hello, World!" ke suatu standard output file (yang biasanya berupa screen, tetapi bisa saja berupa suatu file atau peralatan hardware lainnya). Versi dari program ini muncul pertama kali di K&R.

```
#include <stdio.h>

int main(void)
{
    printf("Hello, World!\n");
    return 0;
}
```

Undefined behaviors

Sesuatu yang menarik (walaupun tidak selamanya persis) aspek dari standard C adalah prilaku dari beberapa kode yang disebut sebagai "undefined". Dalam praktek, hal ini berarti program yang dihasilkan dari kode ini tidak dapat diprediksi, dari (secara tidak sengaja) dimana akan membuat sistem macet setiap kali di run

Sebagai contoh, kode berikut menghasilkan prilaku undefined, karena variabel b dioperasikan lebih dari sekali dalam ekspresi `a = b + b++;`:

```
#include <stdio.h>

int main (void)
{
  int a, b = 1;
  a = b + b++;
  printf ("%d\n", a);
  return 0;
}
```

C Plus Plus

C++ (dibaca sebagai "see plus plus"; aslinya dikenal sebagai: **C with Classes**) adalah suatu bahasa pemrograman komputer yang telah diperluas dengan kemampuan object oriented. Sejak 1990, C++ merupakan bahasa pemrograman komersil yang paling populer.

History of C++

Stroustrup mulai bekerja dengan bahasa ini pada tahun 1979, diinspirasi oleh Simula67. AT&T pertama kali menggunakan bahasa tersebut pada Agustus 1983. Compiler aslinya adalah C front. Dikomersialisasikan pertama kali pada bulan Oktober. Suatu gabungan komite standarisasi C++ ANSI-ISO, pada tahun 1998 (*ISO/IEC 14882-1998*).

ANSI standard adalah suatu usaha untuk memastikan bahwa C++ adalah portable—dimana kode yang anda tulis pada Microsoft compiler akan dapat dikompilasi tanpa kesalahan dengan menggunakan kompiler dari vendor lain pada platform yang berbeda seperti Mac, Window dan Alpha.

Technical Overview

Pada tahun 1998 C++ Standard terdiri dari dua bagian yaitu: Core Language dan Standard Library; yang meliputi Standard Template Library dan C Standard Library.

History of the Name "C++"

Nama ini bermula dari Rick Mascitti (pertengahan tahun 1983) dan pertama kali digunakan pada Desember 1983. Awalnya, sejak periode riset, disebut sebagai "**C with Classes**". Nama akhirnya diambil dari C's "++" operator (yang berarti menambah nilai dari suatu variabel) dan suatu konvensi penamaan umum menggunakan "+" untuk menunjukan pengembangan program komputer, misalnya "dBase3+".

Ownership of C++

Tidak ada yang memiliki C++. Stroustrup dan AT&T tidak menerima royalti dari pemakaian C++.

C++ Examples

Contoh 1
Berikut ini adalah sebuah contoh dari program dimana tidak melakukan apa-apa. Terdiri dari satu hal yaitu: sebuah `main()` fungsi. `main()` merupakan awal dari program C++.

```
int main() {
    return 0;
}
```

Standard C++ membutuhkan `main()` mengembalikan type `int`, dalam hal ini 0 artinya program berakhir normal.

Example 2
Berikut ini adalah contoh dari program Hello world

```
#include <iostream> // needed for std::cout
int main() {
   std::cout << "Hello World !\n";
   return 0;
}
```

C++ Library

[[C++ standard library]] umumnya berupa superset dari C standard library. Sebagian besar bagian dari library C++ adalah terdiri dari Standard Template Library (STL). STL menyediakan fasilitas berguna seperti iterators (merupakan high-level dari pointer) dan containers (menyerupai array yang dapat bertumbuh secara otomatis pada saat penambahan elemen baru). Sebagaimana dalam C, fasilitas dari library dapat diakses dengan menggunakan direktif `#include` untuk memasukan suatu standard header. C++ menyediakan enam puluh sembilan header standard.

C++ merupakan superset dari C artinya semua program C yang legal juga merupakan program C++ yang legal, perkembangan dari C ke C++ adalah sangat nyata. C++ memetik banyak keuntungan dari hubungannya dengan C, dimana programmer C dapat dengan mudah menggunakan C++. Untuk dapat menggunakan keunggulan C++, banyak programmer menemukan bahwa mereka tidak perlu mempelajari hal-hal yang mendasar, tetapi cukup mengembangkan diri dengan mempelajari konsep baru (OOP) dalam pemecahan masalah pemrograman.

Future Development

C++ terus berevolusi untuk memenuhi kebutuhan masa depan, ketika pembuat-pembuat compiler masih berjuang untuk mendukung semua fasilitas C++, situasi tersebut berkembang persisnya dari tahun 1998 sampai 2003.

Pertanyaan

1. Tuliskan pengertian tentang High Level Language dan Low Level Language.
2. Berikan contoh bahasa komputer yang tergolong kepada High Level Language dan Low Level Language.
3. Tuliskan perbedaan cara kerja Interpreter dan Compiler.

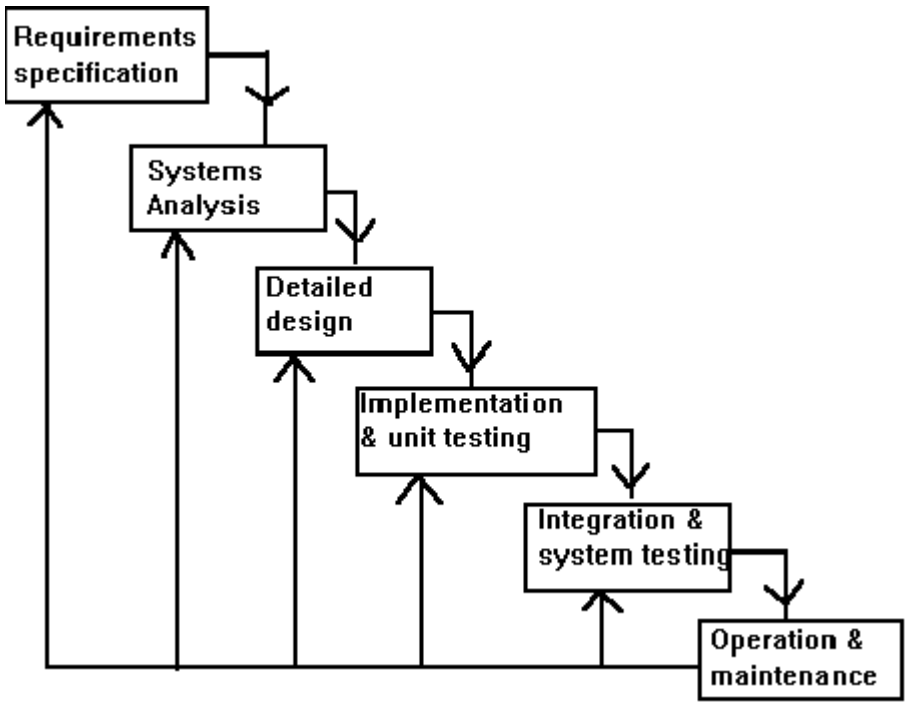
4. Tuliskan pengertian bahasa C adalah bahasa yang portabel.
5. Sebutkan tujuan penetapan standard C oleh ANSI.
6. Tuliskan pengertian bahasa C++ merupakan superset dari bahasa C.
7. Tuliskan struktur program C dan C++ yang paling sederhana.
8. Dengan pemanfaatan fasilitas search engine di internet, carilah jenis-jenis compiler C/C++ yang tersedia oleh berbagai vendor, dan platform.

Latihan

1. Buatlah program C anda yang pertama untuk mencetak "Hello World !" ke layar, dan bagaimana melakulannya dalam C++.
2. Tulis suatu program C++ yang mana membaca dua nilai berjenis float, dan mencetak hasil sum (penjumlahan), product (perkalian) dan quotient (hasil bagi). Tambahkan informasi hasil pencetakan yang jelas.

Section 1
Software Development and the Software Life Cycle

Banyak model telah dikembangkan para ahli untuk menggambarkan proses dari pengembangan software. Suatu pengetahuan akan software life cycle akan membuat pengembangan software yang terstruktur, mudah dikomunikasikan dan mudah diawasi. Suatu model waterfall (air terjun) ditunjukkan oleh gambar dibawah ini mengalir mulai dari Requirements specification, System analyst, Detailed design, Implementation & unit testing, Integration & system testing, dan Operation & maintenance. Software developement dapat memiliki bagian yang sama dengan software life cycle kecuali pada tahapan akhir.



Requirements specification (spesifikasi kebutuhan) mengacu pada layanan apa saja yang harus disediakan oleh software dan di berada dibawah atasan apa sotfware tersebut dioperasikan. Kebutuhan yang akurat sering kali sulit ditentukan karena bidang komputer sangat berkaitan dengan kompleksnya sosial manusia dan dunia secara organisasi.

Proses dari **systems analysis and design** (analisa dan perancangan sistem) akan menentukan fungsi-fungsi akurat yang dibutuhkan untuk memenuhi kebutuhan dan hardware (perangkat keras) yang dibutuhkan untuk mengoperasionalkannya. Hal ini mungkin meliputi sub system yang tidak berkaitan dengan komputer-komputer, bersama dengan suatu definisi lengkap dari (human role) peranan manusia. Bayangkan contoh sebuah aircraft system (system angkutan udara) : disana akan terdiri dari sejumlah komputer monitoring (pemantau) dan controlling (pengendali) aspek dari penerbangan dengan banyak alat display (peraga), dan berinteraksi dengan ground systems (sistem di daratan); peran dari pilot juga menentukan.

Detailed design akan berkaitan dengan pemecahan system atau subsystem kedalam modul-modul definisi yang lebih jelas. Perulangan subdivision (pembagian kepada bagian-bagian) atau secara fungsi akan penting sampai suatu keterangan yang jelas dapat dipenuhi oleh modul-modul yang mana menjadi blok bangunan dari system. Kita akan melihat pada beberapa teknik untuk mengambarkan detail dari program pada level ini.

Implementation and testing berkaitan dengan penulisan kode untuk menghasilkan suatu program yang dapat dijalankan untuk pada modul (penerapan) dan percobaan pada program untuk memastikan program dapat bekerja. Hal ini biasanya berkaitan dengan perbaikan pada kode (debugging) sampai program berjalan dengan benar. Ini adalah area pekerjaan programmer.

Integration and testing berkaitan dengan membawa program kedalam subsystem, dengan dengan berbagai interface (antar muka) antar program, dan demikian juga subsystem kedalam system. Testing dibutuhkan untuk memastikan bahwa program-program berjalan dengan benar ketika mereka digabungkan, dan fungsi dari system benar. Pada saat testing ditemukan kesalahan, rancangan ulang atau pengkodean ulang mungkin dibutuhkan.

Ketika software tersebut telah sempurna haruslah diinstalasi dan pemakai harus dilatih. Sebagaimana ketika digunakan pada lingkungan target kesalahan baru mungkin akan timbul. Juga oleh waktu kebutuhan tersebut dapat berubah. Jika perangkat lunak telah memasuki tahap ini disebut sebagai **maintenance** (pemeliharaan).

Pada saat yang sama siklus manfaat dari software telah berlalu dan suatu system baru dibutuhkan.

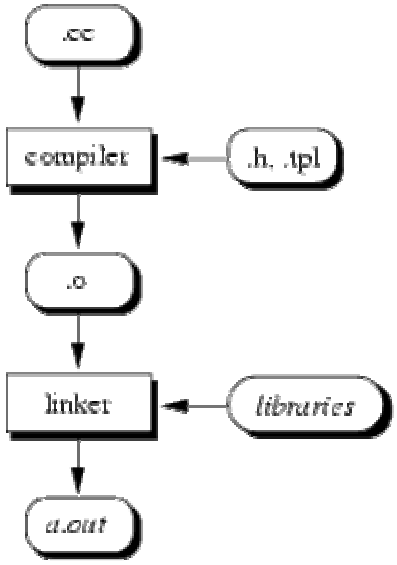
Preparing a Computer Program

Ada beberapa langkah untuk membuat suatu program komputer untuk aplikasi tertentu. Langkah-langkah ini tidak tergantung pada komputer ataupun bahasa pemrograman yang digunakan, langkah-langkahnya adalah sebagai berikut:

1. Pelajari **requirement specification** (spesifikasi kebutuhan) dari aplikasi tersebut. Adalah sangat penting kebutuhan dari aplikasi tersebut dapat ditentukan dengan benar. Sebelum mulai merancang program untuk aplikasi tersebut, adalah perlu bahwa requirement specification telah lengkap dan konsisten. Sebagai contoh suatu requirement specification menyebutkan `buat sebuah program untuk menghitung sebuah persamaan' adalah belum lengkap, anda harus bertanya untuk informasi selanjutnya seperti `Jenis persamaan?', `berapa banyak persamaan?', `sampai berapa akurat?' dll.
2. Analisa masalah tersebut dan putuskan bagaimana untuk memecahkannya. Pada tahap ini anda harus menentukan suatu metode agar masalah tersebut dapat dipecahkan, metode tersebut sering disebut sebagai **Algorithm**.
3. Terjemahkan algoritma tersebut menjadi suatu program dalam dengan high-level language. Bentuk tertulis dari program disebut sebagai **source program** atau

source code. Pada tahap ini **desk-check** untuk memeriksa kebenarannya, dan diperbaiki, pada tahap ini anda membutuhkan **Editor**.

4. **Compile** program tersebut menjadi machine-language. Bahasa mesin yang dihasilkan disebut sebagai **object code**. Pada tahap ini compiler akan mencari **Syntax** errors dalam program.
5. Object code yang dihasilkan oleh compiler akan di link dengan berbagai function library disediakan oleh system. Pada tahap ini diperlukan suatu program **linker** yang akan mengabung object code yang dimuat kememori oleh sebuah program yang disebut sebagai **loader**.
6. Jalankan hasil link, dan periksalah apakah adanya Logical errors. Kesalahan lain yang dapat saja terjadi adalah **run-time error**.



Gambar 1.4 Tahapan kompilasi

Jika ada kesalahan pada tahap ini, maka program harus diperbaiki kembali mulai dari langkah (3), (4) dan (5), sampai hasilnya memuaskan.

7. Sekarang program telah dapat digunakan, dan perlu adanya **documentation** untuk keperluan perbaikan dikemudian hari

Pertanyaan

1. Tuliskan langkah-langkah yang harus dilakukan mulai dari spesifikasi permasalahan untuk menghasilkan suatu program yang dapat berjalan untuk pemecahan masalah tersebut.
2. Jenis kesalahan apa saja yang dapat timbul dalam tahapan produksi program ?
3. Ketika program sedang berjalan dan menghasilkan suatu bilangan yang terlalu besar untuk ditempatkan pada tempat yang telah dialokasi dalam memori, sebutkan kategori jenis kesalahan tersebut?

4. Sebuah program yang berjalan tanpa kesalahan, tetapi menghasilkan output yang tidak benar, jenis kesalahan apa yang menyebabkan hal tersebut?
5. Sebutkan pengertian dari source code, object code dan executable code.
6. Tuliskan pengertian tentang istilah Editor, Compile, Linker dan Libraries.

Section 2

Structured programming

Secara garis besar, kita dapat mengidentifikasi kurva belajar dari seseorang yang mempelajari pemrograman adalah sebagai berikut :

- Pemrograman tidak terstruktur
- Pemrograman procedural
- Pemrograman modular dan
- Pemrograman berorientasi objek

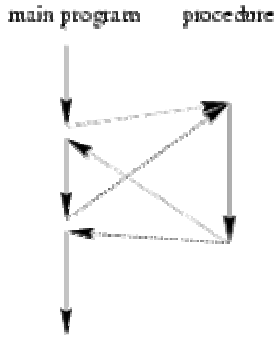
Unstructured programming

Biasanya orang mulai belajar pemrograman dengan menulis sebuah program kecil dan sederhana yang terdiri dari hanya satu program utama, pada program utama ini rangkaian perintah-perintah yang secara langsung melakukan modifikasi data global dalam program.

Teknik pemrograman ini memiliki berbagai kelemahan, terutama ketika program tersebut berkembang menjadi lebih besar. Sebagai contoh, jika sejumlah perintah yang sama dibutuhkan pada beberapa bagian program yang sama, maka perintah-perintah tersebut harus diduplikasi ke bagian yang membutuhkannya. Hal ini menimbulkan suatu ide bagaimana kalau kita keluarkan rangkaian perintah tersebut menjadi suatu program kecil (dikenal sebagai istilah procedure) yang memiliki nama, kemudian program kecil tersebut akan dipanggil oleh bagian program yang membutuhkannya.

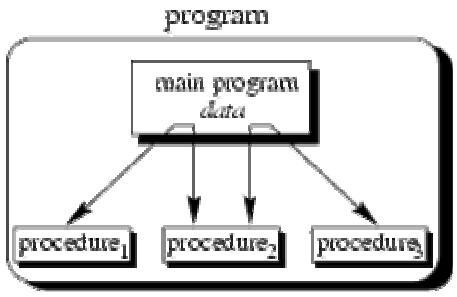
Procedural programming

Pada pemrograman prosedural, anda dapat mengelompokan rangkaian perintah-perintah yang sering digunakan pada satu tempat menjadi suatu procedure, kemudian suatu procedure call dapat digunakan untuk membangkitkan procedure tersebut. Setelah procedure tersebut diproses, aliran kendali proses akan dikembalikan ke posisi baris perintah setelah perintah call (pemanggilan).



Salah satu keunggulan dari pemrograman prosedural adalah program menjadi lebih terstruktur dan tingkat kesalahan dapat ditekan. Sebagai contoh, ketika suatu procedure telah benar, maka setiap kali procedure tersebut digunakan akan tetap menghasilkan hasil yang benar. Konsekuensinya adalah ketika kita melakukan pencarian kesalahan, maka area pencarian dapat dipersempit ke daerah yang belum terbukti kebenarannya.

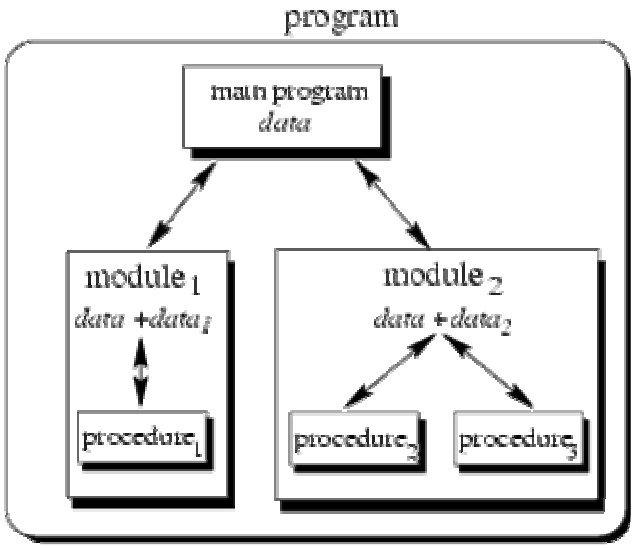
Pada pemrograman prosedural, program dapat dilihat sebagai rangkaian dari procedure calls. Program utama akan melewatkan data ke procedure, dan data tersebut akan diproses oleh procedure, dan setelah prosedure selesai, data hasil telah tersedia. Selanjutnya aliran dari data dapat diilustrasikan sebagai suatu hierarchical graph (grafik hirarki), suatu *tree (pohon)*, sebagaimana yang ditunjukan gambar berikut :



Sekarang kita telah memiliki suatu program yang mana terbagi atas potongan-potongan kecil yang disebut sebagai procedure

Modular programming

Pada pemrograman modular, Procedure dengan fungsi yang umum dikelompokan menjadi suatu modul secara terpisah, sehingga suatu program tidak hanya terdiri dari satu bagian, tetapi terpisah menjadi beberapa bagian yang lebih kecil secara terpisah yang berinteraksi satu sama yang lain dengan melalui procedure call yang membentuk keseluruhan program.



Masing-masing modul memiliki data tersendiri. Hal ini memungkinkan masing-masing modul mengatur status internal yang mana dimodifikasi oleh pemanggilan procedure dalam modul tersebut.

Structured programming

Pemrograman terstruktur dapat dipandang sebagai bagian atau sub-disiplin dari procedural programming, satu dari paradigma besar (dan mungkin yang paling popular) untuk pemrograman komputer.

Berdasarkan sejarah, beberapa teknik terstruktur telah dikembangkan untuk menulis program yang terstruktur. Dua diantaranya yang paling umum adalah Jackson Structured Programming, yang mana didasarkan pada pemerataan struktur dengan struktur program. dan Dijkstra's structured programming, yang mana didasarkan pada pemisahan program kedalam sub-sections, dimana masing-masing terdiri dari satu titik masuk dan satu titik keluar

Sejak 1970 ketika pemrograman terstruktur mulai populer sebagai suatu teknik pemrograman, banyak bahasa pemrograman procedural yang menambah fasilitas untuk mendukung pemrograman terstruktur, (dan beberapa diantaranya telah meninggalkan fasilitas yang dapat membuat pemrograman menjadi tidak terstruktur). Bahasa pemrograman terstruktur yang terkenal baik adalah

- Pascal
- Ada

Pada program yang pendek, pemrograman terstruktur secara sederhana menekankan pada hirarki struktur aliran program. Hal ini dapat dilakukan pada hampir semua bahasa modern dengan menggunakan konstruksi looping terstruktur seperti "while", "repeat",

"for". Juga disarankan agar setiap looping hanya ada satu titik masuk dan satu titik keluar, dan pada beberapa bahasa telah menekankan hal ini.

Pada pemrograman terstruktur, programmer harus memecahkan kode yang besar kedalam sub rutin yang lebih pendek sebagai kesatuan logika yang lebih kecil (procedure dan function), sehingga dapat dimengerti secara mudah. Pada pemrograman terstruktur diusahakan untuk menggunakan variabel global sedikit mungkin; dan sebagai gantinya, sub rutin harus menggunakan variabel lokal serta menggunakan argumen (parameter) sebagai referensi data diantaranya.

Catatan:
Pada saat program anda menjadi besar dan kompleks, adalah semakin susah untuk mencari bugs (kesalahan) yang kemungkinan terkandung didalamnya. Pemakaian variabel global yang terlalu banyak akan mempersulit penanganan bugs, karena data global dapat diubah oleh seluruh fungsi/prosedur dalam program. Berdasarkan pengalaman bertahun-tahun dianjurkan kepada programmer untuk menangani data secara lokal sebisa mungkin

Pemrograman terstruktur sering (tidak selamanya) berkaitan dengan perancangan dengan pendekatan "top-down". Dengan cara ini perancang memetakan garis besar struktur program kedalam operasi-operasi yang lebih kecil, menerapkan dan mencoba operasi yang lebih kecil, dan mengikat mereka kedalam suatu program secara keseluruhan.

Pada akhir abad ke 20, umumnya programmer professional menganut konsep pemrograman prosedural terstruktur. Beberapa diantaranya menyatakan lebih mudah untuk mengerti program terstruktur, lebih reliabel dan lebih mudah di tangani (maintanance).

Bahasa tidak terstruktur mendefinisikan aliran program dengan pemakaian perintah GOTO yang mana memindahkan titik eksekusi ke suatu label tertentu dalam kode, sedangkan bahasa pemrograman terstruktur menyediakan konstruksi (sering disebut sebagai "if-then-else", "unless", "while", "until", dan "for") untuk menciptakan berbagai variasi loop (perulangan) dan percabangan kondisi.

Perlu ditekankan bahwa pada suatu bahasa pemrograman terstruktur, di setiap struktur kode harus memiliki satu titik masuk dan satu titik keluar, banyak bahasa seperti C memungkinkan banyak cara untuk keluar dari suatu struktur (seperti "continue", "break", dan "return"), yang mana dapat memberikan keuntungan dan kerugian dalam pembacaan dan penulisan program.

Top down design

Dalam merancang program secara **top-down structured**, langkah pertama adalah kenalilah komponen-komponen utama yang membentuk solusi pemecahan masalah, kemudian komponen-komponen tersebut disusun berurutan menjadi langkah-langkah

pemecahan solusi, dan kemudian lanjutkan dengan mengembangkan detail dari masing-masing komponen tersebut dengan cara yang sama, dan seterusnya.

Misalnya kita akan membuat sebuah program untuk menghitung luas (area) dan keliling (perimeter) dari suatu persegi panjang (rectangle), dimana panjang (length) dan lebar (width) dimasukan akan oleh pamakai. Suatu algoritma awal yang mungkin adalah sebagai berikut :

```
1. Masukan panjang dan lebar dari persegi panjang.

2. Hitung luas dan keliling dari persegi panjang.

3. Cetak luas dan keliling.
```

Selanjutnya masing-masing garis besar tersebut diatas dapat di kembangkan menjadi langkah-langkah yang lebih rinci sebagai berikut :

```
1. Masukkan
   1.1 Tampilkan pesan untuk memasukan panjang
   1.2 Terima data panjang
   1.3 Tampilkan pesan untuk memasukan lebar
   1.4 Terima data lebar

2. Menghitung
   2.1 Menghitung keliling sebagai dua kali panjang tambah lebar
   2.2 Menghitung luas sebagai perkalian panjang dan lebar

3. Keluaran
   3.1 Cetak keliling
   3.2 Cetak luas
```

Sampai pada tahap ini, masalah tersebut telah dipecahkan secara sempurna tanpa tergantung pada bahasa pemrograman yang akan digunakan.

Structured Design. Top-down design of algorithms using sequence and selection only.

Perhatikan masalah berikut:

> Buatlah sebuah program yang menerima jumlah jam kerja dalam satu minggu dan upah perjam untuk seorang karyawan. Program harus mampu mencetak upah mingguan (wage) dari karyawan tersebut. Karyawan dibayar dengan tarif normal untuk 40 (empat puluh) jam pertama, dan satu setengah dari tarif normal untuk setiap kelebihannya.

Solusi untuk masalah diatas dapat langsung dimodelkan sebagai berikut :

```
1. Masukan jumlah jam kerja dan upah per-jam.
```

```
2. Hitung upah mingguan.

3. Cetak upah mingguan.
```

Tetapi usaha untuk mengembangkan langkah 2, tidak sederhana seperti mengalikan jam kerja dengan upah per-jam, tetapi kita perlu memeriksa apakah jam kerja yang dimasukan tidak lebih dari 40 jam. Sehingga dalam mengembangkannya kita perlu menggunakan pernyataan kondisi seperti berikut ini :

```
1. Masukan
   1.1 Tanyakan user untuk memasukan jumlah jam kerja.
   1.2 Baca jumlah jam kerja.
   1.3 Tanyakan user untuk memasukan upah per-jam.
   1.4 Baca upah per-jam.

2. Hitung upah mingguan
   2.1 Jika jumlah kerja lebih atau sama dengan 40 jam
       kemudian
          2.1.1 hitung upah mingguan normal.
       otherwise
          2.1.2 hitung upah mingguan normal dan kelebihannya.
3. Cetak upah mingguan
```

Structured Design. Top-down design of algorithms using repetition

Algoritma berikut mengilustrasikan berbagai teknik. Dimana program akan menerima sejumlah data positif dan akan menampilkan nilai minimum, nilai maximum dan rata-rata. Program akan terus menanyakan data sampai pemakai memasukan suatu nilai negatif.

Adapun spesifikasi program tersebut adalah sebagai berikut:

> Menampilkan pesan agar user memasukan nilai real positif dari keyboard, dan menampilkan informasi untuk mengakhiri proses input dengan nilai negatif. Sesaat proses input selesai, program akan menampilkan nilai positif minimum dan nilai positif maximum serta nilai rata-rata dari nilai positif yang dimasukan. Jika tidak ada data (pemakai langsung memasukan nilai negatif pada awal program) kemudian program harus menampilkan pesan tidak ada data yang dimasukan.

Berikut ini adalah langkah-langkah solusi pemecahannya :

```
tahap awal.
terima nilai pertama.
ketika (nilai adalah positif)
   kerjakan
   {
    proses nilai.
```

```
    terima nilai berikutnya.
   }
jika tidak ada nilai yang dimasukan
   kemudian cetak `tidak ada data yang dimasukan'
   jika tidak
      {
       hitung rata-rata.
       cetak hasilnya.
      }
```

Berikutnya kita akan mengembangkan bagian proses nilai:

```
proses nilai:
  tambahkan nilai ke akumulasi jumlah.
  tambahkan satu pada jumlah.
  jika nilai lebih besar dari maximum yang tersimpan
     kemudian
     simpan nilai sebagai maximum.
  jika nilai lebih kecil dari minimum yang tersimpan
     kemudian
     simpan nilai sebagai minimum.
```

Berdasarkan ekspansi proses nilai diatas, maka hal yang perlu dilakukan sebelum masuk ke perulangan adalah:

1. sebuah variabel untuk akumulasi jumlah – harus dimulai dari (0) nol.
2. sebuah variabel untuk jumlah data – harus dimulai dari (0) nol.
3. variabel untuk menyimpan maximum dan minimum – pada awalnya berisi nilai yang pertama kali dimasukkan.

Selanjutnya bagian tahap awal dari program dapat dikembangkan menjadi:

```
set nol ke akumulasi.
set nol ke jumlah.
terima nilai yang pertama.
set minimum dan maksimum dengan nilai pertama.
```

Jika tidak ada data yang dimasukan, maka hal ini dapat diketahui dengan nilai jumlah yang berisi 0. Sehingga secara keseluruhan solusi menjadi:

```
set nol ke akumulasi.
set nol ke jumlah.
terima nilai yang pertama.
set minimum dan maksimum dengan nilai pertama.
terima nilai pertama.
ketika (nilai adalah positif)
   kerjakan
   {
    tambahkan nilai ke akumulasi.
    tambahkan satu ke jumlah.
    tambahkan nilai ke akumulasi jumlah.
    tambahkan satu pada jumlah.
```

```
    jika nilai lebih besar dari maximum yang tersimpan
         kemudian
         simpan nilai sebagai maximum.
    jika nilai lebih kecil dari minimum yang tersimpan
         kemudian
         simpan nilai sebagai minimum.
    terima nilai berikutnya.
   }
jika tidak ada nilai yang dimasukan
   kemudian cetak `tidak ada data yang dimasukan'
   jika tidak
      {
       hitung rata-rata.
       cetak hasilnya.
      }
```

Top-down design using Functions

Pada contoh kasus sebelumnya kita telah melihat pemecahan masalah untuk perhitungan upah mingguan berdasarkan jumlah jam kerja dan upah per-jam. Misalnya kita kembangkan program tersebut menjadi dapat menerima data untuk beberapa karyawan dan mencetak upah mingguan. Algoritma yang sesuai untuk masalah tersebut adalah:

```
ulangi
  terima jumlah jam kerja dan upah per-jam.
  proses upah mingguan.
  tanyakan user `ada data yang lain?'
  baca jawaban
sampai jawaban tidak
```

Selanjutnya kita akan memodulkan proses upah mingguan menjadi suatu sub program yang lebih kecil, sehingga algoritma dapat ditulis menjadi :

```
ulangi
  terima jumlah jam kerja dan upah per-jam.
  Panggil function hitungUpah(jam kerja, upah per-jam, upah mingguan).
  cetak upah mingguan.
  tanyakan user `ada data yang lain?'
  baca jawaban
sampai jawaban tidak
```

Pertanyaan

1. Sebutkan tahapan kurva belajar seseorang yang mempelajari teknik pemrograman.
2. Sebutkan garis besar ide dasar dari masing-masing teknik pemrogram tersebut pada soal nomor 1.
3. Sebutkan fasilitas yang biasanya disediakan oleh suatu bahasa pemrograman yang menyediakan fasilitas pemrograman terstruktur, dan berikan pendapat anda tentang pemakaian GOTO.

4. Sebutkan ide dasar dari teknik pemrograman terstruktur dengan pendekatan Top Down Design.

Latihan

Rancanglah sebuah program dengan pendekatan terstruktur top-down untuk pemecahan masalah berikut :

1. Buatlah sebuah program untuk menghitung jumlah konsumsi bahan bakar suatu mobil, dengan input kilometer awal dan akhir perjalanan, isi awal dan akhir tangki dalam liter, kemudian program mencetak jarak tempuh, jumlah konsumsi, dan konsumsi per kilometer.
2. Modifikasi program diatas dimana input tetap dalam kilometer dan liter, tetapi menghasilkan perhitungan dalam gallon dan mil. (gunakan konstanta)
3. Buatlah sebuah program yang mana menerima jumlah jam kerja dalam minggu dan upah perjam seorang tenaga kerja. Program tersebut harus menghitung penghasilan dari pekerja tesebut. Pekerja tersebut dibayar secara normal untu 40 jam pertama, dan satu setengah kali upah normal untuk kelebihan jam berikutnya.

Petunjuk

1 US gallon = 3.7854118 litres, 1 mile = 1.6093 kilometres

Section 3
Structure of a C++ program

Sebagaimana tradisi dalam belajar bahasa komputer adalah dimulai dengan membuat program Hello World, perhatikan koding berikut :

```
// my first program in C++

#include <iostream.h>

int main ()
{
  cout << "Hello World!";
  return 0;

}
```

Simpan program anda ke file dengan nama “first.cpp”, lakukan kompilasi (dalam hal ini menggunakan Borland free C++ compiler, dapat di download pada http://www.borland.com)

BCC32 First

Jalankan program anda :

First

Dan di layar akan tercetak tulisan "Hello World!", Marilah kita analisa bagian program tersebut satu per satu.

`// my first program in C++`
: Ini adalah baris comment (komentar/keterangan). Semua baris yang dimulai dengan tanda (//) dianggap sebagai comment.

`#include <iostream.h>`
: Perintah yang dimulai dengan tanda (#) adalah directives (petunjuk) untuk preprocessor. Baris ini tidak akan dieksekusi, tapi merupakan directives bagi compiler. Dalam hal ini kalimat **`#include <iostream.h>`** memberitahukan kepada preprocessor compiler untuk include (memasukan) header file standar **iostream**. File ini berisi deklarasi input-output standar library didalam C++ yang diperlukan pada bagian berikutnya dalam program ini.

`int main ()`
: Baris ini merupakan deklarasi **`main`** function (fungsi utama). **`Main`** function merupakan titik dimana dimana program akan mulai dijalankan.

  `main` selalu diikuti oleh tanda kurung **`()`** karena merupakan sebuah function. Pada C++ semua function. Isi dari function **`main`** diapit dengan tanda kurawal (**`{}`**), sebagai berikut :

`cout << "Hello World";`

`Cout` merupakan standard output stream pada C++ (biasanya ke layar), yang dalam hal ini akan mencetak tulisan "Hello World". **`Cout`** dideklarasikan dalam header file `iostream.h`, sehingga untuk memanfaatkannya perlu di #include.

Catatan : setiap perintah C++ diakhiri dengan karakter semicolon (`;`).

`return 0;`

Perintah **`return`** menyebabkan selesainya function **`main()`** dan mengembalikan suatu nilai, dalam hal ini **`0`**. Dalam hal ini berarti eksekusi selesai tanpa kesalahan.

Comments (Komentar/keterangan).

Ketika anda sedang menulis sebuah program, segalanya adalah jelas tentang pada yang sedang anda kerjakan. Tetapi setelah satu bulan, anda kembali melihat program tersebut, mungkin saja anda telah lupa dan merasa bingung.

Untuk mengurangi keraguan tersebut, anda perlu membuat komentar secukupnya pada kode anda.

Komentar adalah bagian dari source code yang akan diabaikan oleh compiler.

Ada dua cara penulisan komentar pada C++:

```
// line comment
/* block comment */
```

Yang pertama adalah mengawali setiap baris komentar dengan tanda (**`//`**), sedangkan bentuk kedua mengawali komentar dengan **`/*`** dan diakhiri dengan **`*/`** , jenis komentar ini cocok untuk komentar/keterangan yang lebih dari satu baris.

Perhatikan contoh berikut :

```
/* my second program in C++
   with more comments */

#include <iostream.h>

int main ()
{
  cout << "Hello World! ";     // says Hello World!
  cout << "I'm a C++ program"; // says I'm a C++ program
  return 0;

}
```

Catatan:

Adalah praktek yang baik untuk senantiasa membuat komentar pada awal dari setiap program yang anda buat. Mengenai bentuk komentar tersebut diserahkan kepada masing-masing individu, tetapi setiap header program harus memiliki informasi berikut ini:

- Nama dari fungsi atau program
- Nama file
- Apa yang dikerjakan oleh fungsi atau program
- Keterangan tentang bagaimana program bekerja
- Nama pengarang
- Sejarah revisi (catatan pada setiap perubahan dilakukan)
- Jenis kompiler, linker, dan tools yang digunakan untuk membuat program tersebut.
- Catatan tambahan secukupnya.

Berikut ini adalah contoh komentar yang mungkin pada program Hello World :

```
/*************************************************************
Program:      Hello World
File:         Hello.cpp
Function:     Main (complete program listing in this file)
Description:  Prints the words "Hello world" to the screen
Author:       Jesse Liberty (jl)
Environment:  Turbo C++ version 4, 486/66 32mb RAM, Windows 3.1
              DOS 6.0.  EasyWin module.
Notes:        This is an introductory, sample program.
Revisions:    1.00  10/1/94 (jl) First release
              1.01  10/2/94 (jl) Capitalized "World"
*************************************************************/
```

Variables. Data types. Constants.

Coba bayangkan kalau saya meminta anda untuk mengingat angka 8, dan angka 2, kemudian saya meminta anda untuk menjumlahkan angka pertama dengan 1, sehingga menjadi 9 (8+1), dan akhirnya hasilnya dikurangi dengan bilangan kedua (2).

Proses diatas secara komputer dapat ditulis sebagai berikut :

```
a = 8;
b = 2;
a = a + 1;
result = a - b;
```

dalam hal ini a dan b adalah variable, jadi variable di program komputer menyerupai variable pada matematika.

Sehingga, kita dapat mendefinsikan variable sebagai bagian dari memori untuk menyimpan nilai, dalam hal ini **a** dan **b** disebut sebagai **identifier** (pengenal)

Identifiers

Suatu identifier harus memenuhi syarat berikut :

1. Terdiri dari huruf, angka dan symbol garis bawah (_).
2. Untuk compiler tertentu maksimum 32 karakter.
3. Tidak boleh pakai spasi atau tanda baca lainnya .
4. Boleh diawali dengan symbol garis bawah (_), tetapi biasanya dicadangkan untuk external link.
5. Tidak boleh sama dengan **keyword** (kata kunci)

Key word standar menurut ANSI-C++ yang tidak boleh anda gunakan sebagai identifier:

```
asm, auto, bool, break, 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
```

Operator lainnya yang tidak boleh digunakan sebagai identifier:

```
and, and_eq, bitand, bitor, compl, not, not_eq, or, or_eq,
xor, xor_eq
```

Sangat penting diingat: Bahasa C++ language adalah "case sensitive", hal ini berarti bahwa identifier yang ditulis dengan huruf besar dan kecil adalah tidak sama. Jadi, variable `HASIL tidak sama dengan variable` **`hasil`** ataupun variable **`Hasil`**.

Data types

Dalam pemrograman, kita menggunakan variable sebagai fasilitas untuk menyimpan dan mengambil nilai di memori komputer.

Data pada memory komputer diorganisasikan dalam bentuk byte (8 bit) 0 s/d 255, dan oleh bahasa pemrograman C++ data tersebut dapat dinyatakan menjadi berbagai data type sebagai berikut :

DATA TYPES

Name	Bytes*	Description	Range*
char	1	character or integer 8 bits length.	**signed:** -128 to 127 **unsigned:** 0 to 255
short	2	integer 16 bits length.	**signed:** -32768 to 32767 **unsigned:** 0 to 65535
long	4	integer 32 bits length.	**signed:**-2147483648 to 2147483647

			unsigned: 0 to 4294967295
int	*	Integer. Its length traditionally depends on the length of the system's Word **type**, thus in MSDOS it is 16 bits long, whereas in 32 bit systems (like Windows 9x/2000/NT and systems that work under protected mode in x86 systems) it is 32 bits long (4 bytes).	See **short**, **long**
float	4	floating point number.	3.4e + / - 38 (7 digits)
double	8	double precision floating point number.	1.7e + / - 308 (15 digits)
long double	10	long double precision floating point number.	1.2e + / - 4932 (19 digits)
bool	1	Boolean value. It can take one of two values: true or false NOTE: this is a type recently added by the ANSI-C++ standard. Not all compilers support it. Consult section bool type for compatibility information.	**true** or **false**
wchar_t	2	Wide character. It is designed as a type to store international characters of a two-byte character set. NOTE: this is a type recently added by the ANSI-C++ standard. Not all compilers support it.	wide characters

Declaration of variables

Pada C++ suatu variabel adalah tempat untuk menyimpan informasi. Suatu variabel adalah suatu lokasi pada memori komputer anda yang mana dapat menyimpan sebuah nilai dan dari sana anda dapat mengambil kembali nilai yang tersimpan.

Sebagaimana dengan Pascal, setiap variable dalam C++, kita harus mendeklarasikannya terlebih dahulu.

Adapun tata cara pendeklarasian variabel pada C++ adalah menuliskan type data yang diinginkan (misalnya **`int`**, **`short`**, **`float`**...) kemudian diikuti dengan identifier variable yang sah (lihat bagian identifier). Sebagai contoh:

```
int a;
float mynumber;
```

Dapat juga mendeklarasikan beberapa variable dengan data type yang sama :

```
int a, b, c;
```

Khusus variable type Integer (**`char`**, **`short`**, **`long`** dan **`int`**) dapat ditentukan apakah signed dan unsigned (defaultnya adalah **signed**). Sebagai contoh:

```
unsigned short NumberOfSons;
signed int MyAccountBalance;
```

Catatan: Kebanyakan programmer yang berpengalaman menggunakan notasi Hungarian untuk identifier variabel yang digunakan.

Adapun dasar ide notasi Hungarian adalah mengawali setiap identifier variabel dengan sejumlah karakter yang menerangkan type variabel, seperti i untuk integer, l untuk long. Notasi lain yang mungkin adalah untuk membedakan variabel global, pointer dan lainnya.

Initialization of variables

Pada C++, nilai awal suatu variable adalah tidak tertentu. Anda dapat memberikan nilai awal pada saat deklarasi dengan penulisan :

```
int a = 0;        // Cara deklarasi di C

int a (0);        // atau cara deklarasi di C++
```

Kedua cara tersebut diatas dapat dilakukan pada C++.

Constants: Literals.

Suatu konstanta adalah ekspresi yang memiliki nilai tetap yang dapat berupa data type Integer, Floating-Point, Character dan String

Integer Numbers

```
75          // decimal
0113        // octal
0x4b        // hexadecimal
```

Floating Point Numbers

```
3.14159     // 3.14159
6.02e23     // 6.02 x 10^23
1.6e-19     // 1.6 x 10^-19
3.0         // 3.0
```

Characters and strings

```
'z'
'p'
"Hello world"
"How do you do?"
```

Perhatikan pemakaian single quote (‘), dan double quotes (").

Konstanta karakter memiliki beberapa pengecualian, seperti kode **escape**, yang merupakan karakter spesial yang tidak dapat diekspresikan kecuali dalam source code, seperti *newline* (`\n`) atau *tab* (`\t`). Berikut ini adalah daftar dari kode escape :

\n	newline
\r	carriage return
\t	tabulation
\v	vertical tabulation
\b	backspace
\f	page feed
\a	alert (beep)
\'	single quotes (')
\"	double quotes (")
\?	question (?)
****	inverted slash (\)

Contoh :

```
'\n'
'\t'
"Left \t Right"
"one\ntwo\nthree"
```

Sebagai tambahannya, anda dapat menulis karakter dengan menggunakan ASCII code seperti **\23** atau **\40,** atau secara hexadesimal seperti **\x20** atau **\x4A**.

Konstanta string dapat dibagi menjadi lebih dari satu baris dengan mengakhirinya dengan slash (\):

```
"string expressed in \
two lines"
```

Anda juga dapat mengabungkan beberapa konstanta string dengan spasi:

```
"we form" "a single" "string" "of characters"
```

Defined constants (*#define*)

Anda dapat mendefinisikan nama tertentu pada konstanta yang sering anda gunakan dengan menggunakan prepocessor directive **#define** preprocessor directive yang memiliki syntax berikut :

```
#define identifier value
```

Sebagai contoh:

```
#define PI 3.14159265
#define NEWLINE '\n'
#define WIDTH 100
```

Setelah definisi diatas, kita dapat memakainya seperti berikut:

```
circle = 2 * PI * r;
cout << NEWLINE;
```

Sesuatu yang haris diingat adalah bahwa directive `#define` bukan merupakan baris perintah, tetapi merupakan directive untuk preprocessor, sehingga anda tidak perlu memberikan semicolon (`;`) pada akhirannya.

declared constants (`const`)

Anda dapat juga menggunakan awalan **`const`** untuk mendeklarasikan konstanta dengan type tertentu sebagaimana anda lakukan pada variabel:

```
const int width = 100;
const char tab = '\t';
const zip = 12440;
```

Dalam hal ini, jika type tidak ditentukan (seperti pada contoh terakhir), maka compiler akan mengasumsikan type-nya sebagai **`int`**.

Operators.

Sebagaimana pada Pascal untuk mengoperasikan variable dan konstanta kita membutuhkan operator, adapun operator yang disediakan oleh C++ adalah sebagai berikut :

Assignation (=).

Operator assignation berfungsi memberikan nilai pada suatu variabel.

Contoh :

```
int a, b;     // a:?  b:?
a = 10;       // a:10 b:?
b = 4;        // a:10 b:4
a = b;        // a:4 b:4
b = 7;        // a:4 b:7
```

Salah satu fasilitas C++ yang tidak terdapat pada bahasa pemrograman lainnya adalah operation assignation, Contoh:

```
a = 2 + (b = 5);
```

adalah sama dengan penulisan:

```
b = 5;
a = 2 + b;
```

dan contoh :

```
a = b = c = 5;
```

memberikan nilai lima ke variable **a**, **b** dan **c**.

Arithmetic operators (+, -, *, /, %)

Adapun operasi yang dilakukan adalah:

+ addition
- subtraction
* multiplication
/ division
% module

Saya yakin operasi seperti penjumlahan, pengurangan, perkalian,pembagian dan module sudah jelas bagi anda.

Compound assignation operators (+=, -=, *=, /=, %=, >>=, <<=, &=, ^=, |=)

Salah satu fasilitas C++ yang cukup ditakuti adalah pemakaian operator compound assignation (**+=**, **-=**, ***=** and **/=** among others) :

`value += increase;` adalah sama dengan **`value = value + increase;`**
`a -= 5;` adalah sama dengan **`a = a - 5;`**
`a /= b;` adalah sama dengan **`a = a / b;`**
`price *= units + 1;` adalah sama dengan **`price = price * (units + 1);`**

Increase and decrease.

Contoh lain dari penyingkatan penulisan operator adalah pemakaian *increase* operator (**++**) dan *decrease* operator (**--**). Mereka menambahkan atau mengurangi variable dengan 1. Contoh :

```
a++;
a+=1;
a=a+1;

a--;
a-=1;
a=a-1;
```

Salah satu karakteristik dari pemakaian operator ini adalah dapat digunakan sebagai *prefix* atau sebagai *suffix*. Yang berarti dapat dituliskan sebelum dan sesudah variabel (**++a**) atau (**a++**). Pada dasarnya **a++** atau **++a** memiliki arti yang sama, Cuma bedanya adalah urutan operasi, contoh:

Example 1

```
B=3;
A=++B;
// A is 4, B is 4
```

Example 2

```
B=3;
A=B++;
// A is 3, B is 4
```

Pada contoh 1, nilai **B** ditambah satu sebelum diberikan ke **A**. Sedangkan pada contoh 2, nilai dari **B** diberikan ke **A** dan selanjutnya nilai **B** di tambah satu.

Relational operators (`= =, !=, >, <, >=, <=`)

Untuk evaluasi perbandingan antara dua ekspresi, kita dapat menggunakan operator Relational operators yang akan menghasilkan **`true`** atau **`false`**.

Berikut ini adalah operator Relational pada C++:

== Equal
!= Different
> Greater than
< Less than
>= Greater or equal than
<= Less or equal than

Contoh :

`(7 == 5)`	akan menghasilkan **`false`**.
`(5 > 4)`	akan menghasilkan **`true`**.
`(3 != 2)`	akan menghasilkan **`true`**.
`(6 >= 6)`	akan menghasilkan **`true`**.
`(5 < 5)`	akan menghasilkan **`false`**.

Logic operators (`!, &&, ||`).

Operator **!** adalah sama dengan operator boolean NOT, dan hanya memiliki satu operand yang berada dikanannya. Contoh :

`!(5 == 5)`	mengembalikan **`false`** karena ekspresi `(5 == 5)` menghasilkan **`true`**.
`!(6 <= 4)`	mengembalikan **`true`**.
`!true`	mengembalikan **`false`**.
`!false`	mengembalikan **`true`**.

Operator Logical **`&&`** (AND) **`||`** (OR), dengan hasil operasi berikut:

First Operand **a**	Second Operand **b**	Result **a && b**	Result **a || b**
true	true	**true**	**true**
true	false	**false**	**true**
false	true	**false**	**true**
false	false	**false**	**false**

Contoh:

`( (5 == 5) && (3 > 6) )` returns **`false`** (*`true && false`*).
`( (5 == 5) || (3 > 6) )` returns **`true`** (*`true || false`*).

Conditional operator (`?`).

Operator conditional melakukan evaluasi dan mengembalikan nilai yang berbeda tergantung kepada hasil evaluasi, apakah *true* atau *false*. Dengan format:

```
condition ? result1 : result2
```

Jika *`condition`* adalah **`true`** ekspresi akan mengembalikan *`result1`*, jika tidak akan mengembalikan *`result2`*.

`7==5 ? 4 : 3`	returns **3** since **7** is not equal to **5**.
`7==5+2 ? 4 : 3`	returns **4** since **7** is equal to **5+2**.
`5>3 ? a : b`	returns **`a`**, since **5** is greater than **3**.
`a>b ? a : b`	returns the greater one, **`a`** or **`b`**.

Bitwise Operators (`&, |, ^, ~, <<, >>`).

Operator Bitwise melakukan modifikasi terhadap variable yang didasarkan kepada nilai bit yang tersimpan

.

op	asm	Description
&	**AND**	Logical AND
|	**OR**	Logical OR
^	**XOR**	Logical exclusive OR
~	**NOT**	Complement to one (bit inversion)
<<	**SHL**	Shift Left
>>	**SHR**	Shift Right

Explicit type casting operators

Operator Type casting memungkinkan anda untuk melakukan konversi terhadap suatu data ke type yang lain. Cara yang paling baik adalah mengawali data dengan nama type baru yang ditulis dalam kurung `()`:

```
int i;
float f = 3.14;
i = (int) f;
```

Kode diatas mengubah angka **3.14** ke suatu nilai integer (**3**).

sizeof()

Operator ini memiliki satu parameter, dapat berupa variable maupun type variable, dan mengembalikan ukurannya dalam byte:

```
a = sizeof (char);
```

Contoh diatas akan mengembalikan **1** karena **char** adalah berukuran 1 byte.

Priority of operators

Ketika membuat ekspresi yang kompleks dengan beberapa operand, kita akan ragu operand mana yang akan dievaluasi terlebih dahulu, sebagai contoh :

```
a = 5 + 7 % 2
```

kita akan meragukan, apakah :

`a = 5 + (7 % 2)` with result **6**, or
`a = (5 + 7) % 2` with result **0**

Jawaban yang benar adalah ekspresi yang kedua

Berikut ini adalah prioritas tertinggi sampai prioritas yang terendah:

Priority	Operator	Description	Associativity
1	::	scope	Left
2	() [ ] -> . sizeof		Left
3	++ --	increment/decrement	Right
	~	Complement to one (bitwise)	
	!	unary NOT	
	& *	Reference and Dereference (pointers)	
	(*type*)	Type casting	
	+ -	Unary less sign	
4	* / %	arithmetical operations	Left
5	+ -	arithmetical operations	Left
6	<< >>	bit shifting (bitwise)	Left
7	< <= > >=	Relational operators	Left
8	== !=	Relational operators	Left

9	& ^ |	Bitwise operators	Left
10	&& ||	Logic operators	Left
11	?:	Conditional	Right
12	= += -= *= /= %= >>= <<= &= ^= |=	Assignation	Right
13	,	Comma, Separator	Left

Pertanyaan

1. Tuliskan pengertian bahasa C/C++ merupakan bahasa case sensitif.
2. Tuliskan dasar ide pengunaan notasi Hungarian.
3. Tuliskan pengertian keyword, dan mengapa keyword tidak boleh digunakan sebagai identifier.
4. Tuliskan fungsi #include paga program C/C++.
5. Tuliskan perbedaan pemakaian komentar // dengan /* … */.
6. Tuliskan pengertian escape code, dan rincikan daftar escape code dalam C/C++.
7. Tuliskan perbedaan antara integer dan float, bagaimana dengan unsigned short int dengan long int.
8. Tuliskan ekspresi dalam bahasa C++ untuk formula berikut ini :

$$b^2 - 4ac \quad \frac{a^2+1}{bc} \quad \frac{1}{1+x^2} \quad a*-(b+c)$$

9. Tuliskan hasil evaluasi perhitungan dari ekspresi-ekspresi berikut ini :

```
17/3  17%3  1/2  1/2*(x+y)
```

10. Berdasarkan deklarasi berikut ini

```
float x;
int k, i = 5, j = 2;
```

Tuliskan nilai variabel k dan x berdasarkan penugasan (assigment) berikut :

- `k = i/j;`
- `x = i/j;`
- `k = i%j;`
- `x = 5.0/j;`

11. Type data apa yang sesuai untuk mewakili hasl berikut ini?
    - Jumlah mahasiswa dalam ruang kelas.
    - Nilai huruf hasil ujian mahasiswa.
    - Nilai rata-rata kelas.
    - Jarak antara dua titik
    - Jumlah populasi suatu kota
    - Berat dari suatu surat

12. Tuliskan deklarasi yang sesuai untuk pertanyaan nomor 4. Pilihkan identifiers yang sesuai.
13. Tuliskan perbedaan antara int dengan unsigned integer.

Latihan

1. Tuliskan deklarasi konstanta untuk menyimpan jumlah hari dalam satu minggu, dan jumlah minggu dalam satu tahun. Secara terpisah deklarasikan juga konstanta untuk `pi` sebagai 3.1415927.
2. Tuliskan deklarasi variabel integer I dan j, variabel float x dan y. Kembangkan deklarasi variabel I dan j dengan nilai awal 1, dan 10.0 untuk y.
3. Ketik dan jalankan program berikut, dan analisa hasil outputnya.

```
#include <iostream.h>
int main()
{
cout << "The size of an int is:\t\t"    << sizeof(int)    << " bytes.\n";
cout << "The size of a short int is:\t" << sizeof(short)  << " bytes.\n";
cout << "The size of a long int is:\t"  << sizeof(long)   << " bytes.\n";
cout << "The size of a char is:\t\t"    << sizeof(char)   << " bytes.\n";
cout << "The size of a float is:\t\t"   << sizeof(float)  << " bytes.\n";
cout << "The size of a double is:\t"    << sizeof(double) << " bytes.\n";
return 0;
}
```

Section 4
Communication through console.

Console adalah dasar dari interface komputer, normalnya adalah keyboard dan screen. Keyboard adalah standard *input* device dan screen adalah standard *output* device.

Dalam library *iostream* C++, operasi standar *input* dan *output* untuk sebuah program didukung oleh dua data streams: **`cin`** untuk input dan **`cout`** untuk output. Tambahannya adalah, **`cerr`** dan **`clog`** yang mana merupakan stream khusus untuk menampilkan error messages. Mereka dapat di redirected ke standard output atau ke suatu log file.

Output (`cout`)

Stream **`cout`** digunakan dalam hubungannya dengan overloaded operator << (sepasangan tanda *"less than"*).

```
cout << "Output sentence"; // prints Output sentence on
screen
cout << 120;               // prints number 120 on screen
cout << x;                 // prints the content of
variable x on screen
```

Operator *insertion* (<<) dapat digunakan lebih dari sekali pada kalimat yang sama:

```
cout << "Hello, " << "I am " << "a C++ sentence";
```

Anda dapat menggunakan **`endl`** untuk membuat baris baru, contoh:

```
cout << "First sentence." << endl;
cout << "Second sentence." << endl;
```

hasil cetak:

```
First sentence.
Second sentence.
```

Input (`cin`).

Menangani standard input pada C++ dilakukan dengan menerapkan operator overloaded operator atau *extraction* (>>) pada stream **`cin`**. Hal ini harus diikuti oleh variable yang akan menyimpan data yang akan dibaca, sebagai contoh :

```
int age;
cin >> age;
```

```
// i/o example
```

```
Please enter an integer value: 702
```

```
#include <iostream.h>

int main ()
{
  int i;
  cout << "Please enter an integer value: ";
  cin >> i;
  cout << "The value you entered is " << i;
  cout << " and its double is " << i*2 << ".\n";
  return 0;
}
```

```
The value you entered is 702 and
its double is 1404.
```

Anda dapat juga menggunakan **`cin`** untuk menanyakan lebih dari satu data:

```
cin >> a >> b;
```

adalah sama dengan:

```
cin >> a;
cin >> b;
```

Dalam hal diatas, anda perlu memasukan dua data, satu untuk variabel **`a`** dan variable **`b`** yang harus dipisahkan dengan spasi kosong atau baris baru.

iostream Manipulators

iomanip merupakan bagian dari Standard C++ I/O Library yang digunakan untuk memanipulasi tampilan I/O pada perintah cout.

Berikut ini adalah beberapa contoh pemakaian iomanip dan hasil outputnya.

```
// setprecision example
#include <iostream>
#include <iomanip>
using namespace std;

int main () {
  double f =3.14159;
  cout << setprecision (5) << f << endl;
  cout << setprecision (9) << f << endl;
  return 0;
}
```

The execution of this example shall display:

```
  3.1416
  3.14159
```

```
// setw example
#include <iostream>
#include <iomanip>
using namespace std;
```

```
int main () {
  cout << setw (10);
  cout << 77 << endl;
  return 0;
}
```

This code uses `setw` to set the field width to `10` characters.

```
// setfill example
#include <iostream>
#include <iomanip>
using namespace std;

int main () {
  cout << setfill ('x') << setw (10);
  cout << 77 << endl;
  return 0;
}
```

This code uses `setfill` to set the fill character to `'x'`. The output of this example shall be similar to:

```
 xxxxxxxx77
```

```
// setbase example
#include <iostream>
#include <iomanip>
using namespace std;

int main () {
  cout << setbase (16);
  cout << 100 << endl;
  return 0;
}
```

This code uses `setbase` manipulator to set hexadecimal as the basefield. The output of this example is the hexadecimal value of `100`, i.e. `64`.

```
// resetiosflags example
#include <iostream>
#include <iomanip>
using namespace std;

int main () {
  cout << hex << setiosflags (ios_base::showbase);
  cout << 100 << endl;
  cout << resetiosflags (ios_base::showbase);
  cout << 100 << endl;
  return 0;
}
```

This code first sets flag `showbase` and then resets it using `resetiosflags` manipulator. The execution of this example shall display:

```
  0x64
  64
```

Pertanyaan

1. Tuliskan perintah C++ untuk menanyakan tiga buah bilangan yang akan disimpan kedalam variabel integer first, second dan third.
2. Tuliskan perintah C++ untuk menghasilkan output dari variabel x dengan tampilan sebagai berikut :

   ```
   Nilai dari x adalah .....
   ```

3. Tuliskan perintah C++ untuk menghasilkan output sebagai berikut:

   ```
   A circle of radius (jari-jari) .....
             has area .....
    and circumference (keliling) .....
   ```

   dimana nilai dari , area dan circumference disimpan dalam variabel `rad`, `area`, dan `circum`.

4. Perbaiki syntax errors dalam program C++ berikut ini:

   ```
   include iostream.h

   Main();
   {
    Float x,y,z;
    cout < "Enter two numbers ";
    cin >> a >> b
    cout << 'The numbers in reverse order are'
         << b,a;
   }
   ```

Latihan

1. Buatlah program yang menghasilkan tampilan berikut dengan menggunakan cout:

   ```
   XXXXX
   X
   X
   XXX
   X
   X
   XXXXX
   ```

2. Buatlah program yang membaca empat karakter dan mencetak kembali pada masing-masing baris dengan diapit oleh tanda petik tunggal.
3. Buatlah sebuh program yang menampilkan pesan kepada pemakai untuk memasukan dua bilangan integer dan satu float serta mencetak kembali ke layar.

Section 5
Control Structures

Suatu program biasanya tidak hanya berupa suatu urutan linear dari instruksi-instruksi. Dalam prosesnya mungkin bercabang, berulangan atau membuat keputusan. Untuk keperluan tersebut, C++ menyediakan struktur kendali yang ditujukan untuk menentukan apa yang akan dilakukan pada program kita.

Conditional structure: *if* and *else*

Digunakan untuk menjalankan suatu instruksi atau blok dari instruksi kalau kondisi terpenuhi, penulisannya adalah sebagai berikut :

```
if (condition) statement
```

dimana *condition* adalah ekspresi yang akan di evaluasi. Jika kondisi ini **true**, ***statement*** akan dieksekusi. Jika false, *statement* akan diabaikan (tidak dieksekusi) dan program berlanjut ke instruksi berikutnya setelah struktur conditional.

Sebagai contoh, potongan program berikut akan mencetak **x is 100** hanya jika nilai yang tersimpan dalam variabel **x** adalah 100:

```
if (x == 100)
  cout << "x is 100";
```

Jika anda menginginkan lebih dari satu instruksi dijalankan kalau *condition* adalah **true** , anda dapat memberikan *blok instruksi* dengan mengunakan kurung kurawal { }:

```
if (x == 100)
 {
  cout << "x is ";
  cout << x;
 }
```

Kita dapat menentukan apa yang akan dilakukan kalau condition tidak terpenuhi dengan menggunakan keyword *else*. Bentuk penulisannya adalah:

```
if (condition) statement1 else statement2
```

Contoh:

```
if (x == 100)
  cout << "x is 100";
else
  cout << "x is not 100";
```

mencetak pada layar **x is 100** jika x bernilai 100, dan **x is not 100** jika sebaliknya.

Struktur *if* + *else* dapat digabungkan untuk memeriksa sejumlah nilai seperti contoh berikut :.

```
if (x > 0)
  cout << "x is positive";
else if (x < 0)
  cout << "x is negative";
else
  cout << "x is 0";
```

Ingat, jika instruksi yang akan dijalankan lebih dari satu, kita harus mengelompokannya dengan menggunakan kurung kurawal `{ }`.

Catatan :
Didalam penulisan perintah terstruktur kita perlu melakukan Indentasi, dimana untuk memudahkan pembacaan program, berikut ini adalah beberapa contoh style yang sering dilakukan oleh programmer

* Menempatkan tanda kurawal pembuka pada baris yang sama dengan if.

```
if (expression){
    statements
}
```

* Menempatkan tanda kurawal pada baris baru tetapi sejajar dengan if

```
if (expression)
{
   statements
}
```

* Menempatkan tanda kurawal pada baris baru dan sejajar dengan statement..

```
if (expression)
  {
  statements
  }
```

Pemilihan Indentasi Style diserahkan kepada programmer, sepanjang pemakaiannya konsisten.

Repetitive structures or loops

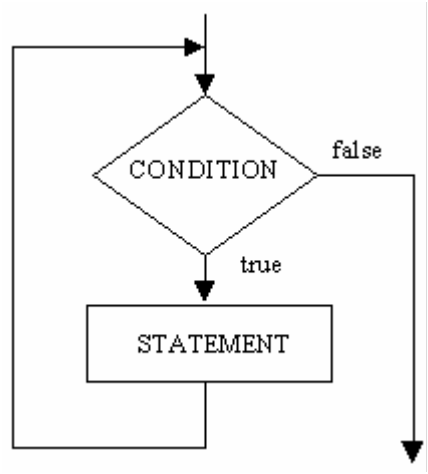
Loops memiliki tujuan untuk mengulangi suatu *statement* untuk sejumlah kali tertentu ketika condition terpenuhi.

The *while* loop.

Formatnya adalah:

```
while (expression) statement
```

dan secara sederhananya fungsi ini akan mengulangi `statement` ketika `expression` adalah benar.



Sebagai contoh, kita akan membuat program untuk menghitung dengan suatu *while* loop:

```
// custom countdown using while
#include <iostream.h>
int main ()
{
  int n;
  cout << "Enter the starting number > ";
  cin >> n;
  while (n>0) {
    cout << n << ", ";
    --n;
  }
  cout << "FIRE!";
  return 0;
}
```

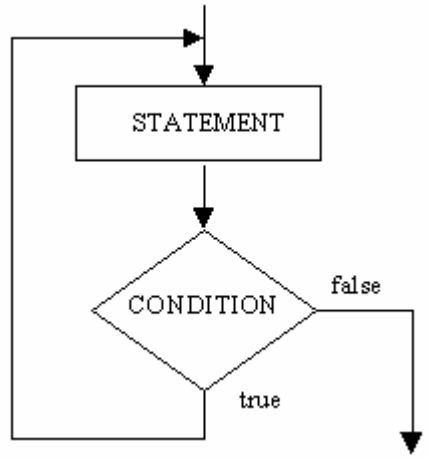
```
Enter the starting number > 8
8, 7, 6, 5, 4, 3, 2, 1, FIRE!
```

The *do-while* loop.

Format:

```
do statement while (condition);
```

Fungsinya hampir sama dengan *while* loop kecuali bahwa `condition` pada *do-while* di evaluasi setelah eksekusi `statement`



Contoh:

```
// number echoer
#include <iostream.h>
int main ()
{
  unsigned long n;
  do {
    cout << "Enter number (0 to end): ";
    cin >> n;
    cout << "You entered: " << n << "\n";
  } while (n != 0);
  return 0;
}
```

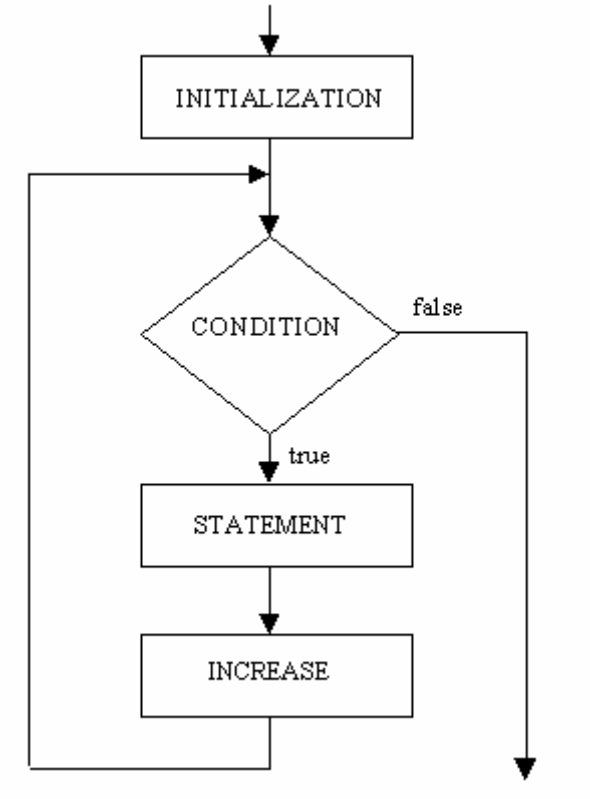
```
Enter number (0 to end): 12345
You entered: 12345
Enter number (0 to end): 160277
You entered: 160277
Enter number (0 to end): 0
You entered: 0
```

The *for* loop.

Format penulisannya adalah sebagai berikut:

```
for (initialization; condition; increase) statement;
```

dan fungsi utamanya adalah mengulangi *statement* ketika *condition* masih benar, menyerupai *while* loop. Tetapi tambahannya adalah, **for** menyediakan tempat untuk menentukan suatu instruksi *initialization* dan suatu *increase* instruction. Jadi loop ini khusus dirancang untuk melakukan suatu aksi perulangan berdasarkan counter.



Cara kerjanya adalah sebagai berikut:

1, *initialization* dijalankan. Umumnya adalah menentukan nilai awal untuk variable counter. Bagian ini hanya dieksekusi sekali.
2, *condition* diperiksa, jika **true** maka loop dilanjutkan, selain itu loop berakhir dan *statement* diloncati.
3, *statement* dijalankan. Pada umumnya dapat berupa instruksi tunggal atau sekumpulan instruksi yang diapit dengan kurung kurawal **{ }**.
4, akhirnya, sebagaimana yang dinyatakan field *increase* akan dijalankan dan loop kembali ke langkah 2

Berikut ini adalah contoh of countdown dengan menggunakan *for* loop.

```
// countdown using a for loop
#include <iostream.h>
int main ()
{
  for (int n=10; n>0; n--) {
    cout << n << ", ";
  }
  cout << "FIRE!";
  return 0;
```

```
10, 9, 8, 7, 6, 5, 4, 3, 2, 1,
FIRE!
```

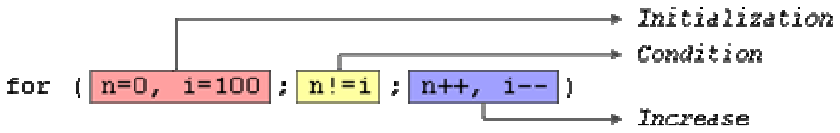
```
}
```

Field *initialization* dan *increase* adalah bersifat optional. Mereka dapat diabaikan dengan semicolon diantaranya. Sebagai contoh anda dapat menulis: **for (;n<10;)** jika tidak ingin menentukan *initialization* dan *increase*; atau **for (;n<10;n++)** jika kita ingin menggunakan *increase* tetapi tidak untuk *initialization*.

Secara optinal, pemakaian operator comma (,) kita dapat menentukan lebih dari satu instruksi dalam field didalam suatu **for** loop, Contoh :

```
for ( n=0, i=100 ; n!=i ; n++, i-- )
{
  // whatever here...
}
```

Loop ini akan dijalankan 50 kali, jika **n** atau **i** tidak dimodifikasi dalam loop:



Bifurcation (percabangan) of control and jumps.

Perhatian, fasilitas-fasilitas berikut bisa saja membuat program anda menjadi tidak terstruktur karena akan menyebabkan titik keluar dari structure lebih dari satu, tetapi anda tetap dapat menggunakannya kalau menguntungkan secara logika.

The *break* instruction.

Dengan menggunakan *break* kita dapat meninggalkan suatu loop walaupun condition masih terpenuhi.

```
// break loop example
#include <iostream.h>
int main ()
{
  int n;
  for (n=10; n>0; n--) {
    cout << n << ", ";
    if (n==3)
    {
      cout << "countdown aborted!";
      break;
    }
  }
  return 0;
}
```

```
10, 9, 8, 7, 6, 5, 4, 3,
countdown aborted!
```

The *continue* instruction.

Instruksi *continue* menyebabkan program loncat ke akhir dari loop, dan melanjutkan ke iterasi berikutnya, contoh :

```
// break loop example
#include <iostream.h>
int main ()
{
  for (int n=10; n>0; n--) {
    if (n==5) continue;
    cout << n << ", ";
  }
  cout << "FIRE!";
  return 0;
}
```

```
10, 9, 8, 7, 6, 4, 3, 2, 1,
FIRE!
```

The *goto* instruction.

Perintah ini memungkinkan anda untuk melakukan loncatan absolut ke suatu bagian program tertentu. Anda harus menggunakan fasilitas ini secara hati-hati dalam kaitannya dengan struktur program anda.

Titik tujuan diidentifikasi oleh suatu label, dimana digunakan sebagai argumen untuk instruksi goto. Penulisan label yang benar diikuti dengan sebuah colon (:).

Berikut ini adalah contoh pemakaian **`goto`** untuk melakukan perulangan yang tidak terstruktur :

```
// goto loop example
#include <iostream.h>
int main ()
{
  int n=10;
  loop:
  cout << n << ", ";
  n--;
  if (n>0) goto loop;
  cout << "FIRE!";
  return 0;
}
```

```
10, 9, 8, 7, 6, 5, 4, 3, 2, 1,
FIRE!
```

The *exit* function.

exit terdefinisi dalam library **`cstdlib`** (stdlib.h).

Tujuan dari *exit* adalah untuk menghentikan program yang berjalan dengan kode exit tertentu. Prototypenya adalah sebagai berikut:

```
void exit (int exit code);
```

Kode exit digunakan oleh system operasi tertentu yang mana memanggil program tersebut. Secara konvensi, suatu *exit code* 0 berarti program berakhir secara normal dan nilai lain berarti suatu kesalahan telah terjadi.

The selective Structure: *switch*.

Syntax penulisan dari perintah *switch* agak aneh. Fungsinya adalah untuk memeriksa beberapa nilai konstanta dari suatu ekspresi, dan melakukan sesuatu, berikut ini adalah bentuk penulisan switch:

```
switch (expression) {
  case constant1:
    block of instructions 1
    break;
  case constant2:
    block of instructions 2
    break;
  .
  .
  .
  default:
    default block of instructions
  }
```

Cara kerjanya adalah sebagai berikut: **switch** mengevaluasi `expression` dan memeriksa jika nilainya ekuivalen dengan `constant1`, jika ya, jalankan `block of instructions 1` sampai menemukan keyword **break**, kemudian program meloncat ke akhir dari struktur seleksi *switch.*
Jika `expression` tidak sama dengan `constant1` , maka pemeriksaan dilakukan pada `constant2`. Jika ya, `block of instructions 2` akan dijalankan sampai ditemukan keyword **break**.
Akhirnya, jika nilai dari `expression` tidak sama dengan konstanta-konstanta sebelumnya, maka bagian **default:** akan dijalankan.

Kedua koding berikut menghasilkan hal yang sama:

***switch* example**

```
switch (x) {
  case 1:
    cout << "x is 1";
    break;
  case 2:
    cout << "x is 2";
    break;
  default:
    cout << "value of x unknown";
  }
```

***if-else* equivalent**

```
if (x == 1) {
  cout << "x is 1";
  }
else if (x == 2) {
  cout << "x is 2";
  }
else {
  cout << "value of x unknown";
  }
```

Pertanyaan

1. Tuliskan fungsi blok { dan } dalam C/C++.

2. Jika `x` memliki nilai 3.5 ketika perintah berikut ini dijalankan, nilai berapa yang akan disimpan ke y?

```
if (x + 1 <= 3.6)
   y = 1.0;
else
   y = 2.0;
```

3. Jelasnkan fungsi pernyatan berikut ? Mengapa secara sintaksisnya salah? Bagaimana memperbaikinya sehingga menjadi benar?

```
if (x >= y)
   sum += x;
   cout << "x is bigger" << endl;
else
   sum += y;
   cout << "y is bigger" << endl;
```

4. Tuliskan suatu pernyataan `if-else` yang mana akan menambah suatu variabel x ke variabel `possum` jika `x` adalah positif dan akan menambahkan x ke `negsum` jika varibel x bernilai negatif.
5. Kembangkan solusi pertanyaan sebelumnya sehingga jika x adakah positif, maka suatu variabel `poscount` ditambah 1 dan sebaliknya variable `negcount` yang akan ditambah 1.
6. Seorang mahasiswa diberi nilai dari `A' ke `F' dalam suatu ujian. Sehingga rata-rata dapat dihitung berdasarkan nilai yang diberikan pada grade tersebut sebagaimana, `A' adalah 10, `B' adalah 8, `C' adalah 6, `D' adalah 4, `E' adalah 2 dan `F' adalah 0. Pada C++ ada dua metode untuk melakukan hal tersebut, apakah itu? Tuliskan perintah yang bersesuaian untuk metoda yang anda pilih.

   Seorang mahasiswa memiliki suatu penilaian berdasarkan kriteria berikut ::

```
>=70   `A'
60-69  `B'
50-59  `C'
40-49  `D'
30-39  `E'
<30    `F'
```

   Dapatkah suatu perintah switch digunakan untuk penentuan kriteria penilaian tersebut? Jika tidak bisa, bagaimana melakukan tersebut? Dan tuliskan perintah untuk penanganan masalah diatas.

Latihan

Untuk setiap latihan berikut, gambarkan algoritma yang digunakan dalam bentuk flowchart, baru dilanjutkan dengan penulisan program.

1. Buatlah sebuah program yang mana menghasilkan selisih nilai absolut dari dua bilangan integer (bulat) x dan y, dimana `(x-y)` atau `(y-x)` adalah positif. Pikirkan segala kemungkinan yang dapat terjadi, dan buatlah solusi untuk menangani semua kemungkinan tersebut.
2. Penilaian terhadap nilai ujian mahasiswa ditampilkan dalam kriteria Pass (lulus) atau Fail (gagal) berdasarkan tiga nilai yang dimasukan. Adapun kriteria untuk lulus adalah sebagai berikut:

   Seorang siswa lulus kalau ketiga nilai tersebut bernilai >= 60, atau hanya gagal disalah satu mata kuliah dengan nilai rata-rata >= 60.

3. Buatlah sebuah program C++ yang mana dapat menghitung luas dari suatu persegi ($area = side^2$) atau segitiga ($area = \frac{base * height}{2}$) setelah menampilkan pertanyaan jenis perhitungan yang akan dilakukan (persegi atau segitiga).
4. Tuliskan suatu program yang akan menghasilkan jumlah nilai dari barisan:

   $$1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots + \frac{1}{n}$$

   Dimana n adalah jumlah suku yang dimasukan oleh pemakai.

   Jawaban :

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

int main()
{
int n;

clrscr();
cout << "jumlah suku :";
cin >> n;

float result = 1;

for (int i=2;i<=n;i++)
    result = result + (float) 1/i;

cout << "hasil :" << result << endl;
return 0;

}
```

   Ubahlah program anda untuk menghitung barisan berikut

$$1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots \frac{1}{n}$$

dari hasil perhitungan, jika nilai n cukup besar, maka ankan menghasilkan nilai yang mendekati 0.7854 (π/4).

7. Buatlah suatu program yang menghasilkan suatu tabel perkalian n x n (`dimana nilai n lebih kecil atau sama dengan` 10). Sebagai contoh, jika n bernilai 4 (empat), maka akan ditampilkan tabel perkalian berikut :

```
   1   2   3   4
1  1   2   3   4
2  2   4   6   8
3  3   6   9  12
4  4   8  12  16
```

Section 6

Functions (I)

Pemakaian function akan membuat program anda menjadi lebih terstruktur (baca kembali section 2 jika anda tidak mengerti pernyataan ini).

Sebuah function adalah blok dari instruksi-instruksi yang akan dijalankan ketika dipanggil dari bagian lain program, berikut ini adalah formatnya:

```
type name ( argument1, argument2, ...) statement
```

dimana:

- ***type*** adalah type data yang akan dikembalikan oleh function.
- ***name*** adalah nama yang digunakan untuk memanggil function.
- ***arguments***. Masing-masing argument terdiri dari type data yang diikuti oleh identifier, seperti dalam deklarasi variabel (sebagai contoh, `int x`) dan perlakuannya dalam function menyerupai variabel.
- ***statement*** adalah badan dari function. Dapat berupa satu instruksi tunggal atau blok dari instruksi-instruksi.

Berikut ini adalah contoh function:

```
// function example
#include <iostream.h>

int addition (int a, int b)
{
  int r;
  r=a+b;
  return (r);
}

int main ()
{
  int z;
  z = addition (5,3);
  cout << "The result is " << z;
  return 0;
}
```

```
The result is 8
```

Catatan :
Ketika program anda melakukan pemanggilan terhadap function, maka posisi alamat pemangilan (alamat register ip) akan di push ke stack, dan ketika function berakhir, maka alamat tersebut akan di pop kembali dari stack ke register ip, sehingga program dapat kembali ke proses berikutnya.

Functions with no types. The use of *void*.

Jika anda melihat syntax deklarasi function :

```
type name ( argument1, argument2 ...) statement
```

anda akan melihat bahwa, penulisannya harus diawali dengan suatu ***type***, yang mana merupakan type data yang akan dikembalikan oleh function dengan perintah **return**. Tetapi bagaimana kalau kita tidak ingin mengembalikan suatu nilai?

Bayangkan misalnya kita ingin membuat function yang hanya mencetak pesan ke layar. Kita tidak memerlukan pengembalian suatu nilai, dan juga tidak membutuhkan parameter. Dalam hal ini dapat digunakan type **void**. Perhatikan contoh berikut:

```
// void function example
#include <iostream.h>

void dummyfunction (void)
{
  cout << "I'm a function!";
}

int main ()
{
  dummyfunction ();
  return 0;
}
```

```
I'm a function!
```

Sesuatu hal yang perlu diperhatikan adalah kita memanggil function tersebut harus mengakhiri nama function dengan kurung :

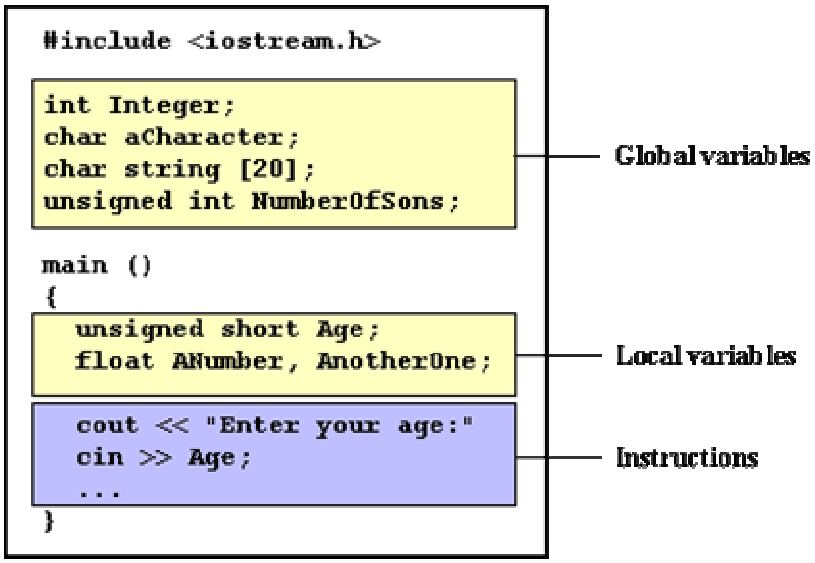
```
dummyfunction ();
```

Hal ini untuk membedakan function, terhadap nama variable ataupun hal yang lain.

Scope of variables

Pada C, variable harus dideklarasikan pada awal setiap function (untuk variable local) atau pada bagian badan program diluar function (untuk variable global).

Suatu perbedaan antara C dan C++ adalah variable dapat dideklarasikan pada saat dibutuhkan.



Pada C++, jangkauan dari variable local adalah berada di blok dimana ia dideklarasikan (suatu blok adalah kumpulan instruksi yang dikelompokan dalam tanda kurawal `{}`). Jika dideklarasikan dalam function, maka jangkauan variable itu adalah dalam function tersebut, sedangkan kalau dalam loop, maka jangkauannya hanya dalam loop tersebut.

Variable Global dapat digunakan pada seluruh program, didalam function, dimanapun setelah deklarasi.

Jangkauan dari **local variables** adalah terbatas pada bagian dimana mereka dideklarasikan. Jika mereka dideklarasikan pada awal dari suatu function (seperti pada **`main`**) jangkauan mereka adalah pada keseluruhan function **`main`**.

Catatan : Programmer pemula sering sekali mendeklarasikan semua variabel yang dibutuhkan dalam program menjadi variabel global. Hal ini akan menyebabkan sulitnya pengendalian nilai variabel pada program yang besar, karena variabel global dapat dimanipulasi oleh semua bagian dalam program.

Math.h library in C++

Beberapa fungsi matematika yang terdapat di C++ mathematics library adalah sebagai berikut :

`acos(x)` inverse cosine, -1 <= `x` <= +1, returns value in radians in range 0 to PI
`asin(x)` inverse sine, -1 <= `x` <= +1, returns value in radians in range 0 to PI
`atan(x)` inverse tangent, returns value in radians in range -PI/2 to PI/2
`cos(x)` mengembalikan nilai cosinus `x`, `x` dalam radianin radians

`sin(x)` returns sine of `x`, `x` in radians
`tan(x)` returns tangent of `x`, `x` in radians
`exp(x)` exponential function, e to power `x`
`log(x)` natural log of `x` (base e), `x` > 0
`sqrt(x)` square root of `x`, `x` >= 0
`fabs(x)` absolute value of `x`
`floor(x)` largest integer not greater than `x`
`ceil(x)` smallest integer not less than `x`
`rand()` an integer value between 0 to RAND_MAX
`srand(x)` initialize a random seed

Contoh

```
/* rand/srand example */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main ()
{
  /* initialize random generator */
  srand ( time(NULL) );

  /* generate some random numbers */
  printf ("A number between 0 and 100: %d\n", rand()%100);
  printf ("A number between 20 and 30: %d\n", rand()%10+20);

  return 0;
}
```

Output:

```
A number between 0 and 100: 93
A number between 20 and 30: 21
```

Semua function diatas menggunakan parameter `x` yang bernilai floating point.

Berikut beberapa contoh pemakaian fungsi yang dimaksud diatas:

```
y = sin(3.14159);
z = cos(a) + sin(a);
factor = sin(theta)/(sin(delta) - sin(delta-theta));
theta = acos(1.0/sqrt(1 - x*x));
if (sin(x) > 0.7071)
  cout << "Angle is greater than 45 degrees";
cout << "The value is " << exp(-a*t)*sin(a*t);
```

Sesuatu hal yang perlu diingat bahwa anda perlu memasukan file `math.h` kedalam program, Library `math.h` juga telah mendefinisikan beberapa konstanta seperti `M_PI` untuk π dan `M_E` untuk e.

Stdlib.h Library in C++

Beberapa fungsi matematika pada Stdlib.h C++ adalah sebagai berikut :

`abs (x)`	absolute value of integer
`div(x,y)`	divide two integer value
`ldiv(x,y)`	divide two long integer value
`labs(x)`	absolute value of long integer

Pertanyaan

1. Sebutkan fungsi pemakaian function?
2. Apa peran parameter pada suatu function?
3. Tuliskan perbedaan antara variabel global dan variabel local.

4. Apa fungsi dari function berikut ini?

```
void example(int n)
  {
    int i;
    for (i=0; i<n; i++)
       cout << '*';
    cout << endl;
  }
```

   Bagaimana anda memanggil function ini dalam program? Bagaimana anda mengunakan fungsi tersebut untuk menghasilkan output berikut ini ke layar?

```
*
**
***
****
```

5. Apa yang menjadi output dari program berikut?

   a)

```
void change(void)
 {
   int x;
   x = 1;
```

```
 }
void main()
 {
   int x;
   x = 0;
   change();
   cout << x << endl;
 }
```

b)

```
void change(int x)
  {
    x = 1;
  }
void main()
  {
    int x;
    x = 0;
    change(x);
    cout << x << endl;
  }
```

6. Tuliskan suatu function prototype untuk suatu function yang memiliki dua paramter dengan type `float` dan mengembalikan **true** (1) jika parameter perta lebih besar dari paramater yang kedua, dan sebaliknya mengembalikan **false** (0).
7. Suatu function dengan nama `ex1` memiliki sebuah variabel local dengan nama `i` dan function lainnya `ex2` juga memiliki suatu variabel local bernama `i`. Kedua function tersebut digunakan bersama dalam suatu program utama yang memiliki sebuah variabel bernama `i`. Andaikan tidak adalah kesalahan lainnya. Apakah program ini dapat dikompilasi dengan tanpa kesalahan? Akankah program tersebut dijalankan tanpa kesalahan waktu run-time?

Latihan

1. Buatlah sebuah function yang akan mencetak suatu barisan sejumlah n tanda asterisk, `n` akan dilewatkan sebagai parameter ke function tersebut. Tuliskan suatu program yang akan menampilkan blok m x n tanda asterik berdasarkan nilai input m dan n oleh pemakai.
2. Kembangkan function pada soal nomor satu, sehingga jika nilai n melewati 80, maka tidak ada yang dicetak, dan function akan mengembalikan nilai false, dan sebaliknya true. Kemudian program utama akan memeriksa nilai pengembalian dari function, serta menampilkan pesan kesalahan overflow jika nilai false.
3. Buatlah suatu function yang dapat menghitung nilai simpanan di bank dengan parameter, present value, term, dan rate.

Section 7
Functions (II).

Arguments passed *by value* and *by reference*.

Sampai saat ini, semua function yang telah dibahas, melewatkan parameter ke function secara *by value*. Hal ini berarti ketika kita memanggil suatu function yang memiliki parameter, apa yang kita lewatkan ke function adalah values jadi bukan variable tersebut. Sebagai contoh, bayangkan misalnya kita memanggil function **`addition`** dengan menggunakan kode berikut:

```
int x=5, y=3, z;
z = addition ( x , y );
```

Tetapi dalam kasus-kasus tertentu, anda perlu memanipulasi nilai dari dalam function terhadap variable external. Untuk keperluan diatas kita perlu menggunakan *arguments passed by reference*, sebagaimana yang ditunjukan pada contoh function **`duplicate`** berikut ini :

```
// passing parameters by reference
#include <iostream.h>

void duplicate (int& a, int& b, int& c)
{
  a*=2;
  b*=2;
  c*=2;
}

int main ()
{
  int x=1, y=3, z=7;
  duplicate (x, y, z);
  cout << "x=" << x << ", y=" << y << ", z=" << z;
  return 0;
}
```

```
x=2, y=6, z=14
```

Default values in arguments.

Kita kita mendeklarasikan suatu function, kita dapat menentukan nilai default untuk masing-masing parameter. Nilai ini akan digunakan ketika parameter tidak diberikan pada saat pemanggilan function, contoh :

```
// default values in functions
#include <iostream.h>

int divide (int a, int b=2)
```

```
6
5
```

```
{
  int r;
  r=a/b;
  return (r);
}

int main ()
{
  cout << divide (12);
  cout << endl;
  cout << divide (20,4);
  return 0;
}
```

Overloaded functions.

Dua function yang berbeda dapat memiliki nama yang sama kalau prototype dari argumen-argumenya berbeda, hal ini berarti bahwa anda dapat memberikan nama yang sama pada lebih dari satu function jika memiliki jumlah argumen atau type yang berbeda pada argumennya, contoh :

```
// overloaded function
#include <iostream.h>

int divide (int a, int b)
{
  return (a/b);
}

float divide (float a, float b)
{
  return (a/b);
}

int main ()
{
  int x=5,y=2;
  float n=5.0,m=2.0;
  cout << divide (x,y);
  cout << "\n";
  cout << divide (n,m);
  cout << "\n";
  return 0;
}
```

```
2
2.5
```

Dalam hal ini kita telah mendefinisikan dua function dengan nama yang sama, tetapi salah satu darinya menerima dua argumen type **`int`** dan yang lainnya menerima type **`float`**. Compiler akan mengetahui yang mana akan dipanggil dengan menganalisa type ketika pemanggilan function.

inline functions.

Directive *inline* dapat diikutsertakan sebelum deklarasi dari function untuk menentukan function mana yang harus di kompilasi sebagai kode sebagaimana ketika mereka dipanggil. Hal ini adalah sama dengan deklarasi sebuah makro. Hal ini bermanfaat kalau function tersebut sangat pendek.

Format dari deklarasi tersebut adalah:

```
inline type name ( arguments ... ) { instructions ... }
```

dan dalam pemakaiannya adalah sama dengan function lainnya.

Recursivity.

Recursivity adalah kemampuan function untuk memanggil dirinya sendiri. Hal ini berguna untuk tugas-tugas seperti sorting dan factorial, sebagai contoh untuk mendapatkan factorial dari bilangan (n), dimana formula matematikanya adalah:

```
n! = n * (n-1) * (n-2) * (n-3) ... * 1
```

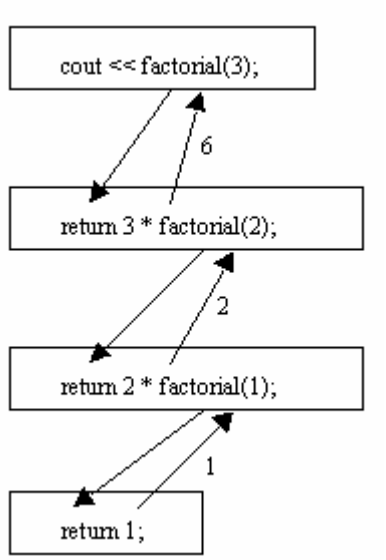
kongkritnya adalah, 5! (factorial dari 5) adalah:

```
5! = 5 * 4 * 3 * 2 * 1 = 120
```

dan contoh programnya adalah:

```
// factorial calculator
#include <iostream.h>

long factorial (long a)
{
  if (a > 1)
   return (a * factorial (a-1));
  else
   return (1);
}

int main ()
{
  long l;
  cout << factorial(3);
}
```

6

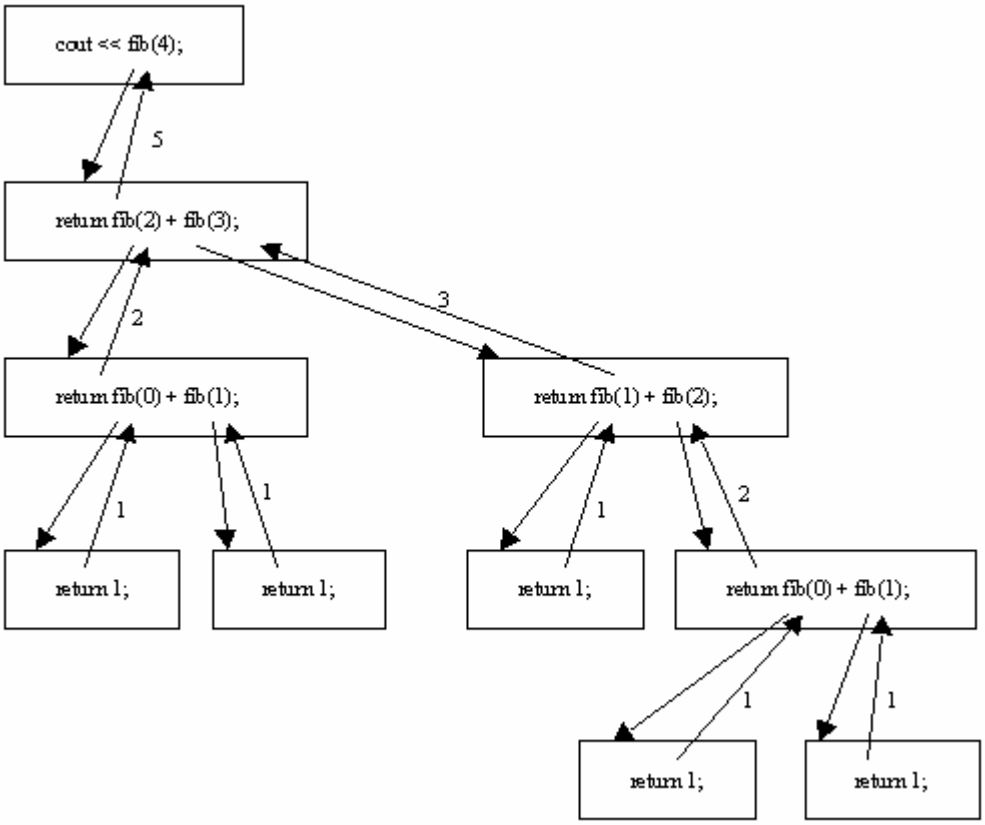


Catatan : Salah satu permasalahan yang sering timbul pada program yang mana function-nya yang menggunakan teknik rekursif adalah kehabisan tempat stack (Stack Overflow).

dan contoh program fibonacci adalah:

```
// fibonacci calculator
#include <iostream.h>

long fib(int n)
{
  if ((n == 0) || (n == 1))
     return 1;
  else
      return fib(n - 1) + fib(n - 2);
}

int main ()
{
  long l;
  cout << fib(4);
}
```

5



Analisis :

Processing fib(4)... Call fib(2) and fib(3).
Processing fib(2)... Call fib(0) and fib(1).
Processing fib(0)... Return 1!
Processing fib(1)... Return 1!
Processing fib(3)... Call fib(1) and fib(2).
Processing fib(1)... Return 1!
Processing fib(2)... Call fib(0) and fib(1).
Processing fib(0)... Return 1!
Processing fib(1)... Return 1!

Catatan : Sesuatu hal yang penting untuk diketahui adalah ketika suatu function memangil dirinya (rekursif), maka suatu duplikasi dari fungsi tersebut akan dijalankan. Variabel local dari fungsi tersebut adalah terpisah antara satu sama yang lainnya dan tidak dapat mempengaruhi yang lain secara langsung.

Prototyping functions.

Sampai saat ini, kita mendefinisikan function sebelum pemanggilannya Tetapi ada cara lain untuk menghindari penulisan semua kode function sebelum function **`main`**, yaitu dengan melakukan *prototyping functions*.

Bentuk prototyping:

```
type name ( argument_type1, argument_type2, ...);
```

Hal ini adalah sama persis dengan header dari suatu function, kecuali:

- Tidak diikuti oleh *`statement`* untuk function tersebut.
- Diakhir dengan tanda semicolon (`;`).

For example:

```
// prototyping
#include <iostream.h>

void odd (int a);
void even (int a);

int main ()
{
  int i;
  do {
    cout << "Type a number: (0 to exit)";
    cin >> i;
    odd (i);
  } while (i!=0);
  return 0;
}

void odd (int a)
{
  if ((a%2)!=0) cout << "Number is odd.\n";
  else even (a);
}

void even (int a)
{
  if ((a%2)==0) cout << "Number is even.\n";
  else odd (a);
}
```

```
Type a number (0 to exit): 9
Number is odd.
Type a number (0 to exit): 6
Number is even.
Type a number (0 to exit): 1030
Number is even.
Type a number (0 to exit): 0
Number is even.
```

Banyak programmer menyarankan anda untuk melakukan prototyped terhadap semua function. Karena dengan prototyped akan memudahkan kita untuk melihat bagaimana penulisan untuk memanggil suatu function.

Pertanyaan

1. Tuliskan tujuan pemakaian prototype pada function.

2. Tuliskan perbedaan antara variabel global, dan local.
3. Tuliskan pengertian function overloading.
4. Bagaimana suatu informasi dijadikan sebagai input ke suatu function? Bagaimana suatu informasi dapat dikembalikan ke program yang memanggilnya?
5. Apa yang akan menjadi output dari program berikut ini?

```
void change(int& y)
  {
    y = 1;
  }
void main()
  {
    int x;
    x = 0;
    change(x);
    cout << x << endl;
  }
```

Latihan

1. Tuliskan suatu function

   void timetopart(float t, int& h, int& m)

   dimana akan mengkonversi waktu menjadi komponen jam dan menit, contoh jika nilai t adalah 2.5, maka akan dikembalikan 2 pada parameter h dan 30 pada parameter m. 2.5 = 2 jam 30 menit.

2. Kembangkan program berikut untuk kemampuan membaca terbilang sampai nilai triliunan

```
//Program baca angka
//Oleh : Hendra Soewarno
//Dengan memanfaatkan fasilitas function

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

void bacaratusan(int n, char kata[]);     // prototype fungsi

int main()
{
char *kata;
clrscr();
bacaratusan(101,kata);
cout << kata;
return 0;
}

// pada type array, variabel dilewatkan secara by reference
```

```
void bacaratusan(int n, char kata[])
{
char angka[20][20] = {"nol","satu","dua","tiga","empat","lima",
     "enam","tujuh","delapan","sembilan","sepuluh",
     "sebelas","duabelas","tigabelas","empatbelas","limabelas",
     "enambelas","tujuhbelas","delapanbelas","sembilanbelas"};
int ratusan,puluhan,satuan,sisa;
char hasil[255]="";
ratusan = n/100;              // ambil digit ratusan
if (ratusan > 0)
   if (ratusan == 1)
      strcat(hasil,"seratus");
   else {
      strcat(hasil,angka[ratusan]);
      strcat(hasil,"ratus");
      }
sisa = n - (ratusan*100);     // ambil sisa
if (sisa < 20 && sisa > 0)    // jika bernilai 0 s/d 19
   strcat(hasil,angka[sisa]); // langsung dibaca dari array
else {
   puluhan = sisa/10;         // ambil digit puluhan
   if (puluhan > 0) {
      strcat(hasil,angka[puluhan]);
      strcat(hasil,"puluh");
      }
   satuan = sisa - (puluhan*10);    // ambil sisa
   if (satuan > 0)
      strcat(hasil,angka[satuan]);
   }
strcpy(kata,hasil);           // duplikasi ke return variabel
}
```

3. Kembangkan program soal no 2 untuk kemampuan membaca angka yang mengandung Sen.
4. Contoh Pemanfaatan teknik rekursif untuk mencari faktor dari suatu bilangan :

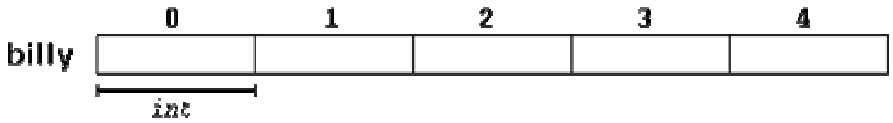
```
#include <iostream.h>

void printfactor(int n);

int main()
{
cout << "faktor dari 100 adalah" << endl;
printfactor(100);
return 0;
}

void printfactor(int n)
{
int factor = 2;
while (factor<n && n % factor)
       factor++;
cout << factor << " " << n / factor<<endl;
if (factor < n)
    printfactor(n/factor);
}
```

Section 8
Arrays

Arrays adalah serial dari elemen-elemen (variable-variabel) yang memiliki type yang sama dan ditempatkan secara berurutan pada memory sehingga dapat dinyatakan dengan menambah index pada nama variable tersebut.

Hal ini berarti bahwa, sebagai contoh, kita dapat menempatkan 5 nilai yang bertype **`int`** tampa harus mendeklarasikan 5 variabel yang berbeda untuk masing-masing nilai. Sebagai gantinya kita dapat menggunakan *array* untuk menyimpan kelima nilai tersebut.

Sebagai contoh, suatu array mengandung lima nilai type **`int`** yang bernama billy dapat digambarkan sebagai berikut:



dimana setiap kotak kosong mewakili *element* dari array, dimana dalam hal ini adalah nilai integer dari type **`int`**. Index array tersebut dimulai dari **`0`** sampai **`4`**.

Sama seperti variable lainnya, suatu array harus dideklarasikan terlebih dahulu sebelum pemakaiannya, deklarasi array pada C++:

```
type name [elements];
```

dima *`type`* adalah type data (**int**, **float**...), *`name`* adalah nama variabel dan *`elements`* adalah jumlah element yang diinginkan, dalam hal ini diapit oleh brackets (kurung siku) **`[]`**.

Selanjutnya, contoh penulisan deklarasi:

```
int billy [5];
```

CATATAN: *Elements* harus berupa nilai constant (konstanta), karena array merupakan blok dari memory static yang harus diberikan, sehingga compiler dapat mengetahui jumlah memory yang diperlukan array sebelum instruksi.

Initializing arrays.

Ketika kita mendeklarasi suatu array local (berada didalam function), jika kita tidak menentukannya nilai awalnya, maka isinya adalah tak tentu.

Jika kita mendeklarasi array global (diluar dari function), nilai awalnya adalah nol. Maka jika kita mendeklarasikan array global berikut:

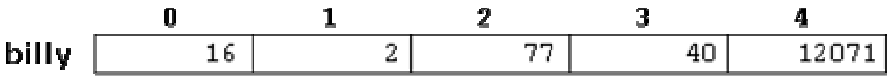
```
int billy [5];
```

setiap elemen dari *billy* adalah `0`:

Kita juga dapat mendeklarasikan suatu Array, berikut dengan nilainya dengan menuliskan nilainya dalam tanda kurawal `{ }`. Sebagai contoh:

```
int billy [5] = { 16, 2, 77, 40, 12071 };
```

Deklarasi diatas akan menghasilkan array sebagai berikut :



Atau pada C++, boleh juga ditulis sebagai berikut :

```
int billy [] = { 16, 2, 77, 40, 12071 };
```

Access to the values of an Array.

Dalam hal ini, setiap element array yang ada dapat diakses satu persatu nilainya dengan format penulisan berikut:

```
name[index]
```

Sebagai contoh, untuk menyimpan nilai 75 pada elemen kelima pada *billy*, penulisan yang sesuai adalah:

```
billy[2] = 75;
```

dan, untuk menyimpan nilai elemen ketiga dari variable billy ke variable `a`, kita dapat menulis:

```
a = billy[2];
```

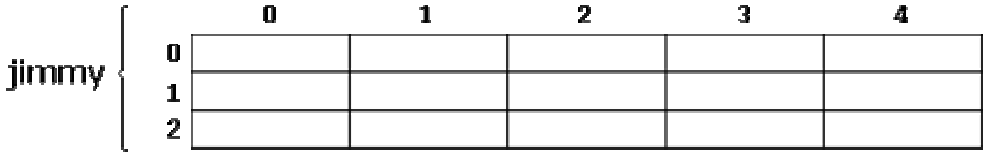
```
// arrays example
#include <iostream.h>

int billy [] = {16, 2, 77, 40, 12071};
int n, result=0;
```

```
12206
```

```
int main ()
{
  for ( n=0 ; n<5 ; n++ )
  {
    result += billy[n];
  }
  cout << result;
  return 0;
}
```

Multidimensional Arrays

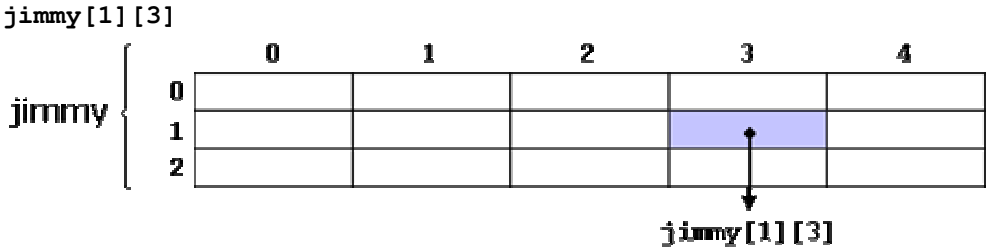
Array multidimensional dapat digambarkan sebagai array dari array.



`Misalnya jimmy` adalah suatu bidimensional array 3 baris 5 kolom dengan type **`int`**. Cara deklarasinya adalah sebagai berikut:

```
int jimmy [3][5];
```

dan, cara untuk menyebutkan elemen baris kedua, kolom keempat adalah dengan ekspresi berikut :



(sesuatu yang harus anda ingat adalah index dari array selalu dimulai dengan **`0`**).

Array Multidimensional tidak hanya terbatas pada dua dimensi saja. Mereka dapat memiliki dimensi sesuai dengan kebutuhan, walaupun sebenarnya jarang melebihi 3 dimensi. Contoh:

```
char century [100][365][24][60][60];
```

Menentukan type **char** untuk setiap detik dalam satu abad, yang terdiri lebih dari 3 milyar **chars**! Hal ini akan menggunakan memory sebesar 3000 *megabytes* dari RAM jika kita mendeklarasikannya.

Arrays as parameters

Jika anda perlu melewatkan suatu array ke sebuah function sebagai parameter. Untuk hal ini kita perlu mendeklarasikan function ini sebagai array berdasarkan **type**, contoh:

```
void procedure (int arg[])
```

Memungkinkan suatu parameter dengan type "Array of **int**" yang bernama **arg**. Untuk dilewatkan kedalam function ini suatu array dideklarasikan sebagai berikut:

```
int myarray [40];
```

Kita cukup menuliskan sebagai berikut:

```
procedure (myarray);
```

Berikut ini adalah contoh konkritnya:

```
// arrays as parameters
#include <iostream.h>

void printarray (int arg[], int length) {
  for (int n=0; n<length; n++)
    cout << arg[n] << " ";
  cout << "\n";
}

int main ()
{
  int firstarray[] = {5, 10, 15};
  int secondarray[] = {2, 4, 6, 8, 10};
  printarray (firstarray,3);
  printarray (secondarray,5);
  return 0;
}
```

```
5 10 15
2 4 6 8 10
```

Dalam deklarasi suatu function, jika dimungkinkan untuk mengikutsertakan suatu array multidimensi. Format untuk array tiga dimensi adalah sebagai berikut:

```
base_type[][depth][depth]
```

sebagai contoh, suatu function dengan array tiga dimensi sebagai parameternya :

```
void procedure (int myarray[][3][4])
```

Contoh Program :

```
//Program permainan tic tac toe
//Oleh : Hendra Soewarno
//Dengan memanfaatkan fasilitas array dan graphic

#include <graphics.h>
#include <conio.h>

int matrix[3][3] = {{0,0,0},{0,0,0},{0,0,0}};  //matrix permainan

// function prototype
int checkgame(int x)              //memeriksa apakah permainan selesai
void drawarena()                  //mengambar kotak 3 x 3 dilayar
void drawcircle(int i,int j)      //mengambar lingkaran di posisi i,j
void drawcross(int i,int j)       //mengambar tanda silang di posisi i,j
void drawselector(int i,int j)    //mengambar kotak seleksi
void clearselector(int i,int j)   //menghapus kotak seleksi
void think()                      //komputer berpikir menyerang/bertahan

// program utama
int main()
{
int gdriver, gmode;

int gameover;
int row =0;
int col = 0;
char key;
detectgraph(&gdriver,&gmode);
initgraph(&gdriver,&gmode,"egavga.bgi");
drawarena();
gameover = 0;
while (!gameover)
 {
 drawselector(row,col);
 key = getch();
 if (key==0)                                  //jika scan code
    key = getch();                            //baca nilai extended
 clearselector(row,col);
 switch (key)                                 //periksa nilai key
    {
    case 27 : gameover = 3; break;            //esc
    case 32 : drawcross(row,col);             //space (manusia)
              gameover = checkgame(1);        //manusia menang ?
              think();                        //giliran komputer
              if (gameover ==0)               //jika belum game
                  gameover = checkgame(9);    //komputer menang ?
              break;
    case 72 : row--; if (row < 0) row = 0; break;  //up
    case 80 : row++; if (row > 2) row = 2; break;  //down
    case 75 : col--; if (col < 0) col = 0; break;  //left
    case 77 : col++; if (col > 2) col = 2; break;  //right
    }
 }
setcolor(15);
```

```
if (gameover==1)
   outtextxy(20,20,"Permainan dimenangkan oleh manusia");
else if (gameover==9)
   outtextxy(20,20,"Permaiaan dimenangkan oleh komputer");
else
   outtextxy(20,20,"Berakhir tanpa pemenang");
getch();
closegraph();
return 0;

}

int checkgame(int x)
{
if (matrix[0][0] == x && matrix[0][1] == x && matrix[0][2] == x)
   return x;
else if (matrix[1][0] == x && matrix[1][1] == x && matrix[1][2] == x)
   return x;
else if (matrix[2][0] == x && matrix[2][1] == x && matrix[2][2] == x)
   return x;
else if (matrix[0][0] == x && matrix[1][0] == x && matrix[2][0] == x)
   return x;
else if (matrix[0][1] == x && matrix[1][1] == x && matrix[2][1] == x)
   return x;
else if (matrix[0][2] == x && matrix[1][2] == x && matrix[2][2] == x)
   return x;
else if (matrix[0][0] == x && matrix[1][1] == x && matrix[2][2] == x)
   return x;
else if (matrix[0][2] == x && matrix[1][1] == x && matrix[2][0] == x)
   return x;
else
   return 0;
}


void drawarena()
{
setcolor(15);
for (int i=0;i<3;i++)
   for (int j=0;j<3;j++)
       rectangle(250+j*50,50+i*50,250+(j+1)*50,50+(i+1)*50);
}

void drawcircle(int i,int j)
{
setcolor(15);
circle(250+j*50+25,50+i*50+25,20);
matrix[i][j] = 9;
}

void drawcross(int i,int j)
{
setcolor(15);
line(250+j*50+2,50+i*50+2,250+(j+1)*50-2,50+(i+1)*50-2);
line(250+j*50+2,50+(i+1)*50-2,250+(j+1)*50-2,50+i*50+2);
matrix[i][j] = 1;
```

```
}

void drawselector(int i,int j)
{
setcolor(4);
rectangle(250+j*50+2,50+i*50+2,250+(j+1)*50-2,50+(i+1)*50-2);
}


void clearselector(int i,int j)
{
setcolor(0);
rectangle(250+j*50+2,50+i*50+2,250+(j+1)*50-2,50+(i+1)*50-2);
}

void think()
{
// algoritma berpikir ini telah dioptimasi sehingga komputer tidak
pernah kalah (kecerdasan yang terprogram)
int r[3];
int c[3];
int d[2];
int i,j,ok;

r[0] = matrix[0][0]+matrix[0][1]+matrix[0][2];
r[1] = matrix[1][0]+matrix[1][1]+matrix[1][2];
r[2] = matrix[2][0]+matrix[2][1]+matrix[2][2];

c[0] = matrix[0][0]+matrix[1][0]+matrix[2][0];
c[1] = matrix[0][1]+matrix[1][1]+matrix[2][1];
c[2] = matrix[0][2]+matrix[1][2]+matrix[2][2];

d[0] = matrix[0][0]+matrix[1][1]+matrix[2][2];
d[1] = matrix[2][0]+matrix[1][1]+matrix[0][2];

ok = 0;
//bertahan atau menyerang pada masing-masing kolom dan baris
for (i=0;i<3;i++)
    for (j=0;j<3;j++)
      if (matrix[i][j]==0 && (r[i]==2 || c[j]==2 || r[i]==18 ||
c[j]==18))
         {
         drawcircle(i,j); ok = 1;
         }

//bertahan atau menyerang pada masing-masing diagonal
if (!ok)
   {
   if (matrix[0][0]==0 && (d[0]==2 || d[0]==18)) {drawcircle(0,0);
ok=1;}
   else if (matrix[2][0]==0 && (d[1]==2 || d[1]==18)) {drawcircle(2,0);
ok=1;}
   else if (matrix[0][2]==0 && (d[1]==2 || d[1]==18)) {drawcircle(0,2);
ok=1;}
   else if (matrix[2][2]==0 && (d[0]==2 || d[0]==18)) {drawcircle(2,2);
ok=1;}
}
```

```
//menyerang pada posisi kosong, dengan prioritas titik tengah
if (!ok)
   {
   if (matrix[1][1]==0) drawcircle(1,1);
   else if (matrix[0][0]==0 && (r[0]==1 || c[0]==1 || d[0]==1 ||
d[0]==9)) {drawcircle(0,0); ok=1;}
   else if (matrix[2][0]==0 && (r[2]==1 || c[0]==1 || d[1]==1 ||
d[1]==9)) {drawcircle(2,0); ok=1;}
   else if (matrix[0][2]==0 && (r[0]==1 || c[2]==1 || d[1]==1 ||
d[1]==9)) {drawcircle(0,2); ok=1;}
   else if (matrix[2][2]==0 && (r[2]==1 || c[2]==1 || d[0]==1 ||
d[0]==9)) {drawcircle(2,2); ok=1;}
   else if (matrix[1][0]==0 && (r[1]==1 || c[0]==1)) {drawcircle(1,0);
ok=1;}
   else if (matrix[0][1]==0 && (r[0]==1 || c[1]==1)) {drawcircle(0,1);
ok=1;}
   else if (matrix[2][1]==0 && (r[2]==1 || c[1]==1)) {drawcircle(2,1);
ok=1;}
   else if (matrix[1][2]==0 && (r[1]==1 || c[2]==1)) {drawcircle(1,2);
ok=1;}
   }
}
```

Pertanyaan

1. Jika suatu array memiliki 100 element, sebutkan range subscript yang diperbolehkan!
2. Apa perbedaan dari ekspresi `a4` dan `a[4]`?
3. Tuliskan sebuah deklarasi untuk suatu array float 100 elemen. Termasuk memberikan nilai awal untuk 4 elemen pertama yaitu 1.0, 2.0, 3.0 dan 4.0.
4. Suatu array dengan nama `day` dideklarasikan sebagai berikut:

   ```
   int day[] = {mon, tue, wed, thu, fri};
   ```

   Berapa banyak element dari array day ?

   Bagaimana kalau deklarasi diganti menjadi

   ```
   int day[7] = {mon, tue, wed, thu, fri};
   ```

5. Apa yang akan menjadi output dari potongan program C++ berikut ini?

   ```
   int A[5] = {1 , 2, 3, 4};
   int i;
   for (i=0; i<5; i++)
    {
     A[i] = 2*A[i];
     cout << A[i] << "   ";
    }
   ```

6. Tuliskan kesalahan dari bagian program berikut!

```
int A[10], i;
for (i=1; i<=10; i++)
   cin >> A[i];
```

Latihan

1. Salah satu metoda penyajian data adalah dengan Histogram. Suatu histogram menghitung jumlah data yang jatuh pada interval tertentu, dan menampilkannya dalam bentuk barchart.

   Buatlah suatu program yang menghasilkan n bilangan bulat random yang berada dalam jangkauan 0-99 dan menghasilkan suatu Histogram dari data tersebut. Asumsikan bahwa kita kita akan membagi data tersebut menjadi interval 0-9, 10-19, 20-29, ..., 90-99. Kemudian cetak Histrogram yang bersesuaian

```
 0 -  9  16  XXXXXXXXXXXXXXXX
10 - 19  13  XXXXXXXXXXXXX
20 - 29  17  XXXXXXXXXXXXXXXXX
dll.
```

2. Kembangkan function pencarian dengan pendekatan binary search dengan memanfaatkan teknik pemrograman rekursif pada function.

Section 9
Strings of Characters.

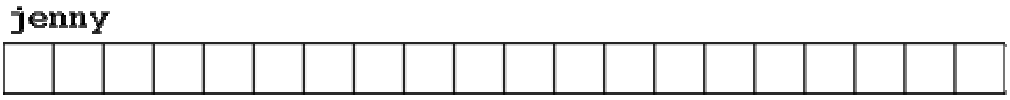
Dalam semua program yang telah kita buat, kita hanya menggunakan variable numeric, dan menggunakan ekspresi numeric. Selain numeric sebenarnya masih ada karakter string

Pada C untuk menyimpan karakter string kita dapat menggunakan type **`char`**, yang mana merupakan urutan dari elemen **`char`**

Sebagai contoh, array berikut (atau karakter string):

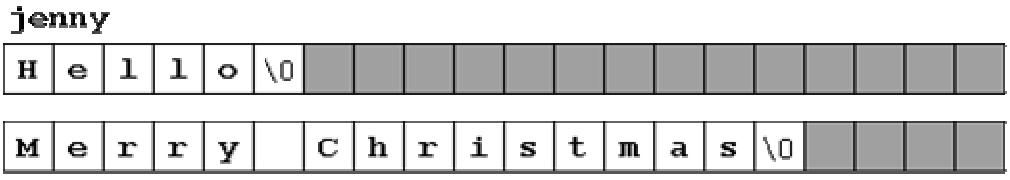
```
char jenny [20];
```

dapat menyimpan 20 karakter. Anda dapat membayangkannya sebagai:



Ukuran maksimum dari 20 karakter tidak selamanya akan digunakan secara keseluruhan. Dapat saja hanya berisi tulisan `"Hello"` ataupun `"Merry christmas"`, sehingga perlu diakhiri dengan suatu karakter null, yang ditulis sebagai **`0`** atau **`'\0'`**.

Sehingga dapat digambarkan sebagai berikut :



Initialization of strings

Karena sebenarnya karakter string adalah array, sehingga cara mengisinya sama dengan array biasanya, contoh :

```
char mystring[] = { 'H', 'e', 'l', 'l', 'o', '\0' };
```

Dalam hal ini, kita akan mendeklarasikan suatu karakter string (array) yang terdiri dari 6 elemen type **`char`** dengan nilai **`Hello`** diikuti dengan suatu karakter null **`'\0'`**.

Berikut ini adalah cara lain untuk deklarasi karakter string dengan nama **`mystring`** dengan hasil yang sama :

```
char mystring [] = { 'H', 'e', 'l', 'l', 'o', '\0' };
char mystring [] = "Hello";
```

Assigning values to strings

Karena merupakan suatu array, untuk memberikan nilai kepada karakter string kita dapat menggunakan metode berikut:

```
mystring[0] = 'H';
mystring[1] = 'e';
mystring[2] = 'l';
mystring[3] = 'l';
mystring[4] = 'o';
mystring[5] = '\0';
```

Tetapi tentu saja cara demikian adalah sangat tidak praktis. Umumnya untuk memberikan nilai pada suatu array, khususnya untuk karakter string kita dapat menggunakan fungsi seperti **`strcpy`**. **strcpy** (**str**ing **c**o**py**) dimana didefinisikan dalam library **`cstring`** (`string.h`) yang dapat digunakan dengan penulisan berikut:

```
strcpy (string1, string2);
```

Hal ini akan menduplikasi isi dari *`string2`* kedalam *`string1`*. *`string2`* dapat berupa suatu array, suatu pointer, atau suatu constant string, contoh :

```
strcpy (mystring, "Hello");
```

Contoh :

```
// setting value to string
#include <iostream.h>
#include <string.h>

int main ()
{
  char szMyName [20];
  strcpy (szMyName,"J. Soulie");
  cout << szMyName;
  return 0;
}
```

```
J. Soulie
```

Perhatikan bahwa kita perlu melakukan include **`<string.h>`** pada bagian header agar dapat digunakan function **`strcpy`**.

Walaupun kita juga dapat menulis suatu fungsi sederhana seperti **`setstring`** berikut dengan operasi yang menyerupai **`strcpy`**:

```
// setting value to string
#include <iostream.h>
```

```
J. Soulie
```

```
void setstring (char szOut [], char szIn [])
{
  int n=0;
  do {
    szOut[n] = szIn[n];
  } while (szIn[n++] != '\0');
}

int main ()
{
  char szMyName [20];
  setstring (szMyName,"J. Soulie");
  cout << szMyName;
  return 0;
}
```

Metode lain yang sering dipergunakan untuk memasukan nilai ke suatu array adalah dengan langsung memasukannya melalui input stream (**cin**). Dalam hal ini nilai string diberikan oleh user pada saat eksekusi program.

Ketika **cin** digunakan dengan string atau karakter biasanya menggunakan metode **getline**, yang dapat dipanggil dengan prototype berikut:

```
cin.getline ( char buffer[], int length, char delimiter = '
\n');
```

dimana ***buffer*** adalah alamat untuk menyimpan input (dalam hal ini adalah array), ***length*** adalah panjang maksimum dari buffer (ukuran dari array) dan ***delimiter*** adalah karakter yang digunakan untuk mengakhiri user input, dimana defaultnya adalah newline character (**'\n'**).

Berikut ini adalah contoh sederhana pemakaian **cin.getline** pada string:

```
// cin with strings
#include <iostream.h>

int main ()
{
  char mybuffer [100];
  cout << "What's your name? ";
  cin.getline (mybuffer,100);
  cout << "Hello " << mybuffer << ".\n";
  cout << "Which is your favourite team? ";
  cin.getline (mybuffer,100);
  cout << "I like " << mybuffer << " too.\n";
  return 0;
}
```

```
What's your name? Juan
Hello Juan.
Which is your favourite team?
Inter Milan
I like Inter Milan too.
```

Sebenarnya kita dapat saja menggunakan perintah berikut untuk membaca string ke suatu variable array:

```
cin >> mybuffer;
```

Tetapi jika dibandingkan dengan **`cin.getline`**, perintah diatas memiliki keterbatasan:

- Hanya dapat menerima satu kata tunggal (bukan kalimat lengkap) karena metode ini menggunakan karakter kosong sebagai delimeternya, seperti spasi, tabulator,.
- Tidak dimungkinkan untuk membatasi ukuran buffer. Hal ini akan membuat program anda menjadi tidak stabil jikalau input oleh user melebihi ukuran array.

Untuk alas an ini, kami menyarankan anda untuk menggunakan **`cin.getline`** sebagai penganti dari **`cin >>`**.

Converting strings to other types

Kadang-kadang kita perlu melakukan konversi string ke type numerik. Misalnya suatu string seperti **`"1977"`**, dan kita ingin mengkonversinya ke suatu type data integer, untuk keperluan tersebut kita membutuhkan library **`cstdlib`** (**`stdlib.h`**) yang mengandung tiga fungsi untuk keperluan ini :

- **atoi:** melakukan konversi string ke type **`int`**.
- **atol:** melakukan konversi string ke type **`long`**.
- **atof:** melakukan konversi string ke type **`float`**.

Semua fungsi diatas membutuhkan satu parameter dan mengembalikan suatu nilai berdasarkan tyoe yang dikehendaki (`int`, `long` or `float`). contoh:

```
// cin and ato* functions
#include <iostream.h>
#include <stdlib.h>

int main ()
{
  char mybuffer [100];
  float price;
  int quantity;
  cout << "Enter price: ";
  cin.getline (mybuffer,100);
  price = atof (mybuffer);
  cout << "Enter quantity: ";
  cin.getline (mybuffer,100);
  quantity = atoi (mybuffer);
  cout << "Total price: " << price*quantity;
  return 0;
}
```

```
Enter price: 2.75
Enter quantity: 21
Total price: 57.75
```

Functions to manipulate strings

Library **cstring** (`string.h`) mendefinisikan banyak fungsi untuk melakukan operasi manipulasi terhadap string pada C (seperti yang diterangakan pada strcpy). Berikut ini adalah ringkasan fungsi yang sering digunakan:

strcat: `char* strcat (char* dest, const char* src);`
Menambahkan *src* string pada akhir dari *dest* string. Mengembalikan *dest*.

strchr: `char* strchr (const char* string, int c );`
Mengembalikan pointer pada karakter pertama ditemukan dan null jika tidak ditemukan.

strcspn: `size_t strcspn ( const char * string1, const char * string2 );`
Mencari dalam *string1* character demi character, mengembalikan posisi pertama yang mengandung salah satu character pada string 2, fungsi akan mengembalikan nilai panjang dari *string1* kalau tidak ada character *string2* didalam *string1*, karena masing-masing diakhiri dengan null.

Example.

```
/* strcspn example */
#include <stdio.h>
#include <string.h>

int main ()
{
  char str1[] = "fcba73";
  char str2[] = "1234567890";
  int i;
  i = strcspn (str1,str2);
  printf ("The first number in str1 is str1[%d]\n",i);
  return 0;
}
```

Output:
```
The first number in str1 is str1[4]
```

strcmp: `int strcmp (const char* string1, const char* string2);`
Membandingkan string *string1* dan *string2*. Mengembalikan `0` jika keduanya serupa.

strcpy: `char* strcpy (char* dest, const char* src);`
Menduplikasi isi dari *src* ke *dest*. Mengembalikan *dest*.

strlen: `size_t strlen (const char* string);`
Mengembalikan panjang dari *string*.

CATATAN: **`char*`** adalah sama dengan **`char[]`**

Latihan

1. Buatlah program untuk mengkonversi Desimal ke Biner, Octal dan Hexa dengan menggunakan C null terminated character string.

```
//Program konversi Descimal ke Hexa
//Oleh : Hendra Soewarno
//Dengan memanfaatkan C null terminated character string

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

void dectohex(int dec, char hex[]);

int main()
{
char hex[10]="";
clrscr();
dectohex(1000,hex);
cout << hex;
return 0;
}

void dectohex(int dec, char *hex)
{
char
digit[]={'0','1','2','3','4','5','6','7','8','9','A','B','C','D',
'E','F'};
while (dec > 0)
      {
      *hex = digit[dec%16];
      dec = dec / 16;
      hex++;
      }
*hex = '\0';                    //character null terminate
}
```

2. Buatlah program untuk mengkonversi bilangan Binary, Octal dan Hexa ke Desimal.

Section 10
Pointers

Kita telah melihat bagaimana variable-variable yang berada dalam sel di memory dapat kita akses dengan suatu identifier (nama variabel). Secara fisik variabel tersebut i disimpan pada tempat tertentu di memori.

Suatu contoh dari memori komputer adalah suatu jalan didalam kota. Pada semua rumah diatas jalan tersebut dinomori secara unik, jika kita ingin ke alamat Jl. Jurung No. 6, maka kita dapat dengan mudah menemukannya.

Demikian juga sistem operasi akan mengorganisasikan memori dengan nomor berurutan secara unik, jika kita berbicara tentang lokasi 1234 dalam memori, maka hanya ada satu lokasi dengan nomor tersebut.

Address (dereference) operator (`&`).

Ketika kita mendeklarasikan suatu variabel, kita tidak menentukan dimana variabel tersebut akan ditempatkan dimemori. Hal tersebut akan dilakukan oleh compiler dan sistem operasi pada saat runtime.

Kadang-kadang kita ingin mengetahui alamat dimana variabel kita ditempatkan, hal tersebut dapat dilakukan dengan mengawali identifier variabel tersebut dengan suatu *ampersand sign* (`&`), dimana dibaca sebagai ***"address of"***. contoh:

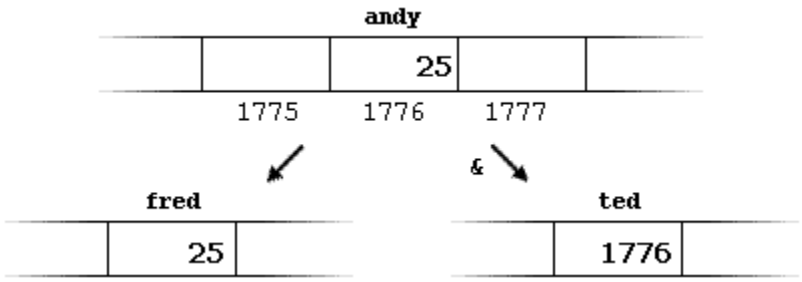
```
ted = &andy;
```

akan mengisi variabel **`ted`** dengan alamat dari variabel **`andy`**.

Misalnya variabel **`andy`** ditempatkan pada alamat memori **`1776`** dan penulisan berikut:

```
andy = 25;
fred = andy;
ted = &andy;
```

akan menghasilkan seperti diagram berikut :



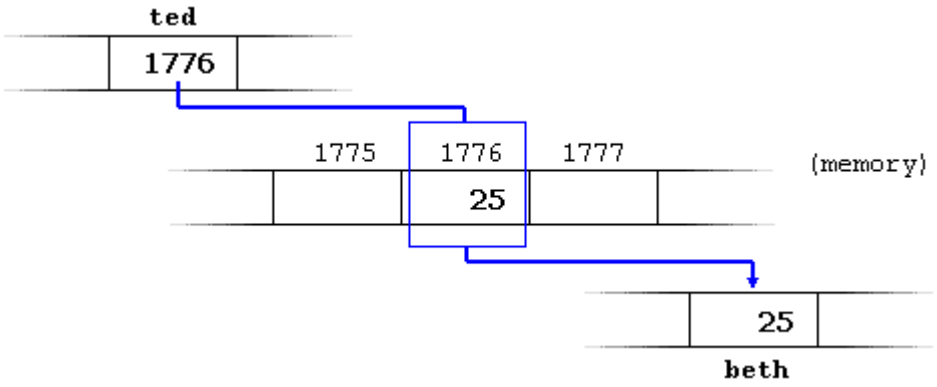
Variabel yang menyimpan alamat dari variabel lain (seperti **`ted`** pada contoh sebelumnya) kita sebut sebagai **pointer**. Pointer pada C++ memiliki beberapa keunggulan dan sering digunakan.

Reference operator (*)

Dengan menggunakan suatu pointer kita dapat langsung mengakses nilai dari alamat yang tersimpan pada variabel dengan mengawalinya dengan operator *asterisk* (*), yang dibaca sebagai ***"value pointed by"***. lanjutan dari contoh sebelumnya:

```
beth = *ted;
```

(dimana dapat dibaca sebagai: "beth sama dengan value pointed by ted") **`beth`** akan berisi nilai **`25`**, karena **`ted`** adalah **`1776`**, dan nilai yang ditunjuk oleh **`1776`** adalah **`25`**.



Operator of address or dereference (`&`)
Digunakan untuk mengawali suatu variabel dan di baca sebagai **"address of"**, sehingga: `&variable1` dapat dibaca sebagai *"address of `variable1`"*

Operator of reference (`*`)
Menunjukan apa yang terdapat pada alamat yang ditunjuk oleh suatu variabel. Dibaca sebagai **"value pointed by"**.

Declaring variables of type pointer

Sehubungan dengan kemampuan suatu pointer untuk secara langsung menunjuk pada nilai yang ditunjuk, sehingga perlu ditentukan jenis data type yang ditunjuk oleh sebuah pointer ketika mendeklareasikannya. Adalah tidak sama pointer pada suatu **`char`** sebagaimana pointer pada **`int`** dan **`float`**.

Deklarasi pointer mengikuti bentuk berikut:

```
type * pointer_name;
```

dimana ***type*** adalah type dari data yang ditunjuk, bukan type dari pointer itu sendirinya, contoh:

```
int * number;
char * character;
float * greatnumber;
```

Perlu ditekankan bahwa pemakaian asterisk (*) pada saat deklarasi pointer menunjukan bahwa itu adalah pointer

Berikut ini adalah contoh pemakaian pointer :

```
// more pointers
#include <iostream.h>

int main ()
{
  int value1 = 5, value2 = 15;
  int *p1, *p2;

  p1 = &value1;     // p1 = address of value1
  p2 = &value2;     // p2 = address of value2
  *p1 = 10;         // value pointed by p1 = 10
  *p2 = *p1;        // value pointed by p2 = value pointed by p1
  p1 = p2;          // p1 = p2 (value of pointer copied)
  *p1 = 20;         // value pointed by p1 = 20
  cout << "value1==" << value1 << "/ value2==" << value2;
  return 0;
}
```

```
value1==10 /
value2==20
```

Pointers and arrays

Suatu array dapat juga diakses dengan menggunakan pointer, perhatikan contoh berikut :

```
int numbers [20];
int * p;
```

dan penulisan berikut:

```
p = numbers;
```

Dalam hal ini **p** akan menunjuk pada akan menunjuk pada elemen pertama dari **numbers**

sehingga pada contoh berikut dapat dilakukan hal berikut :

```
// more pointers
#include <iostream.h>

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] << ", ";
  return 0;
}
```

```
10, 20, 30, 40, 50,
```

Pointer initialization

Ketika mendeklarasikan pointer, kita dapat melakukan secara explicit dengan langsung menunjuk pada variabel yang kita inginkan,

```
int number;
int *tommy = &number;
```

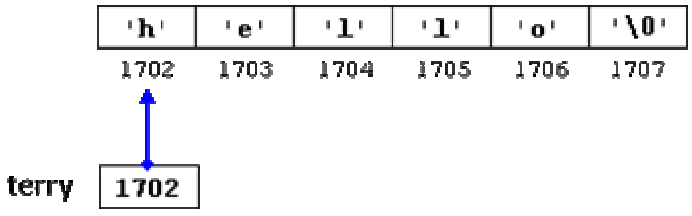
hal ini sama dengan penulisan:

```
int number;
int *tommy;
tommy = &number;
```

Sebagaimana array, compiler juga memungkinkan kasus special dimana kita ingin menginisialisasi isi dari ponter pada saat deklarasi variabel pointer:

```
char * terry = "hello";
```

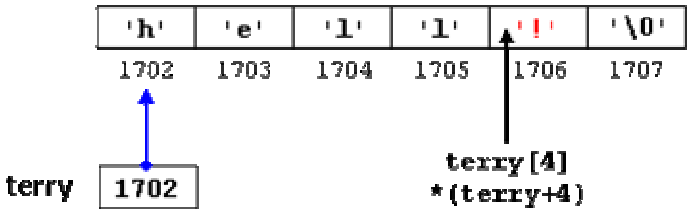
dalam hal ini static storage dicadangkan untuk menyimpan `"hello"` dan suatu pointer pada **`char`** pertama dari memori blok (dalam hal ini adalah 'h') di assign ke **`terry`**. Perhatikan gambar berikut :



Dan kedua penulisan berikut adalah sah :

```
terry[4] = '!';
*(terry+4) = '!';
```

dalam hal ini penulisan **`terry[4]`** adalah sama dengan menulis **`*(terry+4)`**:



Arithmetic of pointers

Operasi aritmatika yang dilakukan terhadap suatu pointer akan berbeda berdasarkan type datanya, hal ini sangat tergantung pada ukuran byte dari masing-masing type data.

Misalnya :

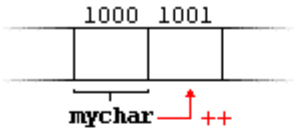
```
char *mychar;
short *myshort;
long *mylong;
```

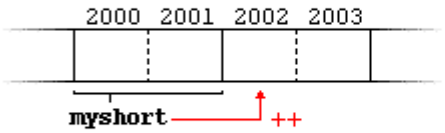
dan bayangkan masing-masing menunjuk pada lokasi memori **`1000`**, **`2000`** dan **`3000`** secara berurutan.

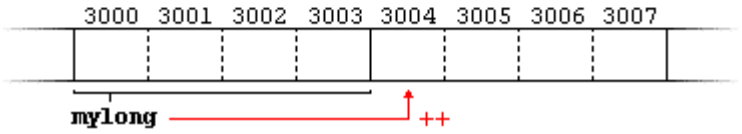
operasi berikut :

```
mychar++;
myshort++;
mylong++;
```

`akan menyebabkan mychar menjadi nilai 1001, myshort` menjadi `2002`, dan `mylong` menjadi `3004`.







Hal ini juga berlaku untuk penulisan dalam bentuk :

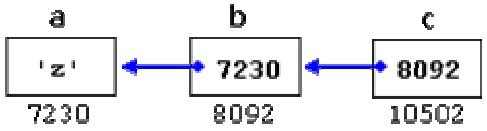
```
mychar = mychar + 1;
myshort = myshort + 1;
mylong = mylong + 1;
```

Pointers to pointers

C++ juga memungkinkan pemakaian pointer untuk menunjuk pada pointer, dalam hal ini berarti menunjuk pada data. Dalam hal ini kita perlu menambah tanda asterisk (*) pada setiap tingkat referensi:

```
char a;
char * b;
char ** c;
a = 'z';
b = &a;
c = &b;
```

Dalam hal ini bayangkan memori dari masing-masing variabel adalah **7230**, **8092** dan **10502**, dapat digambarkan berikut:



(didalam sel terdapat isi dari variabel)

Sesuatu yang baru dari contoh ini adalah variabel **c**, dimana kita dapat membicarakannya dalam tiga cara yang berbeda, dimana masing-masing dari nilainya akan bersesuaian dengan nilai yang berbeda:

```
c adalah variabel type (char **) dengan nilai 8092
*c adalah varibel type (char*) dengan nilai 7230
**c adalah variabel type (char) dengan suatu nilai'z'
```

Pertanyaan

1. Operator apa yang digunakan untuk mendapatkan alamat dari suatu variabel?
2. Operator apa yang digunakan untuk mendapatkan nilai yang tersimpan pada alamat yang tersimpan dalam pointer?
3. Apa yang dimaksud dengan pointer?
4. Apa perbedaan antara alamat yang tersimpan pada suatu pointer dengan nilai pada alamat tersebut?

5. Apa perbedaan antara indirection operator dan `address of` operator?
6. Apa perbedaan antara `const int * ptrOne` dan `int * const ptrTwo`?

Latihan

1. Buatlah sebuah function untuk menghitung jumlah kata dari string yang diberikan dengan fasilitas pointer.
2. Buatlah function untuk membuat setiap huruf pertama dari string yang diberikan menjadi huruf besar dengan fasilitas pointer.
3. Buatlah program yang menyimpan nilai yang dimasukan oleh pemakai, urut nilai tersebut secara descending, dan coba temukan nilai yang dimasukan oleh pemakai.

Section 11
Advanced pointers

void pointers

Pointer type *void* adalah suatu pointer type khusus. *void* pointers dapat menunjuk pada sembarang type data, dari suatu nilai integer ke suatu string characters. Kelemahannya adalah data yang ditunjuk tidak dapat mengacu secara langsung (kita tidak dapat menggunakan referensi asterisk `*` untuk mengoperasikan mereka), karena panjangnya tidak tertentu, dan oleh sebab itu kita akan selalu menggunakan *type casting* atau assignations untuk mengubah *void* pointer kita ke suatu pointer dari suatu concrete data type dimana kita dapat mengacu padanya.

Perhatikan contoh berikut:

```
// integer increaser
#include <iostream.h>

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;
}
```

```
6, 10, 13
```

`sizeof` adalah suatu operator yang integrasi dalam bahasa C++ yang akan mengembalikan suatu nilai konstanta dengan ukuran byte dari parameternya, maka, **`sizeof(char)`** adalah **`1`**, karena type **`char`** panjangnya 1 byte.

Pointers to functions

C++ memungkinkan operasi pointer terhadap function. Pemakaian yang paling umum adalah untuk melewatkan suatu function sebagai parameter pada fungsi lain. Untuk mendeklarasikan suatu pointer ke suatu function kita harus mendeklarasikannya seperti

prototype dari function, cuma nama function diapit dengan tanda parenthesis `()` dan sebuah pointer asterisk (`*`) disisipkan sebelum nama tersebut :

```
// pointer to functions
#include <iostream.h>

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;
}
```

```
8
```

Pada contoh diatas, **`minus`** adalah sebuah variabel global pointer pada suatu function yang memiliki dua parameter type **`int`**, hal itu langsung menunjuk pada function **`subtraction`**, semuanya dalam satu baris tunggal:

```
int (* minus)(int,int) = subtraction;
```

Stdlib for sorting and searching

QSort : void qsort (void * *base*, **size_t** *num*, **size_t width, int (*** *fncompare*)(**const void *, const void *));**

Function ini menggunakan implementasi algoritma QuickSort untuk mengurut elemen yang berada dalam suatu array pada pointer base, masing-masing elemen harus memiliki ukuran sama.

lsearch : **`void* lsearch ( const void *`** *`key`*, **`void *`** *`base`*, **`size_t`** *`num`*, **`size_t`** *`width`*, **`int (*`***`fncompare`***`)(const void *, const void * ) );`**

Mencari key dalam array pada pointer base dengan metode linier search, dan akan mengembalikan alamat pertama elemen yang berhasil ditemukan.

bsearch : `void*  bsearch ( const void * key, const void * base, size_t num, size_t width, int (*fncompare)(const void *, const void * ) );`

Mencari key dalam array pada pointer base dengan metode binary search, dan akan mengembalikan alamat pertama elemen yang berhasil ditemukan.

Adapun paramater dari masing-masing function diatas adalah :

key
: Pointer ke objek sebagai key yang akan dicari.

base
: Pointer ke base dari array dimana pencarian akan dilakukan.

num
: Jumlah elemen dalam array yang di tunjuk *base*.

width
: Ukuran dalam byte masing-masing elemen dari array tersebut.

fncompare
: Function untuk membandingkan dua elemen. Hal ini harus disediakan oleh programmer dengan deklarasi sebagai berikut :

int *fncompare* **(const void *** *elem1***, const void *** *elem2* **);**

Function ini akan menerima dua parameter (*elem1* dan *elem2*) yang menunjuk pada elemen, dan harus mengembalikan suatu nilai `integer` sebagai hasil perbandingan:

return value	description
<0	*elem1 < *elem2
0	*elem1 == *elem2
>0	*elem1 > elem2

Contoh :

```
/* qsort, lsearch, and bsearch example */
#include <stdio.h>
#include <stdlib.h>

int values[] = { 40, 10, 100, 90, 20, 25 };

int compare (const void * a, const void * b)
{
  return ( *(int*)a - *(int*)b );
}

int main ()
{
  int * pItem;
```

```
  int n;
  qsort (values, 6, sizeof(int), compare);
  for (n=0; n<6; n++)
  {
    printf ("%d ",values[n]);
  }

  pItem = (int*) lsearch (&key, values, 6, sizeof (int), compare);
  if (pItem!=NULL)
    printf ("%d using linear search is in the array",*pItem);
  else
    printf ("%d is not in the array",key);


  int key = 40;
  pItem = (int*) bsearch (&key, values, 6, sizeof (int), compare);
  if (pItem!=NULL)
    printf ("%d using binary search is in the array",*pItem);
  else
    printf ("%d is not in the array",key);

  return 0;
}
```

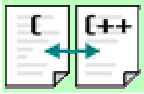
Output:

```
10 20 25 40 90 100
40 using linear search is in the array
40 using binary search is in the array
```

Section 12
Dynamic memory.

Kadang-kadang kita membutuhkan variabel dengan ukuran yang baru dapat ditentukan pada saat runtime, misalnya kita menanyakan jumlah data pada user, mengalokasikan memori untuk menyimpan data.

Dalam hal ini kita membutuhkan *dynamic memory*, dimana pada C++ dapat digunakan operator *new* dan *delete*.


Operator ***new*** dan ***delete*** adalah khusus untuk C++.

Operators *new* and *new[ ]*
Dalam usaha untuk memesan dynamic memory, digunakan operator **new**. *new* diikuti dengan suatu data *type* dan secara optional jumlah elemen yang diperlukan dalam brackets `[]`. Akan mengembalikan suatu pointer pada awal dari blok memori yang berhasil dialokasi. Bentuknya adalah:

```
pointer = new type              // deklarasi 1 elemen
```

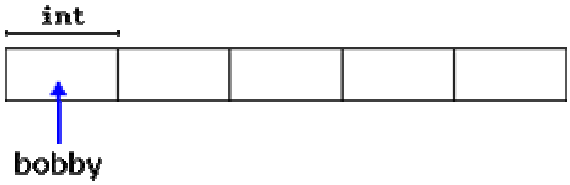
atau

```
pointer = new type [elements] // deklarasi array
```

Contoh:

```
int * bobby;
bobby = new int [5];
```

dalam hal ini, akan dialokasi 5 elemen dengan type **`int`** pada suatu heap dan mengembalikan sebuah pointer yang menunjuk pada alamat awal ke variabel **`bobby`**.



Sesuatu hal yang perlu diperhatikan bahwa dynamic memori tidak selamanya akan berhasil dialokasi karena kehabisan memori. Jika hal ini terjadi maka permintaan operator **`new`**, akan mengembalikan suatu null pointer. Untuk alasan tersebut, maka disarankan untuk selalu memeriksa apakah pointer yang dikembalikan adalah NULL setelah pemanggilan **`new`**.

```
int * bobby;
bobby = new int [5];
if (bobby == NULL) {
  // error assigning memory. Take measures.
  };
```

Operator *delete*.
Jika dynamic memory yang dialokasi tidak dibutuhkan lagi, maka dapat dihapus dari memori dengan menggunakan operator **`delete`**:

```
delete pointer;
```

or

```
delete [] pointer;
```

Contoh Program :

```
//Program mencari rata-rata
//Oleh : Hendra Soewarno
//Dengan memanfaatkan fasilitas alokasi dinamis new dan delete

#include <iostream.h>
#include <stdlib.h>

int *pdat;
int jd;

void tanyajd();
void alokasimemory();
void bacadata();
void hitungrata();
void dealokasimemory();

int main()
{
tanyajd();
alokasimemory();
bacadata();
hitungrata();
dealokasimemory();
return 0;
}

void tanyajd()
{
cout << "jumlah data :";
cin >> jd;
}

void alokasimemory()
{
pdat = new int[jd];
if (pdat == NULL)
```

```
    {
    cout << "gagal alokasi memory"<<endl;
    exit(0);
    }
}

void bacadata()
{
for (int i=0;i<jd;i++) {
     cout << "data ke -" << i+1 << " : ";
     cin >> pdat[i];
     }
}

void hitungrata()
{
float rata;
rata = 0;
for (int i=0;i<jd;i++)
     rata = rata + (float) pdat[i]/jd;
cout << "nilai rata-rata :"<<rata<<endl;
}

void dealokasimemory()
{
delete pdat;
}
```

Dynamic memory in ANSI-C

Operator *new* dan *delete* hanya terdapat pada C++ dan tidak tersedia pada C language. Pada C language, untuk mengalokasi dynamic memory kita dapat menggunakan library **`stdlib.h`**. Kita dapat juga menggunakannya dalam C++.

The function *malloc*
Ini adalah fungsi generic untuk alokasi dynamic memory pada pointer. Adapun prototype-nya adalah:

```
void * malloc (size_t nbytes);
```

dimana `nbytes` adalah jumlah byte yang akan diberikan ke pointer. Function ini akan mengembalikan sebuah pointer dengan type `void*`, oleh karena itu kita perlu melakukan *type cast* terhadap nilai pointer yang ingin dihasilkan, contoh:

```
char * ronny;
ronny = (char *) malloc (10);
```

Hal ini akan memberikan pada pointer `ronny` suatu blok 10 byte yang dapat digunakan.

Jika kita ingin memberikan suatu blok data dengan type data yang bukan char (berukuran lebih dari 1 byte) kita harus mengalikan jumlah elemen yang diinginkan dengan ukuran elemen dari tiap-tiap elemen, dalam hal ini kita dapat menggunakan operator *sizeof* :

```
int * bobby;
bobby = (int *) malloc (5 * sizeof(int));
```

Kode ini akan memberikan pada bobby sebuah pointer yang terdiri dari 5 data integer.

The function *calloc*.
calloc hampir menyerupai operasi malloc, perbedaan utamanya adalah pada prototype:

```
void * calloc (size_t nelements, size_t size);
```

menggunakan dua parameter, yang pertama (*nelements*) adalah jumlah elemen dan kedua (*size*) adalah ukuran masing-masing elemen. Alokasi untuk bobby dapat ditulis menjadi:

```
int * bobby;
bobby = (int *) calloc (5, sizeof(int));
```

Perbedaan lain antara **malloc** dan **calloc** adalah **calloc** menginisialisasi semua elemennya menjadi 0.

The function *realloc*.
Mengubah ukuran dari suatu blok memori yang telah diberikan pada suatu pointer.

```
void * realloc (void * pointer, size_t size);
```

pointer parameter menerima sebuah pointer yang telah diberikan suatu blok memori atau null pointer, dan *size* menentukan ukuran baru dari memori blok yang diinginkan. Dalam hal ini isi pointer lama tidak akan berubah. Jika proses alokasi tambahan gagal akan mengembalikan suatu null pointer, sedangkan isi pointer sebelumnya tidak berubah.

The function *free*.
Membebaskan blok dynamic memory yang sebelumnya telah diberikan dengan ***malloc***, ***calloc*** atau ***realloc***.

```
void free (void * pointer);
```

Fungsi ini hanya dapat digunakan untuk membebaskan memori yang diberikan dengan fungsi ***malloc***, ***calloc*** dan ***realloc***.

Contoh program :

```
//Program mencari rata-rata
//Oleh : Hendra Soewarno
```

```
//Dengan memanfaatkan fasilitas alokasi dinamis malloc dan free

#include <iostream.h>
#include <stdlib.h>

int *pdat;
int jd;

void tanyajd();
void alokasimemory();
void bacadata();
void hitungrata();
void dealokasimemory();

int main()
{
tanyajd();
alokasimemory();
bacadata();
hitungrata();
dealokasimemory();
return 0;
}

void tanyajd()
{
cout << "jumlah data :";
cin >> jd;
}

void alokasimemory()
{
pdat = (int*) malloc(jd * sizeof(int));
//pdat = (int*) calloc(jd, sizeof(int));
if (pdat == NULL)
   {
   cout << "gagal alokasi memory"<<endl;
   exit(0);
   }
}

void bacadata()
{
for (int i=0;i<jd;i++) {
    cout << "data ke -" << i+1 << " : ";
    cin >> pdat[i];
    }
}

void hitungrata()
{
float rata;
rata = 0;
for (int i=0;i<jd;i++)
    rata = rata + (float) pdat[i]/jd;
cout << "nilai rata-rata :"<<rata<<endl;
}
```

```
void dealokasimemory()
{
free(pdat);
}

//Program mencari rata-rata
//Oleh : Hendra Soewarno
//Dengan memanfaatkan fasilitas alokasi dinamis realloc dan free

#include <iostream.h>
#include <stdlib.h>

int *pdat;
int jd;

void bacadata();
void hitungrata();
void dealokasimemory();

int main()
{
bacadata();
hitungrata();
dealokasimemory();
return 0;
}

void bacadata()
{
int dat;
jd = 0;
do {
    cout << "data ke -" << jd+1 << " : ";
    cin >> dat;
    if (!dat==0) {
       if (realloc(pdat,dat*sizeof(int)) == NULL) {
        cout << "gagal alokasi memory !"<<endl;
        exit(0);
        }
       else {
        pdat[jd] = dat;
        jd++;
        }
       }
   } while (!dat==0);
}

void hitungrata()
{
float rata;
rata = 0;
for (int i=0;i<jd;i++)
    rata = rata + (float) pdat[i]/jd;
cout << "nilai rata-rata :"<<rata<<endl;
}
```

```
void dealokasimemory()
{
free(pdat);
}
```

Pertanyaan

1. Bagaimana mengalokasikan memory dinamis ?
2. Bagaimana kita mengetahui suatu alokasi memori berhasil atau tidak ?
3. Terangkan perbedaan cara untuk alokasi memori dinamis pada C dan C++!
4. Bagaimana kita mengetahui ukuran dari suatu data type ?

Latihan

1. Buatlah suatu program untuk menyimpan data yang dimasukan oleh pemakai, urut data tersebut dalam urutan descending, kemudian coba cari data yang dimasukan oleh pemakai. Petunjuk : Tanyakan jumlah data terlebih dahulu dengan memanfaatkan alokasi memori dinamis.

Section 13
Structures

Suatu structure adalah suatu himpunan dari berberapa type data yang berbeda dikelompokan menjadi satu deklarasi yang unik. Bentuknya adalah sebagai berikut:

```
struct model_name {
  type1 element1;
  type2 element2;
  type3 element3;
  .
  .
} object_name;
```

dimana *model_name* adalah nama untuk model structure type dan *object_name* bersifat optional sebagai variabel dengan structure object tersebut. Diantara curly brackets { } adalah type dan sub-identifiers yang bersesuaian dengan elemen-elemen yang membentuk structure tersebut.

Contoh :

```
struct products {
  char name [30];
  float price;
} ;

products apple;
products orange, melon;
```

Dimana **apple**, **orange**, dan **melon** adalah variabel yang memiliki structure **products**.

atau dapat juga ditulis :

```
struct products {
  char name [30];
  float price;
} apple, orange, melon;
```

Dalam hal ini masing-masing variabel (**apple**, **orange** and **melon**) memiliki member **name** dan **price**., dimana penulisannya adalah nama variabel dan nama member dipisahkan dengan suatu point (.) :

```
apple.name
apple.price
orange.name
orange.price
melon.name
melon.price
```

Contoh program :

```
//Program data terstruktur
//Oleh : Hendra Soewarno
//Dengan memanfaatkan fasilitas struct.

#include <iostream.h>
#include <iomanip.h>
#include <string.h>
#include <conio.h>

struct siswa
{
  char nim[10];
  char nama[20];
  float teori,praktek;
} dsiswa[100];

int jd;                    // global variabel

void tanyajd();
void bacadata();
void cetakdata();

int main()
{
clrscr();
tanyajd();
bacadata();
cetakdata();
return 0;
}

void tanyajd()
{
cout << "jumlah data :";
cin >> jd;
}

void bacadata()
{
int i;
char nim[10];

cin.ignore();
for (i=0;i<jd;i++) {
   cout << "Record ke -" << i+1 << endl;
   cout << "Nim    : "; cin.getline(nim,10);
   if (nim[0] != 0) {
      strcpy(dsiswa[i].nim,nim);
      cout << "Nama   : "; cin >> dsiswa[i].nama;
      cout << "Teori  : "; cin >> dsiswa[i].teori;
      cout << "Praktek: "; cin >> dsiswa[i].praktek;
      cin.ignore();
      }
   }
```

```
}

void cetakdata()
{
int i;
cout << "--------------------------------"<< endl;
cout << " Nim   Nama      Teori  Praktek "<< endl;
cout << "--------------------------------"<< endl;
for (i=0;i<jd;i++) {
      cout << setw(5) << dsiswa[i].nim;
      cout << setw(20) << dsiswa[i].nama;
      cout << setw(5) << setprecision(2) << dsiswa[i].teori;
      cout << setw(5) << setprecision(2) << dsiswa[i].praktek<<endl;
      }
}
```

Pointers to structures

Menyerupai type lain, structure dapat ditunjuk dengan menggunakan pointer. Aturannya adalah sama dengan type data lainnya, yaitu Pointer tersebut perlu dideklarasikan sebagai pointer pada structure tersebut:

```
struct siswa
{
  char nim[10];
  char nama[20];
  float teori,praktek;
} dsiswa[100];

siswa *psiswa;
```

Dalam hal ini **amovie** adalah suatu obejk dari structure **movies_t** dan **pmovie** adalah sebuah pointer ke objek type **movies_t**. Maka penulisan berikut adalah sah:

```
psiswa = &siswa[0];
```

Ok, perhatikan contoh berikut :

```
//Program data terstruktur
//Oleh : Hendra Soewarno
//Dengan memanfaatkan fasilitas struct dan pointer.

#include <iostream.h>
#include <iomanip.h>
#include <string.h>
#include <conio.h>

struct siswa
{
  char nim[10];
  char nama[20];
  float teori,praktek;
} dsiswa[100];
```

```
siswa *psiswa;

int jd;


void tanyajd();
void bacadata();
void cetakdata();

int main()
{
clrscr();
tanyajd();
bacadata();
cetakdata();
return 0;
}

void tanyajd()
{
cout << "jumlah data :";
cin >> jd;
}

void bacadata()
{
int i;
char nim[10];

cin.ignore();
for (i=0;i<jd;i++) {
   psiswa = &dsiswa[i];
   cout << "Record ke -" << i+1 << endl;
   cout << "Nim    : "; cin.getline(nim,10);
   if (nim[0] != 0) {
      strcpy(psiswa->nim,nim);
      cout << "Nama   : "; cin >> psiswa->nama;
      cout << "Teori  : "; cin >> psiswa->teori;
      cout << "Praktek: "; cin >> psiswa->praktek;
      cin.ignore();
      }
   }
}

void cetakdata()
{
int i;
cout << "--------------------------------"<< endl;
cout << " Nim   Nama      Teori  Praktek "<< endl;
cout << "--------------------------------"<< endl;
for (i=0;i<jd;i++) {
      psiswa = &dsiswa[i];
      cout << setw(5) << psiswa->nim;
      cout << setw(20) << psiswa->nama;
      cout << setw(5) << setprecision(2) << psiswa->teori;
      cout << setw(5) << setprecision(2) << psiswa->praktek<<endl;
```

```
        }
}
```

Pada contoh diatas kita menemukan operator **->**. Operator ini digunakan khusus oleh pointer untuk menunjuk ke member dari suatu structure, contoh :

```
psiswa->nim
```

atau dapat diterjemahkan menjadi:

```
(*psiswa).nim
```

baik **psiswa->nim** dan **(*psiswa).nim** adalah sah yang berarti kita ingin mengevaluasi field **nim** dari structure yang ditunjuk oleh **psiswa**.

Nesting structures

Structures dapat juga di nested yang berarti bahwa suatu structure dapat memiliki elemen yang berupa structure

```
struct ujian
{
float teori,praktek;
};

struct siswa
{
  char nim[10];
  char nama[20];
  ujian nilai;
} dsiswa[100];

siswa *psiswa;
```

Setelah deklarasi diatas, kita dapat menggunakan ekspresi berikut:

```
dsiswa[0].nim
dsiswa[0].nilai.teori
psiswa->nilai.praktek
```

Contoh Program

```
//Program data terstruktur
//Oleh : Hendra Soewarno
//Dengan memanfaatkan fasilitas nested struct dan pointer.

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

```
struct ujian
{
float teori,praktek;
};

struct siswa
{
  char nim[10];
  char nama[20];
  ujian nilai;
} dsiswa[100];

int jd;
siswa *psiswa;

void tanyajd();
void bacadata();
void cetakdata();

int main()              // program utama
{
clrscr();
tanyajd();
bacadata();
cetakdata();
return 0;
}

void tanyajd()
{
cout << "jumlah data :";
cin >> jd;
}

void bacadata()
{
int i;
char nim[10];

cin.ignore();
for (i=0;i<jd;i++) {
   psiswa = &dsiswa[i];
   cout << "Record ke -" << i+1 << endl;
   cout << "Nim    : "; cin.getline(nim,10);
   if (nim[0] != 0) {
      strcpy(psiswa->nim,nim);
      cout << "Nama   : "; cin >> psiswa->nama;
      cout << "Teori  : "; cin >> psiswa->nilai.teori;
      cout << "Praktek: "; cin >> psiswa->nilai.praktek;
      cin.ignore();
      }
   }
}

void cetakdata()
{
```

```
int i;
cout << "--------------------------------"<< endl;
cout << " Nim   Nama      Teori  Praktek "<< endl;
cout << "--------------------------------"<< endl;
for (i=0;i<jd;i++) {
      psiswa = &siswa[i];
      cout << setw(5) << psiswa->nim;
      cout << setw(20) << psiswa->nama;
      cout << setw(5) << setprecision(2) <<psiswa->nilai.teori;
      cout << setw(5) << setprecision(2) <<psiswa->nilai.praktek<<endl;
      }

}

//Program data terstruktur
//Oleh : Hendra Soewarno
//Dengan memanfaatkan fasilitas nested struct dan dynamic memory
allocation.

#include <iostream.h>
#include <iomanip.h>
#include <string.h>
#include <conio.h>

struct ujian
{
float teori,praktek;
};

struct siswa
{
  char nim[10];
  char nama[20];
  ujian nilai;
} *dsiswa;

int jd;
siswa *psiswa;

void tanyajd();
void alokasimemory();
void bacadata();
void cetakdata();
void dealokasimemory();

int main()              // program utama
{
clrscr();
tanyajd();
alokasimemory();
bacadata();
cetakdata();
dealokasimemory();
return 0;
}
```

```
void tanyajd()
{
cout << "jumlah data :";
cin >> jd;
}

void alokasimemory()
{
dsiswa = new siswa[jd];
}

void bacadata()
{
int i;
char nim[10];

cin.ignore();
for (i=0;i<jd;i++) {
   psiswa = &dsiswa[i];
   cout << "Record ke -" << i+1 << endl;
   cout << "Nim    : "; cin.getline(nim,10);
   if (nim[0] != 0) {
      strcpy(psiswa->nim,nim);
      cout << "Nama   : "; cin >> psiswa->nama;
      cout << "Teori  : "; cin >> psiswa->nilai.teori;
      cout << "Praktek: "; cin >> psiswa->nilai.praktek;
      cin.ignore();
      }
   }
}

void cetakdata()
{
int i;
cout << "---------------------------------"<< endl;
cout << " Nim   Nama      Teori  Praktek "<< endl;
cout << "---------------------------------"<< endl;
for (i=0;i<jd;i++) {
      psiswa = &dsiswa[i];
      cout << setw(5) << psiswa->nim;
      cout << setw(20) << psiswa->nama;
      cout << setw(5) << setprecision(2) << psiswa->nilai.teori;
      cout << setw(5) << setprecision(2) << psiswa-
>nilai.praktek<<endl;
      }
}

void dealokasimemory()
{
delete dsiswa;

}
```

Pertanyaan

1. Apa yang dimaksud dengan suatu struct ?
2. Bagaimana mendeklarasikannya ?
3. Tulis suatu deklarasi struct untuk menyimpan alamat teman anda.

Latihan

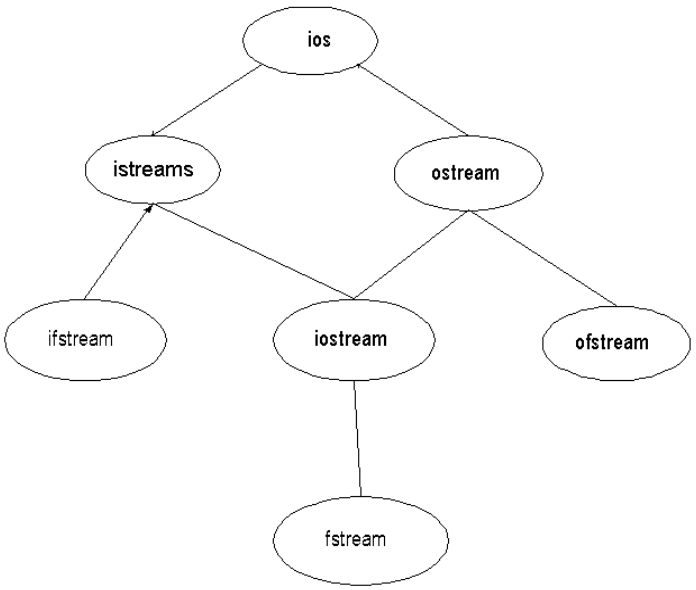
1. Tuliskan suatu program untuk menyimpan alamat dari teman anda, sediakan fasilitas tambah, perbaiki, hapus, tampil, dan cari (berdasarkan nama). Petunjuk : gunakan pendekatan pemrograman terstruktur.

Section 14

Input/Output with files

C++ mendukung proses input dan output file dengan menggunakan class-class berikut:

- **`ofstream`**: File class untuk operasi menulis (diturunkan dari `ostream`)
- **`ifstream`**: File class untuk operasi membaca (diturunkan dari `istream`)
- **`fstream`**: File class untuk operasi membaca dan menulis (diturunkan dari `iostream`)



Open a file

Operasi pertama yang dilakukan pada object ini adalah mengasosiasikannya dengan suatu file, yang mana disebutkan sebagai membuka file. File yang terbuka dinyatakan dalam program sebagai suatu objek stream (suatu instant dari salah satu class diatas) dan setiap input atau output pada object stream tersebut akan dilakukan terhadap file fisiknya

Untuk membuka suatu file dengan suatu object strean, kita dapat menggunakan fungsi **`open()`**:

```
void open (const char * filename, openmode mode);
```

dimana *`filename`* adalah suatu karakter string dari nama file yang akan dibuka dan *`mode`* adalah kombinasi dari flag berikut :

`ios::in`	Open file for reading
`ios::out`	Open file for writing

ios::ate	Initial position: end of file
ios::app	Every output is appended at the end of file
ios::trunc	If the file already existed it is erased
ios::binary	Binary mode

Flag-flag ini dapat dikombinasikan dengan menggunakan operator bitwise OR: `|`. Sebagai contoh, jika kita ingin membuka file "example.bin" dalam mode binary untuk menambah data, kita dapat melakukannya dengan memanggil fungsi **open**:

```
ofstream file;
file.open ("example.bin", ios::out | ios::app |
ios::binary);
```

Berikut adalah default mode yang akan digunakan pada masing-masing class **ofstream**, **ifstream** dan **fstream** :

class	default *mode* to parameter
ofstream	`ios::out	ios::trunc`
ifstream	`ios::in`
fstream	`ios::in	ios::out`

Default value hanya diaplikasikan jika fungsi dibuka tanpa menyebutkan suatu parameter *mode*.

Kita dapat juga menggunakan konstruktor dari masing-masing class dengan penulisan berikut :

```
ofstream file ("example.bin", ios::out | ios::app |
ios::binary);
```

Kita dapat memeriksa apakah file telah terbuka dengan benar dengan menggunakan fungsi **is_open()**:

```
bool is_open();
```

yang mana akan mengembalikan suatu nilai **bool** dengan nilai **true** jika object berasosiasi dengan suatu file yang terbuka dan sebaliknya **false**.

Closing a file

Ketika kita telah selesai membaca, dan menulis maka file tersebut harus ditutup dengan memanggil fungsi **close()**, hal ini akan mencurahkan seluruh isi buffer yang tertahan ke file yang ditutup. Bentuknya cukup sederhana:

```
void close ();
```

Begitu anggota dari fungsi ini dipanggil, objek stream dapat digunakan untuk membuka file lain, dan file tersebut juga dapat dibuka oleh proses lain.

Dalam hal ini, jika objek dihapus dengan masih adanya asosiasi file yang terbuka, maka destructor akan secara otomatis memanggil fungsi **close**.

Text mode files

Class-class **ofstream**, **ifstream** dan **fstream** adalah inheritance dari **ostream**, **istream** dan **iostream**. Oleh sebab itu objek *fstream* dapat menggunakan member yang terdapat pada class parentnya.

Umumnya, ketika menggunakan text file kita akan menggunakan member seperti pada console (yaitu **cin** dan **cout**). Pada contoh berikut kita akan menggunakan overloaded insertion operator **<<**:

```
// writing on a text file
#include <fstream.h>

int main () {
  ofstream examplefile ("example.txt");
  if (examplefile.is_open())
  {
    examplefile << "This is a line.\n";
    examplefile << "This is another line.\n";
    examplefile.close();
  }
  return 0;
}
```

file **example.txt**

```
This is a line.
This is another line.
```

Data input dari file dapat juga dilakukan dengan cara yang sama seperti yang kita lakukan pada **cin**:

```
// reading a text file
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>

int main () {
  char buffer[256];
  ifstream examplefile ("example.txt");
  if (! examplefile.is_open())
  { cout << "Error opening file"; exit (1); }

  while (! examplefile.eof() )
  {
    examplefile.getline (buffer,100);
```

```
This is a line.
This is another line.
```

```
  cout << buffer << endl;
 }
 return 0;
}
```

Binary files

Pada binary file proses input/ouput data tidak dapat menggunakan operator seperti **<<** dan **>>** serta fungsi seperti **getline**.

Untuk binary file class **fstream** memiliki prototype **write** dan **read** berikut :

```
write ( char * buffer, streamsize size );
 read ( char * buffer, streamsize size );
```

Contoh :

```
file.write((char*)&x, sizeof(x));
file.read((char*)&x, sizeof(x));
```

Dimana *buffer* adalah alamat dari memory block yang menyimpan data hasil pembacaan atau yang akan ditulis.

```
// reading binary file
#include <iostream.h>
#include <fstream.h>

const char * filename = "example.txt";

int main () {
  char * buffer;
  long size;
  ifstream file (filename, ios::in|ios::binary|ios::ate);
  size = file.tellg();
  file.seekg (0, ios::beg);
  buffer = new char [size];
  file.read (buffer, size);
  file.close();

  cout << "the complete file is in a buffer";

  delete[] buffer;
  return 0;
}
```

```
the complete file is in
a buffer
```

Verification of state flags

Selain fungsi **eof()** untuk memeriksa posisi end of file, juga disediakan beberapa member function untuk memeriksa status dari stream (semuanya akan menghasilkan nilai **bool**):

bad()
: Mengembalikan **true** kalau ada kegagalan dalam operasi membaca atau menulis

fail()
: Mengembalikan **true** dalam kasus yang sama pada **bad()** khususnya dalam kasus dimana terjadi format error, seperti mencoba membaca suatu nilai integer dan yang diterima adalah character.

eof()
: Mengembalikan **true** jika sebuah file yang terbuka untuk dibaca telah mencapai end of file.

good()
: Seperti namanya, mengembalikan **false** atau **true** berdasarkan hasil operasi terakhir.

Untuk melakukan reset terhadap state flags dapat digunakan function **clear()**.

get and *put* stream pointers

Pada semua objek i/o stream memiliki minimal satu stream pointer:

- **ifstream**, demikian juga **istream**, memiliki *get pointer* yang menunjuk pada lokasi elemen berikutnya yang akan dibaca.
- **ofstream**, demikian juga **ostream**, memiliki *put pointer* yang menunjuk pada lokasi penulisan elemen berikutnya.
- Akhirnya **fstream**, demikian juga **iostream**, memiliki *get* and *put*

Pointer-pointer Stream ini dapat dimanipulasi dengan beberapa member function berikut :

tellg() dan **tellp()**
: Mengembalikan nilai integer yang menunjukan posisi dari *get* stream pointer dan *put* stream pointer.

seekg() dan **seekp()**
: Member function ini untuk mengatur posisi get stream pointer dan *put* stream pointer dengan prototypes:

```
seekg ( pos_type position );
seekp ( pos_type position );
```

dengan nilai relatif terhadap begin of file.

atau

```
seekg ( off_type offset, seekdir direction );
seekp ( off_type offset, seekdir direction );
```

dengan nilai relatif terhadap nilai argumen parameter *direction* berikut :

ios::beg	offset specified from the beginning of the stream
ios::cur	offset specified from the current position of the stream pointer
ios::end	offset specified from the end of the stream

Contoh berikut menggunakan member fungsi diatas untuk mendapatkan ukuran dari suatu binary file:

```
// obtaining file size
#include <iostream.h>
#include <fstream.h>

const char * filename = "example.txt";

int main () {
  long l,m;
  ifstream file (filename, ios::in|ios::binary);
  l = file.tellg();
  file.seekg (0, ios::end);
  m = file.tellg();
  file.close();
  cout << "size of " << filename;
  cout << " is " << (m-l) << " bytes.\n";
  return 0;
}
```

```
size of example.txt is 40 bytes.
```

Buffers and Synchronization

Ketika kita bekerja dengan file streams, kita sebenarnya bekerja pada suatu *buffer* dari type **streambuf**. *Buffer* ini merupakan suatu memory block yang berlaku sebagai intermediasi antara stream dengan file secara fisik. Misalnya ketika function **put** (menulis sebuah karakter tunggal) dipanggil, karakter tersebut tidak secara langsung ditulis ke file fisik, tetapi disisipkan pada *buffer* untuk stream tersebut

Ketika buffer dicurahkan, semua data yang terkandung didalamnya akan ditulis ke media fisik (jika merupakan out stream) atau dihapus (jika merupakan in stream). Proses ini disebut sebagai synchronization dan terjadi dengan kondisi berikut:

- **Ketika file di tutup:** sebelum suatu file ditutup semua buffer akan di sinkron
- **Ketika *buffer* telah penuh:** *Buffer* memiliki ukuran tertentu, ketika telah penuh secara otomatis akan di sinkron.
- **Ditegaskan dengan perintah:** Ketika perintah perintah seperti **flush** dan **endl** yang menyebabkan proses sinkron.
- **Ditegaskan dengan function sync():** Pemanggilan function **sync()** (tanpa parameter) menyebabkan sinkron dilakukan.

Reading and Writing Complex Data

Walaupun metode `read` dan `write` methods mengunakan suatu `char*` pointer, tetapi tidak selamanya data yang akan dibaca atau ditulis harus berupa `char` array. Anda dapat membaca dan menulis data struct dengan menggunakan simple type casting of pointers:

```
#include "fstream.h"

struct {
  char nama[20];
  int age;
}teman;

int main()
{
  ofstream fout;
  fout.open("teman.rec",ios::out || ios::binary);
  strcpy(teman.nama,"hendra soewarno");
  teman.age = 31;
  fout.seekp(0);
  fout.write((char*)&teman, sizeof(teman));
  strcpy(teman.nama,"susan dewichan");
  teman.age = 30;
  fout.seekp(sizeof(teman));
  fout.write((char*)&teman, sizeof(teman));
  fout.close();
  return 0;
}
```

```
#include "iostream.h"
#include "fstream.h"

struct {
  char nama[20];
  int age;
}teman;

int main()
{
  ifstream fin;
  fin.open("teman.rec",ios::in || ios::binary);
  fin.seekg(0);
  fin.read((char*)&teman, sizeof(teman));
  cout << teman.nama << endl;
  cout << teman.age << endl;
  fin.seekg(sizeof(teman));
  fin.read((char*)&teman, sizeof(teman));
  cout << teman.nama << endl;
  cout << teman.age << endl;
  fin.close();
  return 0;
}
```

Contoh Program :

```
//Program random access file
//Oleh : Hendra Soewarno
//Dengan memanfaatkan fasilitas fstream.

#include <iostream.h>
#include <iomanip.h>
#include <fstream.h>
#include <string.h>

struct siswa
{
  char nim[10];
  char nama[20];
  float teori,praktek;
} dsiswa;

fstream fsource;

void bukafile(char namafile[]);
void bacadata();
void cetakdata();
void tutupfile();

int main()
{
bukafile("c:\siswa.dat");
bacadata();
cetakdata();
tutupfile();
return 0;
}

void bukafile(char namafile[])
{
fsource.open(namafile,ios::in | ios::out | ios::binary);
}

void bacadata()
{
float recno;
char nim[10];

fsource.seekg(0,ios::end);

do {
   recno = fsource.tellg()/sizeof(dsiswa);
   cout << "Record ke -" << recno+1 << endl;
   cout << "Nim    : "; cin.getline(nim,10);
   if (nim[0] != 0) {
      strcpy(dsiswa.nim,nim);
      cout << "Nama   : "; cin >> dsiswa.nama;
      cout << "Teori  : "; cin >> dsiswa.teori;
      cout << "Praktek: "; cin >> dsiswa.praktek;
      cin.ignore();
      fsource.write((char*) &dsiswa,sizeof(dsiswa));
      }
```

```
} while (nim[0] != 0);
}

void cetakdata()
{
fsource.seekg(0,ios::beg);
cout << "--------------------------------"<< endl;
cout << " Nim   Nama      Teori  Praktek "<< endl;
cout << "--------------------------------"<< endl;
while (!fsource.eof()) {
      fsource.read((char *) &dsiswa,sizeof(dsiswa));
      if (fsource.gcount() == 0) break;
      cout << setw(10) << dsiswa.nim;
      cout << setw(20) << dsiswa.nama;
      cout << setw(5) << setprecision(2) << dsiswa.teori;
      cout << setw(5) << setprecision(2) << dsiswa.praktek<<endl;
      }
}

void tutupfile()
{
fsource.close();
}
```

Pertanyaan

1. Apa tujuan pemakaian fungsi `open` pada suatu stream?
2. Tuliskan suatu deklarasi untuk suatu input stream dan hubungkan dengan suatu file yang disebut '`datain.txt`'. Bagaimana anda membaca suatu nilai real dari stream ini ke suatu vairabel `float`?
3. Tuliskan suatu deklarasi untuk suatu output stream dan hubungkan dengan suatu file yang disebut sebagai result '`results`'. Andaikan telah tersedia variabel float x dan y, bagaimana anda mencetak pesan ‘nilai dari x dan y adalah …’ kedalam file tersebut?

Latihan

1. Tuliskan sebuah program untuk menyimpan data teman anda ke dalam suatu binary file, sediakan fasilitas tambah, perbaiki, hapus, tampil, cari (dengan nama), dan fasilitas cetak ke printer. Petunjuk : gunakan suatu pendekatan pemrograman terstruktur.

Section 15
Classes

Class merupakan fasilitas baru yang diperkenalkan sejak C++ untuk mendukung pemrograman berorientasi objek. Sebuah class adalah suatu metode secara logika untuk mengorganisasi data dan fungsi dalam struktur yang sama (**encapsulation**). Mereka dideklarasikan dengan menggunakan keyword **`class`**, yang mana akan berfungsi sama seperti keyword **`struct`**, pada C tetapi dengan kemungkinan untuk mengikutkan function sebagai member, daripada hanya data.

Bentuknya adalah:

```
class class_name {
  permission_label_1:
    member1;
  permission_label_2:
    member2;
  ...
  } object_name;
```

dimana ***`class_name`*** adalah nama dari class (user defined *type*) dan field optional adalah ***`object_name`***, atau beberapa, suatu identifier objek yang sah. Badan dari deklarasi dapat mengandung ***`members`***, dimana dapat berupa deklarasi data atau function, dan secara optional ***`permission labels`***, dimana dapat berupa salah satu keyword berikut: **`private:`**, **`public:`** atau **`protected:`**. Mereka membuat acuan pada permission yang mana dapat diperoleh pada *members*

- **`private`** member-member dari suatu class yang hanya dapat diakses oleh anggota dari class yang sama atau dari class-class "*friend*" mereka.
- **`protected`** member-member yang dapat diakses akses oleh anggota dari class yang sama atau dari class-class *friend* mereka, dan juga oleh anggota class-class dari class-class *derived*.
- Akhirnya, **`public`** member-member yang dapat diakses dari semua tempat dimana class tersebut nampak.

Jika kita mendeklarasikan member-member dari suatu class tanpa mengawalinya dengan suatu permission label, maka defaultnya member-member tersebut dianggap sebagai **`private`**.

Pemakaian private maupun protected merupakan suatu konsep **data hiding** dimana untuk menghindari akses langsung terhadap member dari class yang dapat merusak integritas class.

Sebagai contoh:

```
class CRectangle {
    int x, y;
```

```
  public:
    void set_values (int,int);
    int area (void);
  } rect;
```

Mendeklarasikan class **CRectangle** dan sebuah objek dengan nama **rect**. Class ini mengandung empat member: dua variabel type **int** (**x** dan **y**) pada bagian **private** (karena private adalah default permission) dan dua function pada bagian **public**: **set_values()** dan **area()**, dimana kita hanya mencantumkan prototype-nya.

Dan untuk mengakses member-member public dari objek **rect** dapat digunakan penulisan berikut :

```
rect.set_value (3,4);
myarea = rect.area();
```

tetapi harus diingat kita tidak dapat mengakses **x** atau **y** karena mereka adalah member private, contoh kongkret dari **CRectangle**:

```
// classes example
#include <iostream.h>

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();
}
```

area: 12

Pada contoh diatas kita menemui scope operator baru `::` yang digunakan untuk deklarasi suatu member diluar dari class-nya

Penulisannya diawali dengan nama class yang memiliki member tersebut diikuti oleh scope operator (`::`) dan nama member yang bersesuaian.

Anda dapat juga dapat melengkapi member didalam class secara langsung, contoh:

```
// class example
#include <iostream.h>
```

rect area: 12
rectb area: 30

```
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;
}
```

Pada contoh diatas ukuran **`rect.area()`** dan **`rectb.area()`** akan menghasilkan ukuran yang berbeda karena mereka adalah dua objek yang berbeda dan memiliki variables **`x`** dan **`y`** masing-masing.

Catatan : Sesuatu hal yang sering dikacaukan adalah perbedaan antara Class dan Object. Pada dasarnya Object merupakan Instance dari suatu Class. Dan Class merupakan cetak biru dari suatu Object.

Constructors and destructors

Kita dapat menggunakan Constructor untuk menginisialisasi variabel maupun dynamic memory pada saat suatu objek dibuat. Dalam class, suatu function constructor mengunakan nama yang sama dengan nama class-nya. Constructor ini akan secara otomatis dijalankan ketika suatu instance baru (objek) dari class dibuat, contoh :

```
// classes example
#include <iostream.h>

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;
```

```
rect area: 12
rectb area: 30
```

```
}

int main () {
  CRectangle rect (3,4);
  CRectangle rectb (5,6);
  cout << "rect area: " << rect.area() << endl;
  cout << "rectb area: " << rectb.area() << endl;
}
```

Sehingga pada saat kita melakukan instance terhadap class tersebut menjadi:

```
CRectangle rect (3,4);
CRectangle rectb (5,6);
```

Sedangkan **Destructor** akan otomatis dipanggil pada saat suatu objek di release dari memory, atau ketika keberadaanya telah selesai (misalnya jika kita mendefinisikan objek sebagai suatu variabel lokal dalam suatu function dan function tersebut berakhir) atau di release dengan operator **`delete`**.

Destructor memiliki nama yang sama dengan class dan diawali dengan sebuah tilde (**~**) dan tidak mengembalikan nilai.

Pemakaian destructor cocok kalau pada object ada menggunakan dynamic memory during, sehingga perlu dimusnahkan pada saat objek tersebut di release dari memory.

```
// example on constructors and destructors
#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;
```

```
rect area: 12
rectb area: 30
```

```
 return 0;
}
```

Contoh program :

```
//Program bola pantul
//Oleh : Hendra Soewarno
//Hanya berjalan di Turbo C++ 3.0
//Dengan pendekatan OOP dan memanfaatkan fasilitas Graphics.

#include <graphics.h>
#include <stdlib.h>
#include <conio.h>
#include <dos.h>

class bola
{
 private :
 int x,y,dx,dy,c;             // properties

 public :
 bola(int cl);                // constructor
 ~bola();                     // destructor
 void paint(int cl);          // metoda
 void move();
}

bola::bola(int cl)
{
  x = random(640);
  y = random(480);
  dx = 10;
  dy = 10;
  c = cl;
  paint(c);
}

bola::~bola()
{
}

void bola::paint(int cl)
{
setfillstyle(1,cl);
bar(x-5,y-5,x+5,y+5);
}

void bola::move()
{
paint(0);
x = x + dx;
y = y + dy;
if (x < 10) { x = 10; dx = -dx; }
```

```
if (x > 630) { x = 630; dx = -dx; }
if (y < 10) { y = 10; dy = -dy; }
if (y > 470) { y = 470; dy = -dy; }
paint(c);
}

int main()
{
int gd,gm;
detectgraph(&gd,&gm);
initgraph(&gd,&gm,"egavga.bgi");
bola *obola[5];
int i;
for (i=0;i<5;i++) obola[i] = new bola(i);
do {
  for (i=0;i<5;i++) obola[i]->move();
  delay(100);
} while (!kbhit());

for (i=0;i<5;i++) delete obola[i];

closegraph();
return 0;
}

//Program snake
//Oleh : Hendra Soewarno
//Hanya berjalan di Turbo C++ 3.0
//Dengan pendekatan OOP dan memanfaatkan fasilitas Graphics.

#include <graphics.h>
#include <stdlib.h>
#include <conio.h>
#include <dos.h>

class snack
{
 private :
 int x,y,life;

 public :
 snack();
 void paint();
 int hmm(int sx,int sy);
};

snack::snack()
{
  x = random(30)*20+25;
  y = random(22)*20+25;
  life = 1;
  paint();
}

void snack::paint()
{
```

```
setfillstyle(1,7);
bar(x-5,y-5,x+5,y+5);
}

int snack::hmm(int sx,int sy)
{
if (life) {
   if (x==sx && y==sy) {
      life = 0;
      sound(1000);
      delay(100);
      nosound();
      return 1;
      }
   else
      return 0;
   }
else
   return 0;
}

class snake
{
 private :
 int x[255],y[255],len;
 int dx,dy;
 void paint(int x,int y,int cl);
 void checklife();

 public :
 int life;
 snake();
 void move();
 void eat(snack *target);
 void up()    { if (dy==0) {dy=-10; dx=0;}}
 void down()  { if (dy==0) {dy= 10; dx=0;}}
 void left()  { if (dx==0) {dx=-10; dy=0;}}
 void right() { if (dx==0) {dx= 10; dy=0;}}
};

snake::snake()
{
  len = 1;
  life = 1;
  x[0] = 325;
  y[0] = 245;
  for(int i=1;i<255;i++) {
     x[i]=0;y[i]=0;
     }
  dx = 10;
  dy = 0;
  paint(x[0],y[0],2);
}

void snake::paint(int x,int y,int cl)
{
setfillstyle(1,cl);
```

```
bar(x-5,y-5,x+5,y+5);
}

void snake::move()
{
if (!x[len-1]==0)
   paint(x[len-1],y[len-1],0);
for (int i=len-1;i>0;i--) {
    x[i]=x[i-1];
    y[i]=y[i-1];
    }
x[0]+=dx;
y[0]+=dy;
checklife();
paint(x[0],y[0],2);
}

void snake::checklife()
{
life = life && (x[0]>10) && (x[0]<630) && (y[0]>10) && (y[0]<470);

for (int i=1;i<len;i++)
    life = life && !(x[0]==x[i] && y[0]==y[i]);
}

void snake::eat(snack *target)
{
len = len+target->hmm(x[0],y[0]);
}


void initcanvas()
{
setfillstyle(1,7);
bar(0,0,getmaxx(),getmaxy());
setfillstyle(1,0);
bar(10,10,getmaxx()-9,getmaxy()-9);
}

int main()
{
int gd,gm;
detectgraph(&gd,&gm);
initgraph(&gd,&gm,"egavga.bgi");
initcanvas();
snack *osnack[50];
snake *osnake = new snake;
for (int i=0;i<50;i++)
    osnack[i] = new snack;
do {
   if (!kbhit()) {
      osnake->move();
      for (i=0;i<50;i++)
        osnake->eat(osnack[i]);
      delay(100);
      }
   else {
```

```
      if (!getch())
         switch (getch()) {
              case 72: osnake->up(); break;
              case 75: osnake->left(); break;
              case 77: osnake->right(); break;
              case 80: osnake->down(); break;
              }
   }
} while(osnake->life);

for (i=0;i<50;i++)
    delete osnack[i];

closegraph();
return 0;
}
```

Pertanyaan

1. Apa fungsi dari operator titik (dot)?
2. Apa perbedaan antara public dan private data member?
3. Dapatkah suatu member function (method) dibuat menjadi private?
4. Dapatkah data member (property) dibuat menjadi public?
5. Jika anda mendeklarasikan dua object `Cat`, dapatkah mereka memiliki nilai yang berbeda pada property `itsAge`?
6. Apa nama function yang digunakan untuk menginitialisasi suatu class?

Latihan

1. Develop and use the following class

```
//Program object random file
//Oleh : Hendra Soewarno
//Hanya berjalan di Turbo C++ 3.0
//Dengan pendekatan OOP dan memanfaatkan fasilitas Graphics.

#include "iostream.h"
#include "fstream.h"

class rs {
public :
  rs(); // constructor
  rs(char *fname, char *structure, int rsize); //constructor
overloading
  ~rs(); //desctructor
  void open(char *fname, char *structure, int rsize);
  int endoffile();
  long reccount();
  long recno();
  void addnew();
  void update();
```

```
protected :
  fstream fsource;
  char *fields;
  int  recsize;
  long absoluteposition;
  long filesize();
  void refresh();
};

//constructor
rs::rs()
{
}

rs::rs(char *fname, char *structure, int rsize)
{
fsource.open(fname,ios::in | ios::out | ios::binary);
fields = structure;
recsize = rsize;
absoluteposition = 0;
refresh();
}

//destructor
rs::~rs()
{
fsource.close();
}

void rs::open(char *fname, char *structure, int rsize)
{
rs(fname,structure,rsize);
}

int rs::endoffile()
{
return fsource.eof();
}

long rs::filesize()
{
long nfilesize;
absoluteposition = fsource.tellg();
fsource.seekg(0,ios::end);
nfilesize = fsource.tellg();
fsource.seekg(absoluteposition,ios::beg);
return nfilesize;
}

long rs::recno()
{
return fsource.tellg()/recsize;
}

long rs::reccount()
{
```

```
return filesize()/recsize;
}

void rs::refresh()
{
if (!endoffile())
   {
   absoluteposition = fsource.tellg();
   fsource.read(fields,recsize);
   fsource.seekg(absoluteposition,ios::beg);
   }
}

void rs::addnew()
{
absoluteposition = filesize();
}

void rs::update()
{
fsource.seekg(absoluteposition,ios::beg);
fsource.write(fields,recsize);
fsource.seekg(absoluteposition,ios::beg);
}
```

Sample program using clssiswa :

```
//Program Entry data siswa
//Oleh : Hendra Soewarno
//Hanya berjalan di Turbo C++ 3.0
//Dengan pendekatan OOP dan memanfaatkan fasilitas Graphics.

#include "objrs.cpp"
#include "iostream.h"
#include "string.h"

struct siswa {
   char nim[10];
   char nama[30];
   float nilai;
} rsiswa;

int main()              // program utama
{
char nim[10];
rs *ors = new rs("c:\csiswa.rec", (char *) &rsiswa, sizeof(rsiswa));
do {
   cout << "Data ke-" << ors->reccount() + 1<< endl;
   cout << "Nim  :";
   cin.getline(nim,10,'\n');
   if (nim[0]) {
      ors->addnew();
      strcpy(rsiswa.nim,nim);
      cout << "Nama  :";
      cin.ignore();
      cin.getline(rsiswa.nama,50,'\n');
```

```
      cout << "Nilai :";
      cin >> rsiswa.nilai;
      cin.ignore();
      ors->update();
      }
} while (nim[0]);

delete ors;
return 0;
}
```

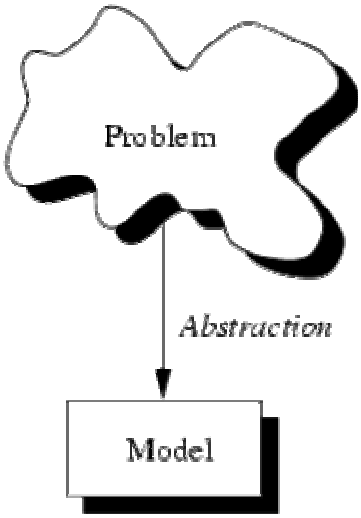
Section 16

C++ and Object-Oriented Programming (OOP)

C++ secara penuh mendukung konsep OOP, termasuk empat pilar dari pengembangan OOP yaitu : encapsulation, data hiding, inheritance, and polymorphism.

OPP berkaitan erat dengan Abstract Data Type (ADT).

Pada konsep pengembangan OOP, awal dari pengembangan program adalah mencoba memodelkan permasalahan yang dikenal sebagai *abstraction* yang digambarkan sebagai berikut :

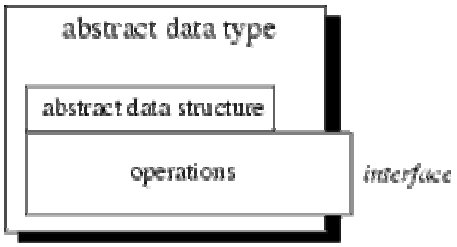


Pada model merupakan suatu pandangan abstrak terhadap permasalahan. Dalam hal ini model hanya terfokus pada permasalahan dimana :

- Data apa saja yang terpengaruh
- Operasi apa yang perlu dilakukan

Misalnya ketika seorang dosen ingin membuat laporan hasil ujian, maka data yang perlu adalah Nim, Nama, Nilai, dan operasi yang dilakukan adalah mentotalkan, dan menghitung nilai rata-rata.

Suatu ADT terdiri dari suatu struktur data abstrak dan operasi-operasi, hanya operasi-operasi yang dapat dilihat dari luar sebagai interface (antar muka), sedangkan detail proses disembunyikan dari pemakai.



Encapsulation dan Data Hiding

Prinsip menyembunyikan dengan mengunakan struktur data dan hanya menyediakan operasi-operasi untuk interface dengan luar dikenal sebagai *encapsulation* dan *data hiding*.

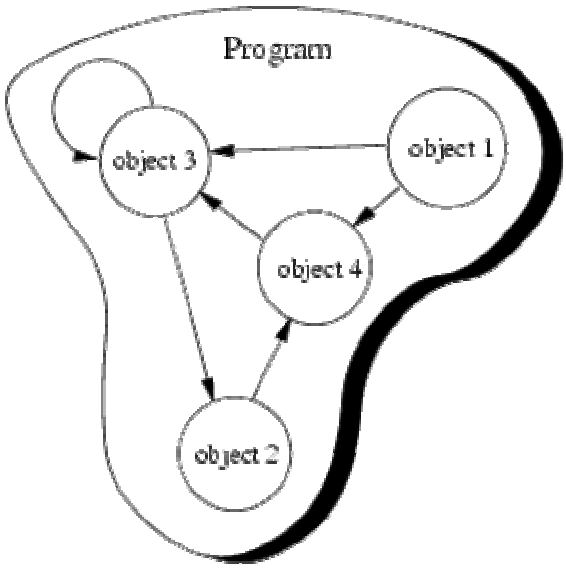
Data hiding merupakan karakteristik yang bernilai tinggi dari yang dapat digunakan oleh suatu objek tanpa diketahui oleh user bagaimana kerja internal dari objek tersebut. Anda dapat membayangkan objek itu sebagai suatu mobil, dimana untuk bisa mengendari mobil, anda tidak perlu mengetahui bagaimana kerja internal dari mobil tersebut, yang penting anda dapat menyetir, masuk gigi, menambah kecepatan, dan menghentikan.

Class pada C++ mengambarkan suatu ADT, dimana terdiri dari properti dan metoda sebagai interface.

Hubungan antara Object dan Class

Object merupakan instance dari Class, beberapa object dapat berasal dari Class yang sama, tetapi memiliki properti yang berbeda-beda.

Pada pembuatan program OOP, dimulai dengan pembuatan Class, dan pada saat runtime akan dibuat instance dari Class sebagai Objek, jadi program tersebut berjalan sebagai objek-objek dimemori komputer.



Objek-objek dalam program akan berkomunikasi antara satu dengan yang lain menggunakan pengiriman message.

Inheritance

Salah satu keunggulan dari OOP adalah isu pemanfaatan kembali. Dengan inheritance tugas pemrograman dapat dipemudah. Suatu class yang telah dikembangkan dapat disempurnakan dengan mengembangkan child class (sub class) yang merupakan inheritance dari parent class (super class).

Pada inheritance semua properti dan metode dari super class akan tersedia untuk diakses oleh sub class.

Inheritance menyerupai pengembangan produk baru pada suatu perusahaan, dimana engineer tidak perlu membuat produk baru dari awal, tetapi dengan mengacu pada produk yang telah ada, dan dilakukan penambahan dan perbaikan.

Overriding

Dalam melakukan inheritance terhadap suatu parent class, kadang-kadang pada sub class kita perlu mendefinisi ulang metoda. Dari hal inilah istilah overriding muncul, dimana overriding adalah mengubah implementasi suatu metoda pada super class di sub class.

Polymorphyms

Pada polymorphyms, suatu variabel dengan type super class dapat digunakan untuk mengacu terhadap sub class yang merupakan inheritance dari super class tersebut.

Contoh :

```
//Program inheritance class rs
//Oleh : Hendra Soewarno
//Hanya berjalan di Turbo C++ 3.0
//Dengan pendekatan OOP dan memanfaatkan fasilitas Graphics.

#include "objrs.cpp"

class rse : public rs {
public :
  rse();
  rse(char *fname, char *structure, int rsize);
  void open(char *fname, char *structure, int rsize); // overiding
  void movefirst();
  void movelast();
  void moveprevious();
  void movenext();
};

rse::rse(char *fname, char *structure, int rsize)
{
```

```
open(fname,structure,rsize);
}

void rse::open(char *fname, char *structure, int rsize)
{
rs::open(fname,structure,rsize);
}


void rse::movefirst()
{
fsource.seekg(0,ios::beg);
refresh();
}

void rse::movelast()
{
fsource.seekg(-recsize,ios::end);
refresh();
}

void rse::moveprevious()
{
if (fsource.tellg()-recsize > 0)
   fsource.seekg(fsource.tellg()-recsize,ios::beg);
else
   movefirst();
refresh();
}

void rse::movenext()
{
if (fsource.tellg()+recsize < filesize())
   fsource.seekg(fsource.tellg()+recsize,ios::beg);
else
   fsource.seekg(0,ios::end);
refresh();
}
```

Contoh pemanfaatan rse :

```
//Program print data siswa
//Oleh : Hendra Soewarno
//Hanya berjalan di Turbo C++ 3.0
//Dengan pendekatan OOP dan memanfaatkan fasilitas Graphics.

#include "objrse.cpp"
#include "iostream.h"
#include "string.h"

struct siswa {
   char nim[10];
   char nama[30];
   float nilai;
} rsiswa;
```

```
int main()
{
char nim[10];
rse *ors = new rse("c:\csiswa.rec", (char *) &rsiswa, sizeof(rsiswa));

while (!ors->endoffile()) {
   cout << "Data ke-" << ors->recno() + 1<< endl;
   cout << "Nim  :" << rsiswa.nim;
   cout << "Nama :" << rsiswa.nama;
   cout << "Nilai:" << rsiswa.nilai;
   ors->movenext();
   }

delete ors;
return 0;
}
```

Pembuatan program permainan Tetris dengan konsep OOP.

```
//file header Shape.hpp
//mendefinisikan bentuk-bentuk pada game tetris
//pada bagian ini kita akan membuat suatu superclass Shape
//dan mengembangkannya menjadi Shape1,Shape2,Shape3,Shape4
//dengan inheritance
//pada shape2 dan shape4
//kita akan melakukan overriding terhadap metoda rotate
const block1[5][6] = {{0,0,0,0,0,0},
                      {0,0,1,0,0,0},
                      {0,0,1,0,0,0},
                      {0,0,1,0,0,0},
                      {0,0,1,0,0,0}};

const block2[5][6] = {{0,0,0,0,0,0},
                      {0,0,1,1,0,0},
                      {0,0,1,1,0,0},
                      {0,0,0,0,0,0},
                      {0,0,0,0,0,0}};

const block3[5][6] = {{0,0,0,0,0,0},
                      {0,1,0,0,0,0},
                      {0,1,1,0,0,0},
                      {0,0,1,0,0,0},
                      {0,0,0,0,0,0}};

const block4[5][6] = {{0,0,0,0,0,0},
                      {0,0,1,0,0,0},
                      {0,0,1,0,0,0},
                      {0,0,1,1,0,0},
                      {0,0,0,0,0,0}};

const rblock4[5][6]= {{0,0,0,0,0,0},
                      {0,0,1,1,0,0},
                      {0,0,0,1,0,0},
                      {0,0,0,1,0,0},
                      {0,0,0,0,0,0}};

class shape
```

```
{
protected:
  int nrotate;
  void savematrix();

public:
  int  matrix[5][6],
      lmatrix[5][6],
      umatrix[5][6],color;
  shape();
  virtual void rotate();  //polymorphys
  void unrotate();
};

shape::shape()            //shape constructor
{
nrotate=1;
color=random(6)+9;
}

void shape::savematrix()
{
memcpy(umatrix,matrix,sizeof(umatrix));
}

void shape::rotate()
{
int i,j;
savematrix();
if (nrotate == 1)
   memcpy(lmatrix,matrix,sizeof(lmatrix));
for (i=1;i<5;i++)
    for(j=1;j<5;j++)
       if (nrotate == 1)
        matrix[i][j] = lmatrix[j][5-i];
       else
        matrix[i][j] = lmatrix[i][j];
nrotate = -nrotate;
}

void shape::unrotate()
{
memcpy(matrix,umatrix,sizeof(matrix));
}

//buat shape1 dimana merupakan inherit dari shape
class shape1: public shape
{
public :
shape1() {memcpy(matrix,block1,sizeof(matrix));}
};

//buat shape2 dimana merupakan inherit dari shape
class shape2: public shape
{
public :
shape2() {memcpy(matrix,block2,sizeof(matrix));}
```

```
void rotate() {savematrix();}  //overiding
};

//buat shape3 dimana merupakan inherit dari shape
class shape3: public shape
{
public :
shape3() {memcpy(matrix,block3,sizeof(matrix));}
};

//buat shape2 dimana merupakan inherit dari shape
class shape4: public shape
{
public :
shape4() {memcpy(matrix,block4,sizeof(matrix));}
void rotate();  // overiding
};

void shape4::rotate()
{
int i,j;
savematrix();
switch (nrotate) {
   case 1:
      memcpy(lmatrix,matrix,sizeof(lmatrix));
      for (i=1;i<5;i++)
          for(j=1;j<5;j++)
            matrix[i][j] = lmatrix[j][5-i];
      break;
   case 2:
      memcpy(matrix,rblock4,sizeof(matrix));
      break;
   case 3:
      memcpy(lmatrix,rblock4,sizeof(lmatrix));
      for (i=1;i<5;i++)
          for(j=1;j<5;j++)
            matrix[i][j] = lmatrix[j][5-i];
      break;
   default:
      memcpy(matrix,block4,sizeof(matrix));
      nrotate=0;
  }
nrotate++;
}

//header file score.hpp
class scoreboard
{
private:
   int score;
   void paint();

public:
   scoreboard();
   void addscore(int value) { score +=value; paint();}
};
```

```
scoreboard::scoreboard()  //constructor
{
score = 0; paint();
}

void scoreboard::paint()
{
char temp1[20];
char temp2[10];
strcpy(temp1,"score :");
itoa(score,temp2,10);
strcat(temp1,temp2);
setfillstyle(1,0); bar(1,1,100,50);   // clear scoreboard
setcolor(15); outtextxy(10,15,temp1); // print new score
}
//program tetris.cpp
//pada bagian ini akan memanfaatkan virtual dan polymorphyms

#include <graphics.h>
#include <string.h>
#include <stdlib.h>
#include <conio.h>
#include <dos.h>
#include <shape.hpp>
#include <score.hpp>

// mengkalkulasi titik koordinat game ke layar
void maptoscreen(int x, int y, int &x1, int &y1, int &x2, int &y2)
{
x1 = x*20;
y1 = y*20;
x2 = (x+1)*20-1;
y2 = (y+1)*20-1;
}

// mengambar 1 sel
void drawcell(int x, int y, int c)
{
int x1,y1,x2,y2;
maptoscreen(x,y,x1,y1,x2,y2);
setfillstyle(1,c);
bar(x1,y1,x2,y2);
// membuat efek 3 D
setcolor(15);
line(x1,y1,x1,y2);line(x1,y1,x2,y1);  // bingkai kiri atas
setcolor(8);
line(x2,y1,x2,y2);line(x1,y2,x2,y2);  // bingkai kanan bawah
}

// menghapus 1 sel
void clearcell(int x, int y)
{
int x1,y1,x2,y2;
maptoscreen(x,y,x1,y1,x2,y2);
setfillstyle(1,0);
bar(x1,y1,x2,y2);
```

```
}

// class untuk menampilkan bentuk berikutnya
class shapeboard
{
private :
shape *cshape;
void paint();

public :
void setshape(shape *nextshape) {cshape = nextshape; paint();}
void getshape(shape *currentshape) {currentshape = cshape;}
};

void shapeboard::paint()
{
int i,j;
for(i=0;i<5;i++)
   for(j=0;j<6;j++)
      if (cshape->matrix[i][j])
       drawcell(j+23,i+1,cshape->color);
      else
       clearcell(j+23,i+1);
}

// class cell, visibel dan warna
class cell
{
public:
int visible,color;
};

// tempat permainan
class gameboard
{

private:
   cell matrix[24][16];
   void paint();

public:
   gameboard();
   int  testshape(int row,int col, shape *shapex);
   void pasteshape(int row,int col, shape *shapex);
   void clearshape(int row,int col, shape *shapex);
   void putshape(int row,int col, shape *shapex);
   int  checkcomplete();
}

gameboard::gameboard()
{
int i,j;
for (i=0; i<24; i++)
    for (j=0; j < 15; j++)
      matrix[i][j].visible = 1;
for (i=0; i<21; i++)
    for (j=2; j < 12; j++)
```

```
      matrix[i][j].visible = 0;
paint();
}

void gameboard::paint()
{
int i,j;
setcolor(15);
line(10*20-1,1*20,10*20-1,(1+20)*20);
line((10+10)*20+1,1*20,(10+10)*20+1,(1+20)*20);
line(10*20-1,(1+20)*20+1,(10+10)*20+1,(1+20)*20+1);
for (i=0; i<21; i++)
    for (j=2; j < 12; j++)
      if (matrix[i][j].visible==1)
         drawcell(10+j-2,i,matrix[i][j].color);
      else
         clearcell(10+j-2,i);
}

int gameboard::testshape(int row,int col, shape *shapex)
{
int i,j;
for (i=0;i<5;i++)
    for (j=0;j<6;j++)
      if(matrix[i+row][j+col].visible + shapex->matrix[i][j] > 1)
         return 0;
return 1;
}

void gameboard::pasteshape(int row,int col, shape *shapex)
{
int i,j;
for (i=0;i<5;i++)
    for (j=0;j<6;j++) {
      if (shapex->matrix[i][j] !=0)
         drawcell(10+j+col-2,i+row,shapex->color);
      }
}

void gameboard::clearshape(int row,int col, shape *shapex)
{
int i,j;
for (i=0;i<5;i++)
    for (j=0;j<6;j++) {
      if (shapex->matrix[i][j] !=0)
         clearcell(10+j+col-2,i+row);
      }
}

void gameboard::putshape(int row,int col, shape *shapex)
{
int i,j;
for (i=0;i<5;i++)
    for (j=0;j<6;j++)
      if (shapex->matrix[i][j]==1)
         {
         matrix[i+row][j+col].visible = 1;
```

```
          matrix[i+row][j+col].color = shapex->color;
          }
}

int gameboard::checkcomplete()
{
int lineok,score;
score = 0;
for (int i=20; i>0; i--)
    {
    lineok = 1;
    for (int j=2; j < 12 && lineok; j++)
      lineok = matrix[i][j].visible;

    if (!lineok==0)
       {
       sound(500);
       delay(100);
       nosound();
       for (int k=i; k>0; k--)
         for (int l=2;l<12;l++)
            matrix[k][l] = matrix[k-1][l];
       for (int m=1;m<11; m++)
         matrix[0][m].visible = 0;
       score+=10;
       i++;
       }
    }
paint();
return score;
}

class gametetris
{

private:
  int row,col,finish,next,second;
  char key;
  gameboard *gb;
  scoreboard *sb;
  shapeboard *sh;
  shape *s0,*s1;

public:
  gametetris();
  ~gametetris();
  void play();
}

gametetris::gametetris()
{
gb = new gameboard();
sb = new scoreboard();
sh = new shapeboard();
finish = 0;
next = 1;
}
```

```
gametetris::~gametetris()
{
delete(gb);
delete(sb);
delete(sh);
delete(s1);
}

void gametetris::play()
{
s0 = new shape1;
do {
   if (next==1) {
      switch (random(4)+1) {
           case 1:  s1 = new shape1; break;
           case 2:  s1 = new shape2; break;
           case 3:  s1 = new shape3; break;
           default: s1 = new shape4; break;
           }
      sh->setshape(s1);
      row = 0;
      col = 4;
      second=250;
      next = 0;
      }
   if (kbhit()) {
      key = getch();
      if (key==0) key = getch();
      switch (key) {
      case 27:finish=1;break;                    // esc
      case 76:s0->rotate();                      // 5
            if (!gb->testshape(row,col,s0))
               s0->unrotate();
            break;
      case 75:if (gb->testshape(row,col-1,s0)) // left
               col--;
            break;
      case 77:if (gb->testshape(row,col+1,s0)) // right
               col++;
            break;
      case 80:second=0; break;                   // down
      }
      }
   else
      if (gb->testshape(row,col,s0)) {
        gb->pasteshape(row,col,s0);
        delay(second);
        if (gb->testshape(row+1,col,s0)) {
            gb->clearshape(row,col,s0);
            row++;
            }
        else {
            gb->putshape(row,col,s0);
            sb->addscore(gb->checkcomplete());
            delete s0;
            s0 = s1;
```

```
            next = 1;
            if (row < 2) finish = 1;
            }
         }
       else
         finish = 1;
} while (!finish==1);

}
//program utama
int main()
{
int gdriver,gmode;
randomize();
detectgraph(&gdriver,&gmode);
initgraph(&gdriver,&gmode,"");
gametetris *gt = new gametetris;  // create game
gt->play();                       // play
delete(gt);                       // end game
closegraph();
return 0;
}
```

Section 17
Exception handling

Exception handling merupakan fasilitas baru yang diperkenalkan pada ANSI-C++ standard. Jika anda menggunakan compiler C++ yang tidak mendukung fasilitas ini, maka anda tidak dapat memanfaatkan fasilitas ini.

Pengembangan program tidak terlepas dari kesalahan dan kesilapan, untuk menangani masalah tersebut diperlukan suatu Exception handling.

Untuk keperluan tersebut C++ menyediakan operator baru untuk membantu kita yaitu: **try**, **throw** and **catch**.

Adapun bentuk strukturnya adalah sebagai berikut:

```
try {
  // code to be tried
  throw exception;
}
catch (type  exception)
{
  // code to be executed in case of exception
}
```

dan secara operational:
- Kode didalam blok **`try`** akan dijalankan secara normal. Dalam hal ini kalau suatu exception (pengecualian) terjadi, kode ini harus menggunakan keyword **`throw`** (lempar) suatu parameter ke suatu exception.
- Jika suatu exception terjadi, maka instruksi **`throw`** didalam blok **`try`** akan dijalankan, blok **`catch`** dijalankan dengan menerima parameter yang dilewatkan oleh **`throw`**.

Sebagai contoh:

```
// exceptions
#include <iostream.h>

int main () {
  char myarray[10];
  try
  {
    for (int n=0; n<=10; n++)
    {
      if (n>9) throw "Out of range";
      myarray[n]='z';
    }
  }
  catch (char * str)
  {
```

```
Exception: Out of range
```

```
  cout << "Exception: " << str << endl;
 }
 return 0;
}
```

Pada contoh diatas, didalam **n** loop, saat nilai **n** lebih besar dari pada 9 maka suatu exception akan dilempar. Ketika **throw** dijalankan, blok **try** akan berakhir dan setiap objek yang dibuat didalam blok **try** akan dimusnahkan. Setelah itu, kendali di lewatkan ke blok **catch** yang bersesuian. Akhirnya program akan dilanjutkan ke program setelah blok **catch**.

Blok **catch** dapat dioverloaded menurut type dari parameternya, contoh :

```
// exceptions: multiple catch blocks
#include <iostream.h>

int main () {
 try
 {
  char * mystring;
  mystring = new char [10];
  if (mystring == NULL) throw "Allocation failure";
  for (int n=0; n<=100; n++)
  {
   if (n>9) throw n;
   mystring[n]='z';
  }
 }
 catch (int i)
 {
  cout << "Exception: ";
  cout << "index " << i << " is out of range" << endl;
 }
 catch (char * str)
 {
  cout << "Exception: " << str << endl;
 }
 return 0;
}
```

```
Exception: index 10 is
out of range
```

Kita dapat juga dapat mendefinisikan suatu blok **catch** yang akan menangkap semua exception tanpa tergantung pada type yang digunakan untuk memperhatikan type yang digunakan pada saat memanggil **throw**. Dalam hal ini kita cukup menulis tiga buah titik (...) sebagai penganti dari nama type, contoh :

```
try {
  // code here
}
catch (...) {
  cout << "Exception occurred";
}
```

Kita dapat juga membuat nested **try-catch** seperti blok berikut :

```
try {
  try {
      // code here
  }
  catch (int n) {
      throw;
  }
}
catch (...) {
  cout << "Exception occurred";
}
```

Exceptions not caught

Jika suatu execption tidak tertangkap oleh semua **catch** statement karena tidak ada type yang sesuai, maka suatu function special **terminate** akan di panggil.

Function ini umumnya telah terdefinisi sehingga akan menghentikan proses saat ini seketika dan menampilkan "Abnormal termination" sebagai pesan kesalahan. Formatnya adalah sebagai berikut:

```
void terminate();
```

Standard exceptions

Beberapa function dalam bahasa C++ standard library mengirim exception-exception yang dapat ditangkap kalau kita mengikutkannya dalam blok **try**. Exception ini akan dikirim dengan suatu class yang diturunkan dari **std::exception** sebagai type. Class ini (`std::exception`) didefinisikan dalam C++ standard header file **<exception>** dan melayani sebagai suatu pola untuk hirarki standard dari exception-exception berikut:

```
exception
├─bad_alloc      (thrown by new)
├─bad_cast       (thrown by dynamic_cast when fails with a
                 referenced type)
├─bad_exception  (thrown when an exception doesn't match any
                 catch)
├─bad_typeid     (thrown by typeid)
├─logic_error
│ ├ domain_error
│ ├ invalid_argum
  ent
│ ├ length_error
│ └ out_of_range
├─runtime_error
│ ├ overflow_erro
  r
│ ├ range_error
```

```
    underflow_err
    or
ios_base::failu
re                (thrown by ios::clear)
```

Karena class ini adalah hirarki, jika anda mengikutkan suatu blok **`catch`** untuk menangkap salah satu dari exception tersebut gunakan argumen dengan reference (tambahkan suatu ampersand & setelah type) anda juga dapat menangkap semua turunannya (aturan dari inheritanceance di C++).

Contoh berikut akan menangkap (catch) suatu **`bad_typeid`** exception (diturunkan dari **`exception`**):

```
// standard exceptions

#include <iostream.h>
#include <exception>
#include <typeinfo>

class A {virtual f() {}; };

int main () {
  try {
    A * a = NULL;
    typeid (*a);
  }
  catch (std::exception& e)
  {
    cout << "Exception: " << e.what();
  }
  return 0;
}
```

```
Exception: Attempted
typeid of NULL pointer
```

Section 18

User defined data types

Kita telah melihat suatu data type yang didefinisikan oleh pemakai (programmer): yaitu type data structures. Pada dasarnya masih ada beberapa jenis user defined data types:

Definition of own types (`typedef`).

C++ memperbolehkan kita untuk mendefinisikan kita sendiri berdasarkan type data yang telah ada. Untuk melakukan hal ini kita akan menggunakan keyword **`typedef`**, yang mana memiliki bentuk:

```
typedef   existing_type   new_type_name ;
```

Contoh pemakaian:

```
typedef char C;
typedef unsigned int WORD;
typedef char * string_t;
typedef char field [50];
```

Sekarang kita telah mendefinisikan empat type data baru yaitu: **`C`**, **`WORD`**, **`string_t`** sehingga dapat digunakan untuk deklarasi variable:

```
C achar, anotherchar, *ptchar1;
WORD myword;
string_t ptchar2;
field name;
```

`typedef` dapat berguna untuk type-type yang sering digunakan, khususnya yang memiliki nama yang panjang.

Unions

Union memperbolehkan suatu bagian dari memory diakses sebagai type data yang berbeda, dimana masing-masing dari mereka memiliki lokasi yang sama di memori. Deklarasi dan pemakaiannya sama dengan suatu structure, tetapi fungsinya berbeda sama sekali :

```
union model_name {
  type1 element1;
  type2 element2;
  type3 element3;
  .
  .
} object_name;
```

Semua elemen dari deklarasi *union* menggunakan tempat memory yang sama. Ukurannya sama dengan elemen yang terbesar. Contoh :

```
union mytypes_t {
  char c;
  int i;
  float f;
  } mytypes;
```

mendefinisikan tiga elemen:

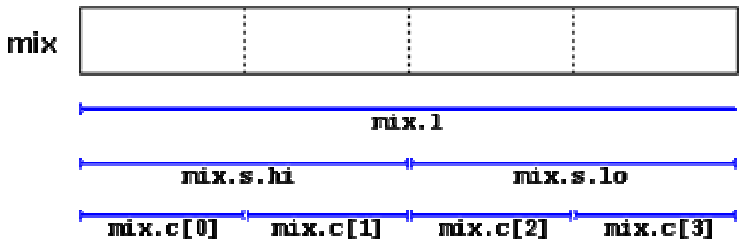
```
mytypes.c
mytypes.i
mytypes.f
```

masing-masing memiliki type data yang berbeda, tetapi sama-sama menggunakan lokasi memori yang sama, modifikasi terhadap suatu elemen akan mempengaruhi nilai dari elemen lain.

Salah satu kegunaan *union* adalah untuk menyatukan suatu type yang lebih panjang dari elemen-elemen type lainnya. Contoh :

```
union mix_t{
  long l;
  struct {
    short hi;
    short lo;
    } s;
  char c[4];
} mix;
```

mendefinisikan tiga nama yang memungkinkan kita mengakses group yang sama dari 4 byte: **`mix.l`**, **`mix.s`** dan **`mix.c`**, dimana gambaran masing-masing field dimemori adalah sebagai berikut



Anonymous unions

Pada C++ kita memiliki option bahwa union dapat berupa anonymous. Jika kita mengikutkan suatu union dalam structure tanpa nama object (biasaya ditempatkan setelah tanda curly brackets `{ }`) union akan menjadi anonymous dan kita akan dapat mengakses

elemen secara langsung dengan berdasarkan namanya. Sebagai contoh, perhatikan perbedaan antara dua deklarasi berikut:

union

```
struct {
  char title[50];
  char author[50];
  union {
    float dollars;
    int yens;
  } price;
} book;
```

anonymous union

```
struct {
  char title[50];
  char author[50];
  union {
    float dollars;
    int yens;
  };
} book;
```

Perbedaan satu-satunya antara kedua potongan kode diatas adalah yang pertama kita memberi satu nama pada union (**`price`**) dan pada yang kedua kita tidak Perbedaannya adalah ketika kita mengakses anggota dari **`dollars`** dan **`yens`** dari objek book. Pada kasus pertama adalah:

```
book.price.dollars
book.price.yens
```

dan pada kasus kedua adalah:

```
book.dollars
book.yens
```

Satu hal lagi yang perlu anda ingat adalah, field **`dollars`** dan **`yens`** menggunakan tempat yang sama dimemory sehingga tidak dapat digunakan untuk menyimpan dua nilai yang berbeda.

Enumerations (`enum`)

Enumerations digunakan untuk membuat data types untuk mengandung sesuatu yang berbeda baik numeric, character ataupun konstanta **`true`** dan **`false`**. Memiliki bentuk sebagai berikut:

```
enum model_name {
  value1,
  value2,
  value3,
  .
  .
} object_name;
```

Sebagai contoh, kita dapat membuat suatu type baru yang bernama **`color`** untuk menyimpan warna-warna dengan deklarasi berikut:

```
enum colors_t {black, blue, green, cyan, red, purple,
yellow, white};
```

Selanjutnya kita dapat menggunakannya untuk deklarasi variable seperti biasanya:

```
colors_t mycolor;

mycolor = blue;
if (mycolor == green) mycolor = red;
```

Pada kenyataannya enumerated data type dikompilasi sebagai suatu integer dan nilai yang mungkin. Jika tidak ditentukan, nilai integer yang ekuivalen pertama yang mungkin adalah `0` and berikutnya adalah a +1.

Kita dapat juga melengkapi suatu nilai integer untuk enumerated type (misalnya untuk elemen yang pertama), contoh ::

```
enum months_t { january=1, february, march, april,
                may, june, july, august,
```

september, october, november, december} y2k;

Appendix A
Preprocessor directives

Preprocessor directives are orders that we include within the code of our programs that are not instructions for the program itself but for the preprocessor. The preprocessor is executed automatically by the compiler when we compile a program in C++ and is in charge of making the first verifications and digestions of the program's code.

All these directives must be specified in a single line of code and they do not have to include an ending semicolon `;`.

#define

At the beginning of this tutorial we have already spoken about a preprocessor directive: **`#define`**, that serves to generate what we called *defined constantants* or *macros* and whose form is the following:

```
#define name value
```

Its function is to define a macro called *name* that whenever it is found in some point of the code is replaced by *value*. For example:

```
#define MAX_WIDTH 100
char str1[MAX_WIDTH];
char str2[MAX_WIDTH];
```

It defines two strings to store up to 100 characters.

`#define` can also be used to generate macro functions:

```
#define getmax(a,b) a>b?a:b
int x=5, y;
y = getmax(x,2);
```

after the execution of this code **`y`** would contain `5`.

#undef

`#undef` fulfills the inverse functionality of **`#define`**. It eliminates from the list of defined constants the one that has the name passed as a parameter to **`#undef`**:

```
#define MAX_WIDTH 100
char str1[MAX_WIDTH];
#undef MAX_WIDTH
#define MAX_WIDTH 200
char str2[MAX_WIDTH];
```

#ifdef, #ifndef, #if, #endif, #else and #elif

These directives allow to discard part of the code of a program if a certain condition is not fulfilled.

#ifdef allows that a section of a program is compiled only if the *defined constant* that is specified as the parameter has been defined, independently of its value. Its operation is:

```
#ifdef name
// code here
#endif
```

For example:

```
#ifdef MAX_WIDTH
char str[MAX_WIDTH];
#endif
```

In this case, the line **`char str[MAX_WIDTH];`** is only considered by the compiler if the *defined constant* **MAX_WIDTH** has been previously defined, independently of its value. If it has not been defined, that line will not be included in the program.

#ifndef serves for the opposite: the code between the **#ifndef** directive and the **#endif** directive is only compiled if the constant name that is specified has not been defined previously. For example:

```
#ifndef MAX_WIDTH
#define MAX_WIDTH 100
#endif
char str[MAX_WIDTH];
```

In this case, if when arriving at this piece of code the *defined constant* **MAX_WIDTH** has not yet been defined it would be defined with a value of `100`. If it already existed it would maintain the value that it had (because the `#define` statement won't be executed).

The **#if**, **#else** and **#elif** (*elif* = *else if*) directives serve so that the portion of code that follows is compiled only if the specified condition is met. The condition can only serve to evaluate constant expressions. For example:

```
#if MAX_WIDTH>200
#undef MAX_WIDTH
#define MAX_WIDTH 200

#elsif MAX_WIDTH<50
#undef MAX_WIDTH
#define MAX_WIDTH 50

#else
#undef MAX_WIDTH
```

```
#define MAX_WIDTH 100
#endif

char str[MAX_WIDTH];
```

Notice how the structure of chained directives **#if**, **#elsif** and **#else** finishes with **#endif**.

#line

When we compile a program and errors happen during the compiling process, the compiler shows the error that happened preceded by the name of the file and the line within the file where it has taken place.

The **#line** directive allows us to control both things, the line numbers within the code files as well as the file name that we want to appear when an error takes place. Its form is the following one:

```
#line number "filename"
```

Where *number* is the new line number that will be assigned to the next code line. The line number of successive lines will be increased one by one from this.

filename is an optional parameter that serves to replace the file name that will be shown in case of error from this directive until another one changes it again or the end of the file is reached. For example:

```
#line 1 "assigning variable"
int a?;
```

This code will generate an error that will be shown as error in file `"assigning variable"`, line `1`.

#error

This directive aborts the compilation process when it is found returning the error that is specified as the parameter:

```
#ifndef __cplusplus
#error A C++ compiler is required
#endif
```

This example aborts the compilation process if the *defined constant* **__cplusplus** is not defined.

#include

This directive has also been used assiduously in other sections of this tutorial. When the preprocessor finds an **#include** directive it replaces it by the whole content of the specified file. There are two ways to specify a file to be included:

```
#include "file"
#include <file>
```

The only difference between both expressions is the directories in which the compiler is going to look for the file. In the first case where the file is specified between quotes, the file is looked for in the same directory that includes the file containing the directive. In case that it is not there, the compiler looks for the file in the default directories where it is configured to look for the standard header files.

If the file name is enclosed between angle-brackets **<>** the file is looked for directly where the compiler is configured to look for the standard header files.

#pragma

This directive is used to specify diverse options to the compiler. These options are specific for the platform and the compiler you use. Consult the manual or the reference of your compiler for more information on the possible parameters that you can define with **#pragma**.

Appendix B

Programming Patterns

A lot of programming is based on using existing patterns. This section presents a number of simple, but commonly used, programming patterns. If you are working on a problem that fits one of these, pull it out and use it, adapting it as needed. This is usually much faster than trying to invent something new.

Skip Guard Plan

Often one needs to skip over a section of code when certain conditions arise. For example, you may need to skip over a computation when it would result in an attempt to divide by zero:

```
if (Count > 0)
   {
   Average = Total / Count;
   cout << "Average: " << Average << endl;
   }
else
   cout << "No data.  The average cannot be computed." << endl;
```

In general, the pattern is given by the following outline, which is written in a mixture of English and C++.

```
if (All is OK)
   Action to sometimes skip;
else
   Output a warning message;
```

Count-Controlled Loop Plan

This is used to read in and process a sequence of data items when you know in advance the number of items to be read. The outline looks like the following. Of course, in a real program one would add messages to the user, code for the `Process` function, etc.

```
int k;
Get the Count of how many data items should be processed;
for (k = 0; k < Count; k++)
   {
   Read one data item into Item;
   Process(Item);
   }
```

Input Checking Plan

This plan forces the user to enter an acceptable data item. For instance, the acceptable items might be Y, y, N, and n. The `Unacceptable` function shown below might, of course, be replaced by a simple condition. There is no need to write a function here unless the algorithm for determining unacceptable items is fairly complex.

```
cout << "Enter an item: ";
cin >> Item;
while (Unacceptable(Item))
   {
   cout << "Re-enter: ";
   cin >> Item;
   }
```

Sentinel-Controlled Loop Plan

Here the idea is to enter a special item (the sentinel) to indicate the end of the sequence of data items to be processed. The sentinel itself is not one of the data items. One might, for example, have data consisting of a string of positive integers and use 0 to signal the end of the data. Or, one might tell the user to press CTRL z to indicate the end of the data. This uses EOF (end of file) as a sentinel. The outline is as follows:

```
cin >> Item;
while (Item != Sentinel)
   {
   Process(Item);
   cin >> Item;
   }
```

As an example, let's write out more detail of how to use EOF as a sentinel. Suppose we want to enter and process a sequence of integers, with CTRL z used to indicate the end of data entry. Then we would use something like this:

```
int Item;
cin >> Item;
while (! cin.fail())
   {
   Process(Item);
   cin >> Item;
   }
```

The `fail` function returns true whenever the previous stream operation failed to input an integer. The usual reasons for such failure are that the user pressed CTRL z or that the user entered something (such as `abcd`) that could not be interpreted as an integer. Although the code shown above reads data from the keyboard, it could easily be adapted to read the data from a file.

The most general form of this pattern is as follows. Again, we use part English and part C++.

```
Get a data item.
while (we haven't reached the sentinel)
   {
   Process the data item;
   Get another data item;
   }
```

Do a Problem Until the User Wants to Quit

Often you want to allow the user to do some task repeatedly. A simple way to allow this is to ask the user whether or not to do another of whatever the task at hand might be.

```
char Reply;
do
   {
   OneProblem();
   cout << "Do another (y/n)? ";
   cin >> Reply;
   }
while (Reply == 'y');
```

Menu-Driven Plan

This is similar to the previous plan, but adds a menu that is printed to show the user possible commands, here assumed to be the letters a, b, c, and q. The letter q is used as the signal that the user wants to quit. Of course, you can use integers or whatever is convenient. After getting the user's choice the plan picks out the proper task to execute. Then, as long as the choice is not to quit, loop around and present the menu again, etc.

```
do
   {
   PrintMenu();
   cin >> Choice;
   if (Choice == 'a')
      DoA();
   else if (Choice == 'b')
      DoB();
   else if (Choice == 'c')
      DoC();
   else if (Choice != 'q')
      cout << "Invalid choice" << endl;
   }
while (Choice != 'q');
```

Adding a Counter to a Sentinel-Controlled While Loop

We have already seen how to create a sentinel-controlled while loop. This type of loop is used to read in data until the sentinel is reached. Often we want to count the number of data items read in (not including the sentinel itself). All you need is an integer `Count`

variable that is initialized to zero and incremented each time a new data item is read. Here is how this is done:

```
int Count;
Count = 0;
cin >> Item;
while (Item != Sentinel)
   {
   Count++;
   Process(Item);
   cin >> Item;
   }
```

Adding a Running Total to a Sentinel-Controlled While Loop

We can also add a running total to a sentinel-controlled while loop. The following example finds the total of all of the data items entered. Of course, the data items must be of a type that can be added, perhaps integers of floats, and the variable `Total` must have the same type. This example assumes that finding the total is the only processing that is needed on the data items.

```
Total = 0;
cin >> Item;
while (Item != Sentinel)
   {
   Total = Total + Item;
   cin >> Item;
   }
```

Finding an Average in a Sentinel-Controlled While Loop

Since we know how to total and count the data items read in by a sentinel-controlled while loop, we can easily find the average of these data items.

```
int Count;
Count = 0;
Total = 0;
cin >> Item;
while (Item != Sentinel)
   {
   Count++;
   Total = Total + Item;
   cin >> Item;
   }

if (Count > 0)
   {
   Average = Total / Count;
```

```
   cout << "Average: " << Average << endl;
   }
else
   cout << "No data.  The average cannot be computed." << endl;
```

Appendix C

C++ Formatting and Documentation

CIS Departmental Guidelines

Formatting

Although a program should first of all meet its specifications, there are important considerations other than correctness. Programs are written not just to be run, but also for others to read. Pity the poor maintenance program who is going to have to read and decipher your work several years from now! Make this person's job easier through good use of indentation, spacing, descriptive identifiers, and general neatness. It is typical, for example, to vertically align opening and closing braces, although there are a few authors who do not do this. One also should put blank lines between major sections of the program, such as between functions. It also helps a lot to put spaces after most types of punctutation, such as commas, and on both sides of binary operators, such as +. Indenting properly can do much to improve the ease of reading a program. It is typical to indent the "loop body" of a loop by about 3 spaces. Similarly, one should indent the body of a function about 3 spaces as well as the choices inside of an if/else. Look at the short example below as a guide.

Documentation

If you provide good documentation, that maintenance programmer is going to bless you, not curse you! There is both internal and external documentation. The internal documentation is contained within the program file(s) and includes a description at the top giving the inputs, overall processing, and outputs of the program. This section of documentation is essentially a brief user manual. A stranger who reads this section should know what your program does as well as how to set it up and run it.

The documentation at the top of a program file should list the date, the name of the author, the names of any other people who contributed to it, and any references used in any significant way (books, web sites, etc.). This allows both you and others to know at a glance what sources were used in producing this program.

Internal documentation also includes a comment section for each function, listing what is being passed into it via the parameters, what the function's main task is, and what values are being passed back out via the parameters or function name. (For object-oriented programming, the implicit object is also used to send values in and out.) The Given/Task/Return style of function comments is suggested. In this, the "Given" section lists each parameter that is used to send a value into the function. Beside the name of the parameter, a description of its meaning is listed. The "Task" section describes the overall

task of this function. It tells the reader what this function does if the function is called. Always describe the task in terms of the parameters. Try NOT to use items outside of the function in this description. The "Return" section lists each parameter (and its meaning) that is used to send a value back out of the function. It also lists any value returned in the function name and its meaning.

Internal documentation also includes a description of each user-defined class. External documentation is separate from the program and may consist of items like the specifications, various diagrams, a record of testing, etc.

Example

```
/* Filename:  area3.cpp

   Author:  Br. David Carlson

   Other Contributors:  None

   References:  C++ from the Ground Up, 2nd ed., by Herbert Schildt

   Date:  January 4, 2000

   Revised:  June 26, 2000

   This program asks the user to enter the length and width of a
   rectangle.  It then computes and prints the area of the rectangle.
   A warning is printed if the user enters a width larger than the
length,
   but the program still prints the area in such a case.
*/

#include <iostream>

using namespace std;


// Function prototypes:

void Explanation(void);

void GetValues(float & Length, float & Width);

float ComputeArea(float Length, float Width);

void PrintArea(float Area);


int main(void)
   {
   float Length, Width, Area;

   Explanation();
   GetValues(Length, Width);

   if (Width > Length)
```

```
      {
      cout << "Warning: the width you entered is larger than the
length.";
      cout << endl << "The area will still be found." << endl;
      }

   Area = ComputeArea(Length, Width);
   PrintArea(Area);

   return 0;
   }


/* Given:  Nothing.
   Task:   To ask the user for the length and width of a rectangle,
insisting
           on positive values for both, and to return these values via
the
           two parameters.
   Return: Length   The length entered by the user.
           Width    The width entered by the user.
*/
void GetValues(float & Length, float & Width)
   {
   cout << "Enter the rectangle's length: ";
   cin >> Length;

   while (Length <= 0)
      {
      cout << "Error: Length must be positive.  Re-enter: ";
      cin << Length;
      }

   cout << "Enter the rectangle's width: ";
   cin >> Width;

   while (Width <= 0)
      {
      cout << "Error: Width must be positive.  Re-enter: ";
      cin >> Width;
      }
   }


/* Given:  Length   The length of the rectangle.
           Width    The width of the rectangle.
   Task:   To compute the area of this rectangle.
   Return: The area in the function name.
*/
float ComputeArea(float Length, float Width)
   {
   return Length * Width;
   }


/* Given:  Nothing.
   Task:   To print an explanation of what the program does.
```

```
   Return: Nothing.
*/
void Explanation(void)
   {
   cout << "This program computes the area of a rectangle." << endl;
   cout << "You will be prompted to enter both the length and width.";
   cout << endl << "Enter a real number for each." << endl;
   cout << "The program will then compute and print the area.";
   cout << endl << endl;
   }


/* Given:  Area    The area of a rectangle.
   Task:   To print Area.
   Return: Nothing.
*/
void PrintArea(float Area)
   {
   cout << "The area is: " << Area << endl << endl;
   }
```

Other Concerns

Efficiency is also of some importance. Don't needlessly waste computer time or memory space. In particular, don't waste a lot of it. If there is a tradeoff between efficiency and a clear design, having a clear design is probably better (unless you would waste a lot of time or space to get it). Of course, it also does not make sense to waste a lot of your own time (or your employer's) to make minor speed improvements to a program that will be rarely used and runs fast enough as is.

Here are some "thou shalt not" rules of thumb: Don't use an obscure method when clear methods are available. Do not have a variable do double duty, as in using a variable called `Total` to hold both the total of some data items, and later the average of these data items. (The name `Total` indicates that it should hold a sum, not an average. Don't confuse the readers of your program!) Do not have a section of program do double duty in a non-obvious way. The intention of each section of code should be reasonably clear.

Appendix D

Tips & Tricks

Clearing the Screen

To clear the entire output window of a program, you must use a system function found in the stdlib.h include file. First, include stdlib.h at the top of your program. Then, just insert the following call whenever you want to clear the screen:

```
system("CLS");
```

The program below is an example of how to use this method of clearing the screen.

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

int main()

{
cout << "Text before the clear screen." << endl;
system("CLS");
cout << "Text following the clear screen." << endl;
return 0;
}
```

Sending Output to the Printer

How you print is now a function of your operating system. To print using Windows 95/98 or NT, you have to get your output to a print manager in the operating system.

To direct output to a printer, you must first open a file stream to send output to. You can do this using the same method that you use to send output to a file. First, declare a variable of type fstream. Then open the stream using the member function, open. Instead of opening the stream using the name of a file, use the name of your printer port, for example, "LPT1."

Once a file stream is open, you can print using the name of the stream variable followed by the << operator and the data you want to send to the printer. Be sure to close the stream when you no longer need to use it.

The example program below will send text to a printer on port LPT1.

```
#include <fstream.h>
```

```
int main()

{
  ofstream print; // stream variable declaration
  print.open("LPT1"); // open stream

  // Print Text (the character '\f' will produce a form
feed)
  print << "This text will print on the printer.\f";
  print.close(); // close stream
  return 0;
}
```

Important Note:

Blindly directing text to a printer port assumes some things are true.

1. It assumes that you have a printer attached to the specified port or that the port is being captured and redirected to a printer.
2. It assumes that the printer is capable of accepting plain text. For example, a PostScript printer will not print unless the data comes through a PostScript printer driver. The code above does not make use of any drivers.

Object Oriented Programming: An Introduction

Magazine Archives - Technologies

Object Oriented Programming: An Introduction

By Shailesh Khanal Jan/Feb, 1994-CORE

Object oriented programming is not just another buzz word of software developers. Object-oriented programming is a relatively new method for designing and implementing software systems. We see them in literature, we hear about them and many of us actually use them without knowing what object oriented software really is. Nowadays softwares based on this concept is pretty ubiquitous, Microsoft's Windows is one of them. Its major goals are to improve programmer productivity by increasing software extensibility and reusability and to control the complexity and cost of software maintenance. It rewards the development of generic functions which can be used with different data types e.g. integer, real etc..

The conventional method of programming using languages as Pascal, basic, C etc. is called imperative or declarative programming. A fundamental flaw of imperative programming is that global variables can potentially be accessed (and updated) by every part of the program. Large programs that lack any discipline tend to be unmanageable. The reason for this is that no module (a group of program codes) that accesses a global variable can be developed and understood independent of other modules that also access that global variable.

Object-oriented programming enables us to create software that can be readily comprehended and shared with others. Unlike more traditional programming methods that are based on concepts such as data flow or mathematical logic, object-oriented programming directly models the application.

There are as many different views of what object-oriented programming is as there are computer scientists and programmers. But basically a language must have following four elements to support object-oriented programming :

• encapsulation
• message passing
• inheritance and
• late biding.

What is an object An object is a logical entity that contains both data and code that manipulates that data within an object, some of the code and/or data may be private to the object and inaccessible to anything outside the object. Thus an object is a variable equipped with operations that have the exclusive rights to access it. Programs perform computations by passing messages between active objects, which are computer analogy of entities in the real world. For example an object oriented financial application might see Customer objects sending debit and credit messages to Accounts objects.

Objects can be reused in similar applications in the same area. Reusing, rather than reinventing software speeds the development and maintenance of large applications.

What is a class Object and class are fundamental concepts of object oriented programming. An object class is a set of objects that share the same operations. In most object-oriented languages, a class definition describes the behavior of the underlying abstract data type by defining the interface to all the operations that can be performed on the underlying type.

Each class definition also specifies the implementation details or data structure of the type. The data structure which are accessible only within the scope of the class is called a private type. And the one which are accessible outside the class are called public.

The operations that are defined on the type are also generally classified as either public or private. The public operations are those that are accessible outside the scope of the class. The private operations are accessible only within the scope of the class.

Methods and message Passing In object-oriented programming, the operations that are defined for a class

are called methods. These methods are analogous to procedures and functions in non-object oriented languages.

Actions may be performed on an object by invoking one or more of the methods defined in the class definition. The process of invoking a method is called sending a message to the object. Such a message typically contains parameters just as in a procedure or functions call invocation in a non-object-oriented languages. The invocation of a method (sending a message to an object) typically modifies the data stored in the particular object.

Encapsulation Encapsulation is the technical name for information hiding. An object in object oriented programming allows some part of its code and/data to be inaccessible outside the object. In this way, an object provides a significant of protection against some other, unrelated part of the program accidentally modifying or incorrectly using the private parts of the object. This linkage of code and data is often referred to as encapsulation.

Instead of organizing programs into procedures that share global data, the data is packaged with the procedures that access data. This concept is often called data abstraction or modular programming. The goal here is to separate the user of the object from its implementer. The user is no longer aware of how the object is implemented. Users can only operate on an object using those messages that the implementer provides. This has the obvious benefit that you can change the implementation of the encapsulated object without affection the applications using it.

Inheritance Inheritance is the major feature distinguishing an OOPS from other programming systems.Every OOPS provides simple inheritance in one form or another. Inheritance increases code sharing by allowing the language rather than the programmer to reuse code form one class in another related class.

Inheritance enables programmers to create classes and, therefore, objects that are specializations of other objects. For example, we might create an object, vehicle, that is a specialization of a car, which is a specialization of sports car. A sports car inherits behavior that is appropriate for car, and vehicle.

Creating a specialization of an existing class is called subclass.The new class is a subclass of the existing class, and the existing class is the superclass of the new class. The subclass inherits instance variables,class variables and methods from its superclass. The subclass may add instance variables, class variables and methods that are appropriate to more specialized objects.

Late binding and Polymorphism It is quite common in object-oriented systems to code multiple classes of an object that respond to the same messages. The ability of different objects to respond differently to the same message is known as polymorphism. This approach is clearly superior to using a huge case statement for all the known graphical data types in a single global draw procedure. Polymorphism is partly responsible for a well-known characteristic of object-oriented systems, a style of programming sometimes referred to as differential programming or programming by modifications.

At the design level, when a software engineer is deciding what type of action is appropriate for a given object, he or she should not be concerned about how the object interprets the action (message) and implements the methods but only what the effect of the action is on the object. Object-oriented languages like C++ and Smalltalk allow a programmer to send identical messages to dissimilar but related objects and achieve identical actions while letting the software system decide how to achieve the required action for the given object. A key issue associated with polymorphism is the timing of the software system's implementations decision. If the system decides how to implement an action at compile time, this is called early binding. If the decision is made dynamically at run-time,it is called late biding. Generally late biding offers the advantages of flexibility and a high level of problem abstraction.

C++ and Object-Oriented Programming C++ is based on C and is a superset of C. C++ retains C's power and flexibility to deal with the hardware-software interface and low-level systems programming. The efficiency, economy, and power of expression of C are retained in C++. More importantly, however, C++ provides a platform to support object-oriented programming and high-level problem abstraction. C++ goes further than ADA in its support for object-oriented programming and is similar to MODULAR-2 in its simplicity and support for modularity,while retaining the cutting-edge efficiency and compactness of C.

C++ is a hybrid language. The object-oriented paradigm can be exploited fully to produce a purely object-oriented solution to a problem, or C++ can be treated as a procedural language that contains some additional constructs beyond C. In practice, C++ software reflects both the procedural programming

paradigm as well as the newer object-oriented paradigm. This duality in C++ presents a special challenge to the beginning C++ programmer. Not only is there a new language to learn, but there is also a new way of problem solving and thinking.

C++ was developed by Bjarne Stroustrup at Bell Labs in the early 1980s. C++ is designed to support software development on a large scale. The most significant aspect of the C++ language is its support for the object-oriented programming .

Classes are what make C++ suitable for developing object-oriented programs. The class structure meets the requirements for creating objects. However, object-oriented programming is also supported by C++'s strong typing, operator overloading, and less reliance on the preprocessor. In a more formal definition, a class defines the properties and attributes that depict the actions of an object that is an instance of that class.

Object-Oriented Design Methods At the current stage of evolution, OOD methodology combines elements of all three design categories : data design, architectural design, and procedural design. By identifying objects, data abstractions are created. By defining operations, modules are specified and a structure for the software is established.

The steps for OOD are as follows :

Define the problem.

Develop an informal strategy for the software realization of the real-world problem domain.

Formalize the strategy using the following substeps : a. Identify objects and their attributes b. Identify operations that may be applied to objects c. Establish interfaces by showing the relationship between objects and operations. d. Decide on detailed design issues that will provide an implementation description for objects.

Mapping from real-world domain to software domain.

Although object oriented system design is the design method for the future, its effect can be hardly felt on smaller programs (less than 3000 lines). For large applications like Windows you know the effect. You might have noticed that in Windows every pop up menu have similar options e.g. move, size. etc., this means the code is being reused many times which reduces the program length and execution time. And that's what object oriented programming is all about.

Daftar Pustaka

http://www.cplusplus.com, The C++ resources network

http://www.research.att.com/~bs/C++.html, Stroustrup: C++

http://cis.stvincent.edu/carlsond/swdesign/swd.html, software design using C++