modul_kuliah

PEMROGRAMAN BERORIENTASI OBJEK

dengan C++

oleh :

L.Putu Ayu Prapitasari

STMIK STIKOM BALI

Denpasar, Bali
2005

MODUL 1

(Waktu : 2 x pertemuan)

1. PENGANTAR C++

C++ diciptakan oleh Bjarne Stroustrup di laboratorium Bell pada awal tahun 1980-an, sebagai pengembangan dari bahasa C dan Simula. Saat ini, C++ merupakan salah satu bahasa yang paling populer untuk pengembangan software berbasis OOP.

Kompiler untuk C++ telah banyak beredar di pasaran. Software developer yang paling diminati adalah Borland Inc. dan Microsoft Corp.

Produk dari Borland untuk kompiler C++ adalah Turbo C++, Borland C++, Borland C++ Builder. Sedangkan dari Microsoft adalah Ms. Visual C++. Walaupun banyak kompiler yang tersedia, namun pada intinya bahasa pemrograman yang dipakai adalah C++.

≈ Sebelum mulai melakukan kode program, sebaiknya diingat bahwa C++ bersifat "***case sensitive***", yang artinya huruf besar dan huruf kecil dibedakan.

2. STRUKTUR BAHASA C++

Cara terbaik untuk balajar bahasa pemrograman adalah dengan langsung mempraktikannya. Cobalah contoh program berikut :

```
// program pertamaku

#include <iostream.h>

int main ()
{
  cout << "Selamat Belajar C++";
  return 0;
}
```

Program di atas, misalnya dapat disimpan dengan nama *latih1.cpp*. Cara untuk menyimpan dan mengkompile program berbeda-beda, tergantung kompiler yang dipakai.

Ketika di-*run*, maka di layar akan muncul sebuah tulisan "Selamat Belajar C++". Contoh di atas, adalah sebuah contoh program sederhana menggunakan C++. Namun, penggalan program tersebut telah menyertakan sintak-sintak dasar bahasa C++. Sintak dasar tersebut, akan kita bahas satu per satu:

// program pertamaku

merupakan sebuah baris komentar. Semua baris, yang ditandai dengan dua buah tanda slash (//), akan dianggap sebagai baris komentar dan tidak akan berpengaruh pada hasil. Biasanya, baris komentar dipakai oleh programmer untuk memberikan penjelasan tentang program.

Baris komentar dalam C++, selain ditandai dengan (//) juga dapat ditandai dengan (/*….*/)

Perbedaan mendasar dari keduanya adalah :

```
// baris komentar
/* blok komentar */
```

#include <iostream.h>

pernyataan yang diawali dengan tanda (#) merupakan pernyataan untuk menyertakan preprocessor. Pernyataan ini bukan untuk dieksekusi. `#include <iostream.h>` berarti memerintahkan kompiler untuk menyertakan file header `iostream.h`. Dalam file header ini, terdapat beberapa fungsi standar yang dipakai dalam proses input dan output. Seperti misalnya perintah *cout* yang dipakai dalam program utama.

`int main ()`

baris ini menandai dimulainya kompiler akan mengeksekusi program. Atau dengan kata lain, pernyataan **main** sebagai penanda program utama. Adalah suatu keharusan, dimana sebuah program yang ditulis dalam bahasa C++ memiliki sebuah **main**.

main diikuti oleh sebuah tanda kurung () karena main merupakan sebuah fungsi. Dalam bahasa C++ sebuah fungsi harus diikuti dengan tanda (), yang

nantinya dapat berisi argumen. Dan sintak formalnya, sebuah fungsi dimulai dengan tanda {}, seperti dalam contoh program.

cout << "Selamat Belajar C++";

perintah ini merupakan hal yang akan dieksekusi oleh compiler dan merupakan perintah yang akan dikerjakan. `cout` termasuk dalam file iostream.

`cout` merupakan perintah untuk menampilkan ke layer.

Perlu diingat, bahwa setiap pernyataan dalam C++ harus diakhiri dengan tanda semicolon (;) untuk memisahkan antara pernyataan satu dengan pernyataan lainnya.

return 0;

pernyataan `return` akan menyebabkan fungsi `main()` menghentikan program dan mengembalikan nilai kepada `main`. Dalam hal ini, yang dikembalikan adalah nilai 0. Mengenai pengembalian nilai, akan dijelaskan nanti mengenai **Fungsi** dalam C++.

Coba tambahkan sebaris pernyataan lagi, sehingga program contoh di atas akan menjadi seperti berikut:

```
// latihan keduaku

#include <iostream.h>

int main ()
{
  cout << "Selamat Belajar C++";
  cout << "di kampusku";
  return 0;
}
```

Maka perintah `cout` yang kedua akan menampilkan sebuah kalimat lagi di layar, dengan tulisan “di kampusku”.

3. TIPE DATA

Terdapat 5 tipe data bawaan dari bahasa C, yaitu : **void**, **integer**, **float**, **double**, dan **char**.

Type	Keterangan
void	diartikan sebagai tanpa tipe data dan tanpa pengembalian nilai
int	bilangan bulat (integer)
float	bilangan pecahan (floating point)
double	bilangan pecahan dengan jangkauan data yang lebih luas
char	Karakter

Sedangkan C++ sendiri menambahkan dua buah tipe data lagi, yakni : **bool** dan **wchar_t**.

Type	Keterangan
bool	isi bilangan Boolean (True dan False)
wchar_t	wide character

Dengan jangkauannya adalah sebagai berikut :

Tipe	Ukuran (bits)	Range
unsigned char	8	0 s/d 255
char	8	-128 s/d 127
short int	16	-32,768 s/d 32,767
unsigned int	32	0 s/d 4,294,967,295
int	32	-2,147,483,648 s/d 2,147,483,647
unsigned long	32	0 s/d 4,294,697,295
long	32	-2,147,483,648 s/d 2,147,483,647
float	32	3.4 e-38 s/d 1.7 E +38
double	64	1.7 E-308 s/d 3.4 E + 308
long double	80	3.4 E-4932 s/d 1.1 E + 4932

4. VARIABEL

Berbeda dengan pendeklarasian variabel di bahasa pemrograman lain, dalam C++ sebelum mendeklarasikan variabel, hal pertama yang harus dideklarasikan adalah tipe data yang akan digunakan untuk menampung data.

Format penulisannya adalah :

```
Tipe_data pengenal = nilai ;
```

Sebagai contoh :

```
int a;
float nomor;
```

atau dapat juga pemberian nilai awal untuk variable dilakukan pada saat deklarasi, contoh :

```
int a=10;
char s='a';
```

Jika hendak mendeklarasikan beberapa variabel sekaligus dengan tipe data yang sama, dapat dilakukan dengan 2 cara, yaitu :

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

atau dapat disederhanakan dengan deklarasi :

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

Perhatikan contoh berikut:

```
// bekerja dengan variabel

#include <iostream.h>

int main ()
{
  // inisialisasi variabel :
  int a, b;
  int hasil;

  // proses :
  a = 5;
  b = 2;
  a = a + 1;
  hasil = a - b;

  // cetak hasilnya :
  cout << hasil;

  // menghentikan program :
  return 0;
}
```

5. KONSTANTA

Konstanta mirip dengan variable, namun memiliki nilai tetap. Konstanta dapat berupa nilai Integer, Float, Karakter dan String.

Pendeklarasian konstanta dapat dilakukan dengan 2 cara :

- menggunakan (**`#define`**)

deklarasi konstanta dengan cara ini, lebih gampang dilakukan karena akan menyertakan **`#define`** sebagai preprocessor directive. Dan sintaknya diletakkan bersama – sama dengan pernyataan `#include` (di atas `main()`).

Format penulisannya adalah :

```
#define pengenal nilai
```

Contoh penggunaan :

```
#define phi 3.14159265
#define Newline '\n'
#define lebar 100
```

pendeklarasian dengan `#define` tanpa diperlukan adanya tanda = untuk memasukkan nilai ke dalam pengenal dan juga tanpa diakhiri dengan tanda semicolon(;).

- menggunakan (**`const`**)

Sedangkan dengan kata kunci `const`, pendeklarasian konstanta mirip dengan deklarasi variable yang ditambah kata depan `const`.

Contoh :

```
const int lebar = 100;
const char tab = '\t';
const zip = 1212;
```

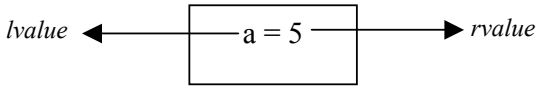
Untuk contoh terakhir, deklarasi variable `zip` yang tanpa tipe data, maka compiler akan secara otomatis memasukkannya ke dalam tipe `int`.

6. OPERATOR

Dalam C++, terdapat berbagai macam operator yang dapat dimanfaatkan dalam aplikasi.

- **Operator Assign** (=)

Operator (=), akan memberikan nilai ke dalam suatu variable.



artinya memberikan nilai 5 ke dalam variable a. Sebelah kiri tanda = dalam pernyataan di atas, dikenal dengan ***lvalue*** (left value) dan di sebelah kanan tanda = dikenal dengan ***rvalue*** (right value). lvalue harus selalu berupa variable, sedangkan rvalue dapat berupa variable, nilai, konstanta, hasil operasi ataupun kombinasinya.

- **Operator Aritmatika** (`+, -, *, /, %`)

Operator	Keterangan
+	Penjumlahan
-	Pengurangan
*	Perkalian
/	Pembagian
%	Modulus

Untuk operator %, sama dengan modulus, yaitu untuk mengetahui sisa hasil bagi. Misalnya a = 11 % 3, maka variable a akan terisi nilai 2 karena sisa hasil bagi 11 dan 3 adalah 2.

- **Operator Majemuk** (`+=, -=, *=, /=, %=, <<=, >>=, &=, |=`)

Dalam C++, operasi aritmatika dapat disederhanakan penulisannya dengan format penulisan operator majemuk.

Misalnya :

`a += 5` sama artinya dengan menuliskan `a = a+5`

`a *= 5` sama artinya dengan menuliskan `a = a*5`

`a /= 5` sama artinya dengan menuliskan `a = a/5`

`a %= 5` sama artinya dengan menuliskan `a = a % 5`

- **Operator Penaikan dan Penurunan** (++ dan --)

  Operator penaikan (++) akan menaikkan atau menambahkan 1 nilai variable. Sedangkan operator (--) akan menurunkan atau mengurangi 1 nilai variable.

  Misalnya :

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

  untuk ketiga pernyataan tersebut, memiliki arti yang sama yaitu menaikkan 1nilai variable 1.

  Karakteristik dari operator ini adalah dapat dipakai di awal (++a) atau di akhir (--a) variable. Untuk penggunaan biasa, mungkin tidak akan ditemui perbedaan hasil dari cara penulisannya. Namun untuk beberapa operasi nantinya harus diperhatikan cara peletakan operator ini, karena akan berpengaruh terhadap hasil.

  Contoh 1 :

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

  Contoh 2:

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

  Dari contoh1, nilai B dinaikkan sebelum dikopi ke variable A. Sedangkan pada contoh2, nilai B dikopi terlebih dahulu ke variable A baru kemudian dinaikkan.

- **Operator Relasional** (==, !=, >, <, >=, <=)

  Yang dihasilkan dari operator ini bukan berupa sebuah nilai, namun berupa bilangan bool yaitu benar atau salah.

Operator	Keterangan
==	Sama dengan
!=	Tidak sama dengan
>	Lebih besar dari
<	Kurang dari

>=	Lebih besar dari atau sama dengan
<=	Kurang dari atau sama dengan

Contoh :

`(7==5)` hasilnya adalah **`false`**

`(5>4)` hasilnya adalah **`true`**

`(5<5)` hasilnya adalah **`false`**

- **Operator Logika** (`!, &&, ||`)

  Operator logika juga digunakan untuk memberikan nilai atau kondisi **`true`** dan **`false`**. Biasanya operator logika dipakai untuk membandingkan dua kondisi. Misalnya:

  ((5==5) && (3>6)) mengembalikan nilai **`false`**, karena (`true && false`)

  untuk logika NOT (!), contohnya !(5==5) akan mengembalikan nilai **`false`**, karena `!(true)`.

- **Operator Kondisional** (`?`)

  Format penulisan operator kondisional adalah :

  ```
  kondisi ? hasil1 : hasil2
  ```

  Jika kondisi benar maka yang dijalankan adalah hasil1 dan jika kondisi salah, maka akan dijalankan hasil2.

  Contoh :

  `7==5 ? 4 : 3` hasilnya adalah **3**, karena 7 tidak sama dengan 5

  `5>3 ? a : b` hasilnya adalah a, karena **5** lebih besar dari 3

MODUL 2
INPUT & OUTPUT STANDAR

(Waktu : 1 x pertemuan)

Dalam library C++, iostream mendukung dua operasi dasar yaitu `cout` untuk output dan `cin` untuk input. Biasanya, dengan perintah `cout` akan menampilkan sesuatu ke layar monitor dan dengan perintah `cin` akan menerima masukan melalui keyboard.

1. Output (`cout`)

Untuk cout menggunakan operator << (*insertion operation*).

```
cout << "Selamat Datang"; //mencetak tulisan Selamat datang ke layar
cout << 120;                    //mencetak angka 120 ke layar
cout << x;                  //mencetak isi nilai variable x ke layer
```

Operator << dikenal sebagai *insertion operator* yang memberikan perintah kepada cout. Untuk contoh pertama, kalimat yang akan di cetak di layar di apit oleh tanda " " karena berupa string. Sedangkan untuk contoh kedua dan ketiga, tanpa tanda " ", karena yang akan ditampilkan ke layar bukan berupa string ataupun karakter.

Sebagai contoh, perhatikan perbedaan dua pernyataan berikut:

```
cout << "Hello";     //menampilkan tulisan Hello ke layar
cout << Hello; //menampilkan isi dari variable Hello ke layar
```

Insertion Operation (<<) dapat digunakan lebih dari satu dalam sebuah pernyataan :

```
cout << "Halo, "<<" saya "<<" belajar C++ ";
```

dengan perintah di atas, maka dilayar akan muncul pesan `Halo, saya belajar C++`.

Selanjutnya, dapat juga dikombinasikan dengan variable. Misalnya :

```
cout << "Halo, saya berusia"<<age<<" tahun ";
```

maka tampilan di layar, adalah sebagai berikut:

```
Halo, saya berusia 23 tahun
```

Yang paling penting dari cout adalah bahwa perintah ini tidak akan menambahkan perintah ganti baris. Untuk membuktikannya, perhatikan contoh berikut:

```
cout<<"kalimat pertama.";
cout<<"kalimat kedua.";
```

maka di layar akan tampil:

```
kalimat pertama.kalimat kedua.
```

Untuk menambahkan perintah ganti baris, ada dua perintah yang dapat dipakai:

```
cout<<"kalimat pertama.\n";
cout<<"kalimat kedua.\nkalimat ketiga.";
```

tampilan di layar adalah sebagai berikut:

```
kalimat pertama.
kalimat kedua.
kalimat ketiga.
```

atau dapat juga dengan menggunakan perintah `endl`:

```
cout<<"kalimat pertama."<<endl;
cout<<"kalimat kedua."<<endl;
```

tampilan dilayarnya adalah sebagai berikut:

```
kalimat pertama.
kalimat kedua.
```

2. Input (`cin`)

Untuk menerima inputan dengan perintah cin, maka operator yang akan digunakan adalah overloaded operator (>>) dan diikuti oleh variable tempat menyimpan inputan data. Seperti contoh:

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

`cin` hanya dapat diproses setelah penekanan tombol ENTER. Jadi, walaupun hanya satu karakter yang dimasukkan, sebelum penekanan Enter, `cin` tidak akan merespon apa-apa.

`cin` juga dapat digunakan menerima beberapa inputan dalam sekali pernyataan :

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

sama dengan pernyataan :

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

kedua pernyataan di atas, jika dijalankan akan meminta dua kali inputan data. Satu untuk variable a dan satunya lagi adalah untuk variable b. Dan untuk pemasukan datanya dipisahkan dengan pemisah, misalnya dengan Spasi, Tab atau Enter.

MODUL 3

Waktu : 2 x pertemuan

Dalam sebuah proses program, biasanya terdapat kode penyeleksian kondisi, kode pengulangan program, atau kode untuk pengambilan keputusan. Untuk tujuan tersebut, C++ memberikan berbagai kemudahan dalam sintaknya.

Terdapat sebuah konsep, yakni Blok Instruksi. Sebuah blok dari instruksi merupakan sekelompok instruksi yang dipisahkan dengan tanda semicolon (;) dan berada diantara tanda { dan }.

Untuk Blok Instruksi, penggunaan tanda { dan } boleh ditiadakan. Dengan syarat, hanya pernyataan tunggal yang akan dilaksanakan oleh blok instruksi. Apabila pernyataan yang dijalankan lebih dari satu, maka tanda { dan } wajib disertakan.

1. SELEKSI KONDISIONAL (`if …else…`)

Format penulisannya :

```
if (kondisi) pernyataan;
```

`kondisi` adalah ekspresi yang akan dibandingkan. Jika `kondisi` bernilai benar, maka `pernyataan` akan dijalankan. Namun, jika `kondisi` bernilai salah, maka `pernyataan` akan diabaikan.

Contoh pernyataan berikut akan menampilkan tulisan `x adalah 100` apabila `x` bernilai 100:

```
if (x==100)
cout << “x adalah 100”;
```

Jika menginginkan lebih dari sebuah pernyataan yang dijalankan, ketika kondisi terpenuhi maka blok instruksi harus menyertakan tanda { dan }.

```
if (x==100)
{
      cout << “x adalah ”;
      cout << x;
}
```

Bila menginginkan sesuatu dijalankan ketika kondisi tidak terpenuhi, dapat ditambahkan keyword ***else***.

Sintaknya adalah :

```
if (kondisi)
      pernyataan1;
else
      pernyataan2;
```

Contoh :

```
if (x==100)
      cout <<"x adalah 100";
else
      cout <<"x bukan 100";
```

Pernyataan `if…else…` dapat terdiri dari beberapa `else`. Pada contoh berikut, program akan memberikan jawaban terhadap inputan data, apakah berupa nilai positif, negative atau nol :

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

2. PERULANGAN (loops)

Sebuah atau beberapa pernyataan akan dijalankan secara berulang-ulang, selama kondisi terpenuhi.

- Perulangan dengan **while**

  Sintaknya adalah :

  ```
  while (kondisi) pernyataan;
  ```

  `pernyataan` akan dijalankan selama `ekspresi` bernilai **true**.

  Contoh :

```
//hitungan mundur menggunakan while
#include<iostream.h>
int main()
{
      int n;
      cout<<"Masukkan angka untuk mulai : ";
      cin>>n;
      while (n>0)
      {
            cout << n << ", ";
            --n;
      }
      cout <<"STOP!";
      return 0;
}
```

Di layar akan tampil :

`Masukkan angka untuk mulai : 4`

`4, 3, 2, 1, STOP!`

Algoritma untuk perulangan di atas adalah sebagai berikut :

1. User menginputkan sebuah nilai ke variable `n`.
2. Pernyataan `while` akan melakukan pengecekan apakah (`n>0`) ?.
   Dalam kondisi ini, terdapat 2 kemungkinan :
   a. **true** : lakukan pernyataan (langkah **3**)
   b. **false** : lompati pernyataan (lanjutkan ke langkah **5..**)
3. Lakukan perintah :
   ```
   cout << n << ", ";
   --n;
   ```
   (cetak n ke layar, dan turunkan 1 nilai n)
4. Akhiri blok. Kembali lagi ke langkah **2..**
5. Lanjutkan program setelah blok `while`. Cetak `STOP!` Dan akhiri program.

- Perulangan dengan **do…while**

  Sintaknya :

  do *pernyataan* **while** (*kondisi*);

Konsep `do…while` mirip dengan `while`. Namun pernyataan akan dijalankan terlebih dahulu sebelum pengecekan kondisi. Untuk lebih jelasnya, perhatikan contoh berikut:

```
//sampai penekanan 0
#include<iostream.h>
int main()
{
      unsigned long n;
      do {
          cout << "Masukkan nomor (tekan 0 untuk mengakhiri) : ";
          cin>>n;
          cout<<"Anda memasukkan angka : " <<n<<"\n";
      } while (n!=0);
      return 0;
}
```

Akan tampil :

```
Masukkan nomor (tekan 0 untuk mengakhiri) : 67
Anda memasukkan angka : 67
Masukkan nomor (tekan 0 untuk mengakhiri) : 12
Anda memasukkan angka : 12
Masukkan nomor (tekan 0 untuk mengakhiri) : 0
Anda memasukkan angka : 0
```

- Perulangan dengan **for**

  Sintaknya :

  `for (inisialisasi; kondisi; counter) pernyataan;`

  Pernyataan akan diulangi jika kondisi bernilai true. Sama seperti perulangan dengan while. Namun for menetapkan inisialisasi dan penaikan berada dalam (dan).

Penjelasannya adalah sebagai berikut:

1. *Inisialisasi:* akan dieksekusi. Biasanya merupakan variable yang akan dipakai sebagai counter atau pencacah. Bagian ini akan dieksekusi hanya sekali.
2. *Kondisi:* akan diperiksa, jika bernilai true maka perulangan akan dilanjutkan dan jika bernilai false maka perulangan akan dilewati.
3. *Pernyataan:* akan dieksekusi. Biasanya dapat terdiri dari sebuah instruksi atau blok instruksi yang berada di antara { dan }.
4. Terakhir, apapun perintah dalam *counter* akan dijalankan dan kemudian perulangan kembali lagi ke langkah 2.

Contoh :

```
// hitungan mundur dengan for

#include<iostream.h>
int main()
{
      for(int n=10; n>0;n--)
      { cout<<n<<", ";
      }
      cout<<"STOP!";
      return 0;
}
```

Hasilnya adalah :

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

Inisialisasi dan *Counter* adalah optional atau dapat ditiadakan. Namun tidak demikian dengan tanda semicolon (;). Misalnya kita dapat menuliskan: for(;n<10;) jika tanpa *inisialisasi* dan tanpa *penaikan*, atau for(;n<10;n++) jika tanpa *inisialisasi* namun tetap menggunakan *penaikan*.

Nested Loops (Perulangan Bertumpuk)

Perulangan bertumpuk secara sederhana dapat diartikan : terdapat satu atau lebih loop di dalam sebuah loop. Banyaknya tingkatan perulangan, tergantung dari kebutuhan.

Biasanya, nested loops digunakan untuk membuat aplikasi matematika yang menggunakan baris dan kolom. Loop luar, biasanya digunakan untuk mendefinisikan baris. Sedangkan loop dalam, digunakan untuk mendefinisikan kolom.

Contoh:

```
for(int baris = 1; baris <= 4; baris++)
{
      for (int kolom = 1; kolom <= 5; kolom++)
      {
            cout<<kolom<<" ";
      }
      cout<<endl;
}
```

Penjelasan program:

Perulangan akan menghasilkan nilai sebagai berikut :

baris =1 ; kolom = 1; cetak 1
kolom = 2; cetak 2
kolom = 3; cetak 3
kolom = 4; cetak 4
kolom = 5 ; cetak 5

ganti baris !

baris =2 ; kolom = 1; cetak 1
kolom = 2; cetak 2
kolom = 3; cetak 3
kolom = 4; cetak 4
kolom = 5 ; cetak 5

ganti baris !

baris =3 ; kolom = 1; cetak 1
kolom = 2; cetak 2
kolom = 3; cetak 3
kolom = 4; cetak 4
kolom = 5 ; cetak 5
ganti baris !
baris =4 ; kolom = 1; cetak 1
kolom = 2; cetak 2
kolom = 3; cetak 3
kolom = 4; cetak 4
kolom = 5 ; cetak 5
ganti baris !
selesai.

Dan di layar akan muncul hasil dengan bentuk matrik sebagai berikut:

1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5

Tambahan :

- perintah **break**

  break berfungsi untuk keluar dari loop, walaupun kondisinya belum seluruhnya terpenuhi. Biasanya, perintah ini digunakan untuk memaksa program keluar dari loop. Contoh berikut akan berhenti menghitung sebelum terhenti secara normal.

```
for (int n=10; n>0;n--)
{
             cout<<n<<", ";
   if (n==3)
   {
             cout<<"penghitungan dihentikan !";
      break;
   }
}
```

dan di layar akan tampak hasil sebagai berikut :

```
10, 9, 8, 7, 6, 5, 4, 3, penghitungan dihentikan !
```

- perintah **continue**

perintah ini akan melewati satu iterasi yang sesuai dengan syarat tertentu, dan melanjutkan ke iterasi berikutnya.

Contoh:

```
for (int n=10; n>0;n--)
{
             if (n==5) continue;
   cout<<n<<", ";
}
cout<<"STOP !";
```

dan di layar akan muncul :

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

Struktur Selektif dengan **switch**

Logika menggunakan switch sama dengan menggunakan perintah if yang telah dijelaskan sebelumnya.

Sintaknya adalah :

```
switch (pilihan)
{
case nilai1 :
          blok pernyataan 1
          break;
case nilai2 :
          blok pernyataan 2
          break;
-
-
-
default :
          blok pernyataan default
}
```

Cara kerjanya:

1. **switch** akan mengevaluasi *pilihan* dan apabila isinya sama dengan *nilai1*, maka *blok pernyataan 1* akan dijalankan sampai menemukan perintah **break** untuk kemudian keluar dari blok switch.
2. Bila pilihan tidak sama isinya dengan nilai1, maka akan dicocokkan lagi dengan *nilai2*. dan apabila isinya sama dengan *nilai2*, maka *blok pernyataan 2* akan dijalankan sampai menemukan perintah **break** untuk kemudian keluar dari blok switch.
3. Terakhir, apabila isi pilihan tidak sesuai dengan *nilai1*, *nilai2* dan seterusnya maka secara otomatis yang dijalankan adalah *blok pernyataan default*.

Contoh (*kedua penggalan program memiliki arti yang sama*) :

contoh **switch**

```
switch (x) {
case 1:
     cout<<"x adalah 1";
     break;
case 2:
     cout<<"x adalah 2";
     break;
default:
     cout<<"tidak teridentifikasi";
}
```

contoh **if**

```
if (x==1) {
     cout<<"x adalah 1";
     }
else if(x==2) {
     cout<<"x adalah 2";
     }
else {
     cout<<"tidak teridentifikasi";

}
```

Sedangkan untuk program yang memiliki beberapa nilai pilihan, maka dapat ditulis seperti contoh berikut:

```
switch (x) {
case 1:
case 2:
case 3:
          cout<<”x = 1, 2 atau 3”;
          break;
default:
          cout<<”x tidak sama dengan 1, 2 atau 3 ”;
}
```

MODUL 4

Waktu : 2 x pertemuan

FUNGSI (1)

Dengan mempergunakan fungsi, maka struktur program akan terlihat lebih ramping. Fungsi merupakan sebuah blok instruksi yang dieksekusi dan dipanggil dari bagian lain tubuh program.

Format penulisannya adalah sebagai berikut:

tipe nama(argumen1, argumen2,…) pernyataan;

Dimana:

tipe berisi tipe data yang akan dikembalikan oleh fungsi

nama merupakan pengenal untuk memanggil fungsi

argumen (dapat dideklarasikan sesuai dengan kebutuhan). Tiap-tiap argumen terdiri dari tipe-tipe data yang diikuti oleh pengenalnya. Sama seperti mendeklarasikan variable baru (contoh, `int x`).

pernyataan merupakan bagian tubuh fungsi. Dapat berupa pernyataan tunggal ataupun pernyataan majemuk.

Contoh penggunaan fungsi:

```
//contoh fungsi
#include <iostream.h>
#include <conio.h>
int penjumlahan(int a, int b)
{  int r;
   r=a+b;
   return (r);
}
int main()
{  int z;
   z=penjumlahan(5,3);
   cout<<"Hasil penjumlahan = " << z;
   return 0;
}
```

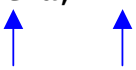
Hasil eksekusinya adalah :

```
Hasil penjumlahan = 8
```

Ketika program dieksekusi, yang dijalankan pertama kali adalah fingsi main(). Terlihat jelas bahwa dalam main() terdapat variable z dengan tipe data integer. Setelah itu, fungsi **penjumlahan** dipanggil. Maka akan terdapat proses pertukaran data sebagai berikut:

```
int penjumlahan(int a, int b)

z = penjumlahan ( 5  ,  3 );
```



maka setelah terjadi pengisian nilai, variable a akan terisi dengan nilai 5 dan variable b akan terisi dengan nilai 3.

Fungsi **penjumlahan** mendeklarasikan sebuah variable baru lagi (`int r;`) dan kemudian menjumlahkan nilai `r=a+b;` dengan hasil akhir variable r = 8. Karena masing-masing variable a dan b sudah terisi dengan nilai 5 dan 3.

Kode :

```
return (r);
```

merupakan pengakhir fungsi **penjumlahan** dan memberikan hasil akhir nilai r kepada fungsi yang memanggilnya (dalam hal ini fungsi **main()**). Proses pengembalian nilai dapat digambarkan sebagai berikut:

```
int penjumlahan(int a, int b)
8
z = penjumlahan ( 5  ,  3 );
```

maka ketika perintah `cout<<"Hasil penjumlahan = " << z;` dijalankan, hasilnya adalah 8.

Scope (Batasan) Variabel

Variabel yang dideklarasikan di dalam tubuh fungsi, hanya dapat diakses oleh fungsi itu. Dan tidak dapat dipergunakan di luar fungsi.

Contoh:

Pada program sebelumnya, variable a dan b atau r tidak dapat digunakan dalam fungsi main(), sebab variable tersebut merupakan **variable lokal** dalam fungsi penjumlahan.

Demikian juga halnya dengan variable z. Tidak dapat dipergunakan dalam fungsi penjumlahan karena merupakan variable lokal dalam fungsi main().

Selain variable lokal, terdapat pula **variable global** yang dapat diakses dari mana saja. Dari dalam maupun luar tubuh fungsi. Untuk mendeklarasikan variable global, harus dituliskan di luar fungsi atau blok instruksi.

Contoh lain fungsi:

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

int kurang(int a, int b)
{
      int r;
      r=a-b;
      return (r);
}

int main()
{
      int x= 5, y=3, z;
      z=kurang(7,2);
      cout<<"Pertama : " << z<<endl;
      cout<<"Kedua : " << kurang(7,2)<<endl;
      cout<<"Ketiga : " << kurang (x,y) <<endl;
      z=4+kurang(x,y);
      cout<<"Keempat : "<<z<<endl;
      return 0;
}
```

Hasilnya adalah :

```
Pertama : 5
Kedua : 5
Ketiga : 2
Keempat : 6
```

Dalam program di atas, terdapat sebuah fungsi yang dinamakan **kurang**. Fungsi ini mengerjakan tugas untuk mengurangkan nilai dua buah variable dan kemudian mengembalikan hasilnya.

Sedangkan di dalam fungsi main(), terdapat beberapa kali pemanggilan terhadap fungsi kurang. Disana terdapat dengan jelas perbedaan cara pengaksesan dan pengaruh terhadap hasilnya.

Untuk lebih memperjelas, beberapa sintak akan dijelaskan secara rinci sebagai berikut:

```
z=kurang(7,2);
cout<<"Pertama : " << z<<endl;
```

Jika diganti dengan hasil (sesuai dengan pemanggilan fungsinya), maka akan didapatkan baris:

```
z=5;
cout<<"Pertama : " << z<<endl;
```

sama seperti baris:

```
cout<<"Kedua : " << kurang(7,2)<<endl;
```

jika sesuai dengan pemanggilan fungsinya, maka akan didapatkan:

```
cout<<"Kedua : " << kurang(7,2)<<endl;
```

sedangkan untuk:

```
cout<<"Ketiga : " << kurang (x,y) <<endl;
```

maka isi dari variabel x=5 dan y=3, sehingga dapat diartikan sebagai baris:

```
cout<<"Ketiga : " << kurang (5,3) <<endl;
```

demikian juga halnya dengan baris:

```
z=4+kurang(x,y);
```

dapat diartikan dengan baris:

```
z=4+kurang(5,3);
```

Pendeklarasian fungsi tanpa tipe (menggunakan *void*)

Kadang-kadang terdapat fungsi yang tanpa memerlukan adanya pengembalian nilai. Misalkan, sebuah fungsi yang hanya bertugas mencetak kalimat ke layar monitor dan tanpa memerlukan adanya pertukaran parameter. Dalam kondisi seperti ini, maka dipergunakan kata kunci ***void***.

Contoh program:

```
#include<iostream.h>
#include<conio.h>
void Ucapan(void)
{
      cout<<"Selamat Belajar C++";
}
int main()
{
      Ucapan();
      return 0;
}
```

Yang harus diperhatikan adalah, pemanggilan fungsi harus disertai dengan tanda (). Seperti contoh di atas, fungsi Ucapan walaupun didekarasikan tanpa tipe data dan tanpa argumen, dipanggil dalam fungsi main dengan Ucapan().

FUNGSI (2)

Cara pelewatan argumen adalah :

1. Pemanggilan dengan nilai (*arguments passed by value*)
2. Pemanggilan dengan acuan (*arguments passed by reference*)

Sampai saat ini, fungsi-fungsi yang telah dibuat adalah menggunakan pemanggilan argumen berdasarkan nilai. Contoh :

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

Pada penggalan di atas, terjadi pemanggilan terhadap fungsi kurang() dengan x dan y masing-masing bernilai 5 dan 3.

```
int kurang(int a, int b)

z = kurang(  x,    y);
```

Dengan pemanggilan tersebut, maka pengisian nilai terhadap variabel a dan b dilakukan oleh variabel x dan y yaitu a=5 dan b=3. Ketika terjadi pengisian nilai seperti ini, maka nilai x dan y tidak akan mengalami perubahan apapun.

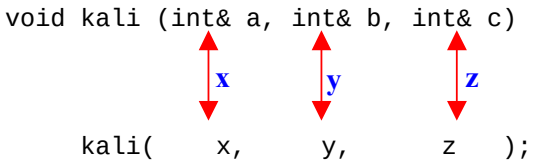
Namun, kadangkala diinginkan sebuah pertukaran nilai yang mempengaruhi nilai variabel pemberinya. Untuk melakukannya, diperlukan sebuah fungsi dengan pertukaran nilai secara acuan (*argumen passed by value*), seperti contoh berikut:

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

void kali (int& a, int& b, int& c)
{
      a *= 2;
      b *= 2;
      c *= 2;
}
int main()
{
      int x=1, y=3, z=7;
      kali(x,y,z);
      cout<< "x= "<<x<<", y= "<<y<<", z= "<<z;
      return 0;
}
```

Hasilnya adalah : **x=2, y=6, z=14**

Yang berbeda adalah cara pertukaran argumen menggunakan tanda ampersand (&), yang artinya fungsi melayani pengisian berdasarkan acuan (reference).



dengan pengisian nilai seperti ini, maka apabila terjadi perubahan nilai pada variable a, b dan c maka akan mempengaruhi nilai variable x, y dan z.

Nilai default dalam argument

Ketika mendeklarasikan fungsi, maka tiap-tiap parameter yang dideklarasikan dapat diberikan sebuah nilai default. Nilai default ini akan dipergunakan bila dalam pemanggilan fungsi, tidak diberikan nilai kepada parameter. Contoh:

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

void bagi (int a, int b=2)
{
      int r;
      r=a/b;
      return (r);
}
int main()
{
      cout<<bagi(12);
      cout<<endl;
      cout<<bagi(20,4);

      return 0;
}
```

Hasilnya : **6**
5

Dalam program di atas, terdapat dua kali pemanggilan terhadap fungsi bagi(),

```
bagi(12)
```

hanya memberikan sebuah nilai kepada fungsi bagi(), sedangkan dalam pendeklarasian fungsinya, bagi() memerlukan 2 buah parameter. Maka variable b akan otomatis bernilai 2 sesuai dengan nilai defaultnya. Sedangkan pada pemanggilan yang kedua,

```
bagi(20,4)
```

Variabel a dan b masing-masing diberikan nilai 20 dan 4. Untuk nilai default, dalam hal ini akan diabaikan.

Fungsi Rekursif

Fungsi rekursif adalah suatu fungsi yang memanggil dirinya sendiri, artinya fungsi tersebut dipanggil di dalam tubuh fungsi itu sendiri. Fungsi rekursif sangat berguna bila diimplementasikan untuk pekerjaan pengurutan data, atau menghitung nilai factorial suatu bilangan. Misalnya:

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

long factorial (long a)
{
      if (a>1)
            return (a* factorial (a-1));
      else
            return (1);
}
int main()
{
      long l;
      cout<<"tuliskan bilangan : ";
      cin>>l;
      cout<<"!"<<l<<" = "<<factorial(l);
      return 0;
}
```

Hasil :

Tuliskan bilangan : 9

!9 = 362880

Prototype Fungsi

Sampai saat ini, setiap dideklarasikan sebuah fungsi baru diletakkan di atas fungsi main(). Namun terdapat pula alternative lain dalam pendeklarasian fungsi baru, yaitu dideklarasikan di bawah fungsi main() dengan menggunakan *prototype fungsi*.

Bagi compiler, informasi dalam prototype akan dipakai untuk memeriksa validitas parameter dalam pemanggilan fungsi.

Keuntungan pemakaian prototype yaitu compiler akan melakukan konversi seandainya antara tipe parameter dalam definisi dan parameter saat pemanggilan fungsi tidak sama, atau akan menunjukkan kesalahan kalau jumlah parameter dalam definisi dan saat pemanggilan berbeda.

Sintak prototype:

```
tipe nama (argumen1, argumen2,…);
```

sama seperti pendeklarasian judul fungsi, kecuali:

1. tidak memiliki baris pernyataan (tubuh fungsi) yang ditandai dengan { dan }.
2. diakhiri dengan tanda ;
3. dalam pendeklarasian argumennya, cukup hanya dengan mendeklarasikan tipe datanya saja. Walaupun sangat dianjurkan untuk menyertakan argumen secara lengkap.

Contoh:

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

void bagi (int a, int b);

int main()
{

      cout<<bagi(20,4);
      return 0;
}

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

untuk pendeklarasian prototype fungsi dapat berbentuk seperti berikut:

```
void bagi (int a, int b);
```

atau

```
void bagi (int , int );
```

MODUL 5

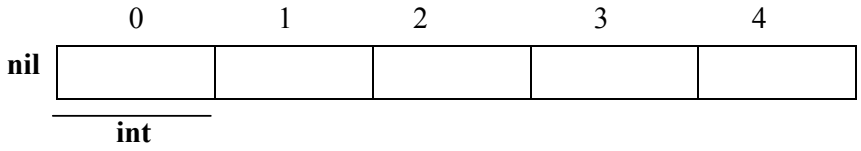
Waktu : 1 x pertemuan

LARIK (ARRAY)

Pada program yang dibahas terdahulu, banyak menggunakan variabel tunggal, artinya sebuah variabel hanya digunakan untuk menyimpan satu nilai.

Array atau yang juga biasa disebut array merupakan koleksi data dimana setiap elemen memakai nama yang sama dan bertipe sama dan setiap elemen diakses dengan membedakan indeks array-nya.

Misal, sebuah array bernama **nil** yang terdiri dari 5 data dengan tipe int, dapat digambarkan sebagai berikut:



Tiap ruang kosong merupakan tempat untuk masing-masing elemen array bertipe integer. Penomorannya berawal dari 0 sampai 4, sebab dalam array index pertama selalu dimulai dengan 0.

Deklarasi array

Sama seperti variabel, array harus dideklarasikan dulu sebelum mulai digunakan. Sintaknya adalah:

```
tipe nama[elemen];
```

Contoh, untuk pendeklarasian array dengan nama **nil** di atas adalah:

```
int nil[5];
```

Inisialisasi Array

Nilai suatu variabel array dapat juga diinisialisasi secara langsung pada saat deklarasi, misalnya:

```
int nil[5] = { 1,3,6,12,24 }
```

Maka di penyimpanan ke dalam array dapat digambarkan sebagai berikut:

	0	1	2	3	4
nil	**1**	**3**	**6**	**12**	**24**

Mengakses nilai array

Untuk mengakses nilai yang terdapat dalam array, mempergunakan sintak:

```
nama[index]
```

Pada contoh di atas, variabel nil memiliki 5 buah elemen yang masing-masing berisi data. Pengaksesan tiap-tiap elemen data adalah:

	nil[0]	nil[1]	nil[2]	nil[3]	nil[4]
nil					

Misal, untuk memberikan nilai 75 pada elemen ke 3, maka pernyataannya adalah:

```
nil[2] = 75
```

atau jika akan memberikan nilai array kepada sebuah variabel a, dapat ditulis:

```
a=nil[2]
```

contoh program:

```
//contoh array
#include<iostream.h>
#include<conio.h>

int nil[] = {16, 2, 77, 40};
int n, hasil = 0;

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

Array Dua Dimensi

Struktur array yang dibahas di atas, mempunyai satu dimensi, sehingga variabelnya disebut dengan variabel array berdimensi satu. Pada bagian ini, ditunjukkan array berdimensi lebih dari satu, yang sering disebut dengan array berdimensi dua.

Sebagai contoh, sebuah matrik B berukuran 2 x 3 dapat dideklarasikan sebagai berikut:

Int b[2][3] = {{2,4,1},{5,3,7}}; yang akan menempati lokasi memori dengan susunan sebagai berikut:

	0	1	2
0	2	4	1
1	5	3	7

Contoh program dengan array dua dimensi

```
//contoh array dua dimensi

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

void main()
{
int matrik[2][3] = {{1,2,3},{4,5,6}}
for (i = 0; i<=2; i++)
{
      for (j=0;j<=3;j++)
      {
            cout<<matrik[i,j];
      }
      cout<<endl;
}
}
```

MODUL 6

Waktu : 1 x pertemuan

String

Dalam pemrograman C++, karena string merupakan kumpulan dari karakter maka untuk inisialisasi string, dapat dilakukan dengan cara sebagai berikut:

```
char namaku[20] ;
```

maka dari pernyataan di atas, dapat digambarkan sebagai deklarasi sebuah variabel string (array dari karakter) dengan panjang hingga 20 karakter, termasuk diakhiri dengan karakter null.

namaku

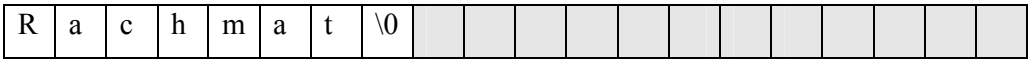
Ukuran maksimum 20 karakter untuk pernyataan di atas, dalam pengisiannya tidak harus penuh.

Contoh, variabel namaku, dapat diisi dengan string “Rachmat” yang panjangnya 7 karakter , atau dapat digantikan dengan string “Johny” yang memiliki panjang 5 karakter.

Dari contoh tersebut, suatu string dapat menyimpan karakter kurang dari panjang totalnya. Dan untuk mengakhiri string, di tiap-tiap akhir akan ditambahkan sebuah karakter null yang dapat ditulis sebagai karakter konstan 0 atau ‘\0’.

Contoh berikut akan memberikan string “Rachmat” dan “Johny” pada variabel namaku.

Namaku



Inisialisasi String

Untuk inisialisasi string (*pemberian nilai kepada variabel string*), dapat dilakukan dengan beberapa cara :

```
char namaku[20] = {‘R’,’a’,’c’,’h’,’m’,’a’,’t’,’\0’};
```

atau

```
char namaku[20];
namaku[0] = ‘R’;
namaku[1] = ’a’;
namaku[2] = ’c’;
namaku[3] = ’h’;
namaku[4] = ’m’;
namaku[5] = ’a’;
namaku[6] = ’t’;
namaku[7] = ’\0’;
```

atau

```
char namaku[20] = “Rachmat”;
```

Perbedaannya, disini adalah pada tanda (‘) yang berarti menginputkan nilai berupa karakter ke dalam variabel string sedangkan tanda (“) berarti menginputkan sebuah nilai string ke dalam variabel string.

Fungsi – fungsi untuk manipulasi string:

Salah satu fungsi yang paling sering digunakan adalah **strcpy**, yaitu fungsi untuk mengkopi isi suatu nilai string ke dalam variabel string lainnya. Fungsi strcpy (**str**ing **c**o**py**) didefinisikan dalam library cstring(file header `string.h`) dan dipanggil dengan cara:

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

dengan cara seperti di atas, maka isi dari string2 akan dikopikan ke dalam string1.

Contoh program

```
// penggunaan strcpy
#include<iostream.h>
#include<conio.h>
int main()
```

```
{
  char namaku[20];
  strcpy(namaku,”Ayu”);
  cout<<namaku;
  return 0;
}
```

Hasil outputnya adalah:

```
Ayu
```

Untuk memberikan nilai kepada sebuah variabel string, biasanya digunakan perintah input stream (**cin**) dan diikuti oleh metode **getline**.

Contoh penggunaannya:

```
// penggunaan cin untuk input string
#include<iostream.h>
int main()
{
  char namaku[20];
  cout<<”Inputkan data nama : ”;
  cin.getline(namaku,20);
  cout<<”Nama anda adalah : ”;
  cout<<namaku;
  return 0;
}
```

Hasil outputnya adalah

```
Inputkan data nama : Ayu
Nama anda adalah : Ayu
```

MODUL 7

Waktu : 1 x pertemuan

Pointer

Setiap kali komputer menyimpan data, maka sistem operasi akan mengorganisasikan lokasi pada memori pada alamat yang unik. Misal untuk alamat memori 1776, hanya sebuah lokasi yang memiliki alamat tersebut. Dan alamat 1776 pasti terletak antara 1775 dan 1777.

Dalam pointer, terdapat 2 jenis operator yang biasa digunakan.

1. ***Operator Alamat / Dereference Operator(&)***

   Setiap variabel yang dideklarasikan, disimpan dalam sebuah lokasi memori dan pengguna biasanya tidak mengetahui di alamat mana data tersebut disimpan.

   Dalam C++, untuk mengetahui alamat tempat penyimpanan data, dapat digunakan tanda ampersand(&) yang dapat diartikan "alamat".

   Contoh :

   ```
   Bil1 = &Bil2;
   ```

   dibaca: isi variabel bil1 sama dengan alamat bil2

2. ***Operator Reference (*)***

   Penggunaan operator ini, berarti mengakses nilai sebuah alamat yang ditunjuk oleh variabel pointer.

   Contoh :

   ```
   Bil1=*Bil2;
   ```

   dibaca: bil1 sama dengan nilai yang ditunjuk oleh bil2

Deklarasi variabel pointer

```
tipe * nama_pointer;
```

tipe merupakan tipe data yang akan ditunjuk oleh variabel, bukan tipe data dari pointer tersebut.

Contoh program menggunakan pointer

```
// program pointer
#include <iostream.h>
int main()
{
        int nil1 = 5, nil2 = 15;
        int *ptr;
        ptr = &nil1;
        *ptr = 10;
        ptr=&nil2;
        *ptr=20
        cout<<”Nilai 1 = ”<<nil1<<”dan nilai 2 = “<<nil2;
        return 0;
}
```

Hasil :

```
Nilai 1 = 10 dan nilai 2 = 20
```

MODUL 8

Waktu : 2 x pertemuan

PEMROGRAMAN BERORIENTASI OBJEK

Pengantar

Pemrograman Berorientasi Objek sebenarnya bukanlah bahasa pemrograman baru, tetapi jalan baru untuk berpikir dan merancang aplikasi yang dapat membantu memecahkan persoalan mengenai pengembangan perangkat lunak. Pemrograman berorientasi objek disusun dan dipahami oleh ilmuwan yang memandang dunia sebagai populasi objek yang berinteraksi dengan yang lain. Prosedur yang digunakan dalam objek dipandang sebagai kepentingan kedua karena tergantung pada objek itu sendiri.

Tentunya hal tersebut berbeda dengan pemrograman terstruktur. Pemrograman terstruktur mempunyai sifat pemisahan data dengan kode yang mengolahnya.

Pemrograman Berorientasi Objek (PBO) adalah metode pemrograman yang meniru cara kita memperlakukan sesuatu(benda). Ada tiga karakteristik bahasa Pemrograman Berorientasi Objek, yaitu :

1. Pengkapsulan (*Encapsulation*) : mengkombinasikan suatu struktur dengan fungsi yang memanipulasinya untuk membentuk tipe data baru yaitu kelas (class).
2. Pewarisan (*Inheritance*) : mendefinisikan suatu kelas dan kemudian menggunakannya untuk membangun hirarki kelas turunan, yang mana masing-masing turunan mewarisi semua akses kode maupun data kelas dasarnya.
3. Polimorphisme (*Polymorphism*) : memberikan satu aksi untuk satu nama yang dipakai bersama pada satu hirarki kelas, yang mana masing-masing kelas hirarki menerapkan cara yang sesuai dengan dirinya.

Struktur dan kelas

Dalam C++, tipe data struktur yang dideklarasikan dengan kata kunci struct, dapat mempunyai komponen dengan sembarang tipe data, baik tipe data dasar maupun tipe data turunan, termasuk fungsi. Dengan kemampuan ini, tipe data struktur menjadi sangat berdaya guna.

Misalnya, kita ingin membentuk tipe data struktur yang namanya kotak. Maka dapat dideklarasikan sebagai berikut:

```
struct tkotak
{
      double panjang;
      double lebar;
};
tkotak kotak;
```

Untuk memberi nilai ukuran kotak tersebut, kita dapat menggunakan perintah-perintah ini:

```
kotak.panjang = 10;
kotak.lebar = 7;
```

Untuk memberi nilai panjang dan lebar kotak, salah satu caranya adalah seperti di atas. Cara lain untuk memberi nilai panjang dan lebar adalah dengan membentuk suatu fungsi. Karena fungsi ini hanya digunakan untuk memberi nilai data panjang dan lebar suatu kotak, tentunya fungsi ini khusus milik objek kotak, sehingga harus dianggap sebagai anggota struktur kotak. C++ sebagai bahasa pemrograman dapat mendefinisikan anggota tipe struktur yang berupa fungsi.

Dengan menambah fungsi tersebut, maka struktur kotak menjadi lebih jelas bentuknya.

```
struct tkotak
{
  double panjang;
       double lebar;
       void SetUkuran(double pj, double lb)
       {
         panjang = pj;
```

```
        lebar = lb;
      };
};
tkotak kotak;
```

dengan tipe struktur kotak seperti itu, untuk memberi nilai panjang dan lebar hanya dengan memanggil fungsi SetUkuran()

```
kotak.SetUkuran(10,7);
```

Selain punya ukuran panjang dan lebar, kotak juga mempunyai keliling dan luas. Dengan demikian, kita dapat memasukkan fungsi untuk menghitung keliling dan luas ke dalam struktur kotak.

Sebagai catatan, bahwa definisi fungsi yang menjadi anggota struktur dapat ditempatkan di luar tubuh struktur. Dengan cara ini maka deklarasi struktur kotak menjadi seperti berikut:

```
struct tkotak
{
  double panjang;
  double lebar;
  void SetUkuran(double pj, double lb);
  double Keliling();
  double Luas();
};
tkotak kotak;
```

contoh penerapan struktur kotak dapat dilihat dalam program berikut:

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

struct tkotak
{
      double panjang;
      double lebar;
      void SetUkuran(double pj, double lb);
      double Keliling();
      double Luas();
};

int main()
{
   tkotak kotak;
   kotak.SetUkuran(10,7);
   cout<<"Panjang : "<<kotak.panjang<<endl;
   cout<<"Lebar : "<<kotak.lebar<<endl;
   cout<<"Keliling : "<<kotak.Keliling()<<endl;
   cout<<"Luas : "<<kotak.Luas()<<endl;
   getch();
   return 0;
}

void tkotak::SetUkuran(double pj, double lb)
{
   panjang = pj;
   lebar = lb;
}

double tkotak::Keliling()
{
   return 2*(panjang+lebar);
}

double tkotak::Luas()
{
   return panjang*lebar;
}
```

Tampilan Output:

```
Panjang : 10
Lebar : 7
Keliling : 34
Luas : 70
```

Bentuk program di atas, adalah contoh gaya pemrograman berorientasi prosedur (terstruktur) yang sudah mengubah pola pikirnya menjadi berorientasi objek. Dalam pemrograman berorientasi objek, jika kita telah menentukan suatu objek tertentu, maka objek tersebut kita definisikan dalam bentuk tipe baru yang namanya kelas.

Tipe data kelas didefinisikan dengan kata kunci (*keyword*) class, yang merupakan generalisasi dari pernyataan struct. Pernyataan struct secara umum digantikan dengan pernyataan class. Jika objek kotak dideklarasikan dalam bentuk kelas, maka deklarasinya mirip dengan struktur.

```
class tkotak
{
      double panjang;
      double lebar;
public:
      void SetUkuran(double pj, double lb);
      double Keliling();
      double Luas();
};
tkotak kotak;
```

Dalam deklarasi kelas tersebut, muncul kata public. Data atau fungsi yang dideklarasikan di bawah kata kunci public mempunyai sifat dapat diakses dari luar kelas secara langsung. Dalam deklarasi tersebut, variabel panjang dan lebar tidak bersifat public, sehingga tidak dapat diakses secara langsung dari luar kelas. Dengan demikian perintah-perintah di bawah ini tidak dapat dijalankan.

```
kotak.panjang = 10;
kotak.lebar = 7;
cout<<”Panjang : ”<<kotak.panjang<<endl;
cout<<”Lebar : “<<kotak.lebar<<endl;
```

Inilah yang membedakan struktur dan kelas. Dalam kelas, masing-masing data dan fungsi anggota diberi sifat tertentu. *Jika semua anggota kelas bersifat public, maka kelas sama dengan struktur*.

Untuk dapat mengakses data panjang dan lebar pada kelas tkotak harus dilakukan oleh fungsi yang menjadi anggota kelas dan bersifat public.

Pada deklarasi kelas tkotak, satu-satunya jalan untuk memberi nilai panjang dan lebar adalah dengan menggunakan fungsi SetUkuran(). Untuk mengambil nilai panjang dan lebar juga harus dilakukan oleh fungsi yang menjadi anggota kelas. Misalnya, kita definisikan fungsi GetPanjang() dan GetLebar() untuk mengambil nilai panjang dan lebar.

Sebagai contoh:

```
//program class
#include<iostream.h>
#include<conio.h>

class tkotak
{
   double panjang;
   double lebar;

public:
   void SetUkuran(double pj, double lb);
   double Keliling();
   double Luas();
   double GetPanjang();
   double GetLebar();
};
int main()
{
   tkotak kotak;
   kotak.SetUkuran(10,7);
   cout<<"Panjang : "<<kotak.GetPanjang()<<endl;
   cout<<"Lebar : "<<kotak.GetLebar()<<endl;
   cout<<"Keliling : "<<kotak.Keliling()<<endl;
   cout<<"Luas : "<<kotak.Luas()<<endl;
   getch();
   return 0;
}
void tkotak::SetUkuran(double pj, double lb)
{
   panjang = pj;
   lebar = lb;
}
double tkotak::Keliling()
{
   return 2*(panjang+lebar);
}
double tkotak::Luas()
{
   return panjang*lebar;
}
double tkotak::GetPanjang()
{
   return panjang;
}
double tkotak::GetLebar()
{
   return lebar;
}
```

Tampilan Output:

```
Panjang : 10
Lebar : 7
Keliling : 34
Luas : 70
```

Dapat dilihat dari contoh program, bentuk pendefinisian kelas adalah sebagai berikut:

```
Tipe Nama_Kelas::NamaFungsi()
{
  IsiFungsi
}
```

Untuk mendefinisikan variabel kelas, digunakan deklarasi :

```
Nama_Kelas Nama_Variabel;
```

Contoh :

```
Tkotak kotak;
```

Pengkapsulan(Encapsulation)

Salah satu keistimewaan C++ adalah pengkapsulan. Pengkapsulan adalah mengkombinasikan suatu struktur dengan fungsi yang memanipulasinya untuk membentuk tipe data baru yaitu kelas(class).

Kelas akan menutup rapat baik data maupun kode. Akses item di dalam kelas dikendalikan. Pengendalian ini tidak hanya berupa data tetapi juga kode. Saat kelas akan digunakan, kelas harus sudah dideklarasikan. Yang penting, pemakai kelas mengetahui deskripsi kelas, tetapi bukan implementasinya. Bagi pemakai, detail internal kelas tidak penting. Konsep ini disebut penyembunyian informasi (*information hiding*).

Untuk menggunakan kelas, kita perlu mengetahui sesuatu tentangnya. Kita perlu mengetahui fungsi apa yang bisa digunakan dan data apa saja yang dapat diakses. Fungsi yang dapat digunakan dan data yang dapat diakses disebut antarmuka pemakai (user interface). Antarmuka pemakai menceritakan bagaimana kelas berperilaku, bukan bagaimana kelas dibuat. Kita tidak perlu mengetahui implementasi kelas. Sekali kelas dibuat, kita bisa memakainya berulang-ulang. Bagi pandangan pemakai, kelas adalah kotak hitam dengan perilaku tertentu.

Kendali Akses terhadap Kelas

Tugas kelas adalah untuk menyembunyikan informasi yang tidak diperlukan oleh pemakai. Ada tiga macam pemakai kelas:

1. kelas itu sendiri

2. pemakai umum
3. kelas turunan

Setiap macam pemakai mempunyai hak aksesnya masing-masing. Hak akses ini ditandai dengan kenampakan anggota kelas. Kelas pada C++ menawarkan tiga aras kenampakan anggota kelas (baik anggota data maupun fungsi anggota):

1. **private**

   Anggota kelas private mempunyai kendali akses yang paling ketat. Dalam bagian private, hanya fungsi anggota dari kelas itu yang dapat mengakses anggota private atau kelas yang dideklarasikan sebagai teman (friend).

2. **public**

   Dalam bagian public, anggotanya dapat diakses oleh fungsi anggota kelas itu sendiri, instance kelas, fungsi anggota kelas turunan. Suatu kelas agar bisa diakses dari luar kelas, misalnya dalam fungsi main(), perlu mempunyai hak akses publik. Hak akses ini yang biasanya digunakan sebagai perantara antara kelas dengan dunia luar.

3. **protected**

   Suatu kelas dapat dibuat berdasarkan kelas lain. Kelas baru ini mewarisi sifat-sifat dari kelas dasarnya. Dengan cara ini bisa dibentuk kelas turunan dari beberapa tingkat kelas dasar. Bila pada kelas dasar mempunyai anggota dengan hak akses terproteksi, maka anggota kelas ini akan dapat juga diakses oleh kelas turunannya. Anggota kelas terproteksi dibentuk dengan didahului kata kunci protected. Pada bagian protected, hanya fungsi anggota dari kelas dan kelas-kelas turunannya yang dapat mengakses anggota.

Contoh program :

```
//File persegiPJ.Cpp
#include <iostream.h>
class PersegiPanjang
{
protected :
      int panjang;
      int lebar;

public :
      PersegiPanjang(int pj, int lb)
      {
            panjang = pj;
         lebar = lb;
      }
      int Panjang()
      {
            return panjang;
      }
      int Lebar()
      {
            return lebar;
      }
      int Keliling()
      {
            return 2*(panjang+lebar);
      }
      int Luas()
      {
            return panjang*lebar;
      }
      void Skala(float sk)
      {
            panjang *= sk;
         lebar *= sk;
      }
      void TulisInfo()
      {
            cout<<"Panjang : "<<panjang<<endl
             <<"Lebar :"<<lebar<<endl
             <<"Keliling :"<<Keliling()<<endl
             <<"Luas :"<<Luas()<<endl;
      }
};
```

Simpan file di atas dengan nama **`PersegiPJ.Cpp`**

Kemudian lanjutkan pembuatan program berikut pada halaman baru:

```
// penggunaan kelas persegi panjang
#include "PersegiPJ.Cpp"
void main()
{
   PersegiPanjang pp1(10,5);
   pp1.Skala(2);
   pp1.TulisInfo();
   getch();

}
```

MODUL 9

Waktu : 1 x pertemuan

Konstruktor dan Destruktor

Konstruktor adalah fungsi khusus anggota kelas yang otomatis dijalankan pada saat penciptaan objek (mendeklarasikan instance). Konstruktor ditandai dengan namanya, yaitu sama dengan nama kelas. Konstruktor tidak mempunyai tipe hasil, bahkan juga bukan bertipe void. Biasanya konstruktor dipakai untuk inisialisasi anggota data dan melakukan operasi lain seperti membuka file dan melakukan alokasi memori secara dinamis. Meskipun konstruktor tidak harus ada di dalam kelas, tetapi jika diperlukan konstruktor dapat lebih dari satu.

Tiga jenis konstruktor:

1. *Konstruktor default* : tidak dapat meneriman argumen, anggota data diberi nilai awal tertentu
2. *Konstruktor penyalinan dengan parameter* : anggota data diberi nilai awal berasal dari parameter.
3. *Konstruktor penyalinan objek lain* : parameter berupa objek lain, anggota data diberi nilai awal dari objek lain.

Perhatikan contoh berikut:

```
//Program Konstruktor
#include<iostream.h>
#include<conio.h>
class titik
{
      int x;
      int y;
public:
      titik()                                  //konstruktor
default
      {
            x=0;
            y=0;
      }
      titik(int nx, int ny)    // konstruktor penyalinan
      {
            x=nx;
            y=ny;
      }
      titik(const titik& tt)   // konstruktor penyalinan objek
      {
            x=tt.x;
            y=tt.y;
      }
      int NX() { return x; } // fungsi anggota biasa
      int NY() { return y; } // fungsi anggota biasa
};
void main()
{
      titik t1;  // objek dg konstruktor default
      titik t2(10, 20); // objek dg konstruktor penyalinan
      titik t3(t2); // objek dg konstruktor penyalinan objek
      cout<<"t1 = "<< t1.NX() << ", "<<t1.NY()<<endl;
      cout<<"t2 = "<< t2.NX() << ", "<<t2.NY()<<endl;
      cout<<"t3 = "<< t3.NX() << ", "<<t3.NY()<<endl;
      getch();
}
```

Hasil keluaran program:

```
t1 = 0,0
t2 = 10,20
t3 = 10,20
```

Objek t1 diciptakan dengan otomatis menjalankan konstruktor default. Dengan demikian anggota data diberi nilai x=0, dan y = 0. Penciptaan objek t2 diikuti dengan menjalankan konstruktor kedua, mengakibatkan anggota data

diberi nilai x=10 dan y=20. Terakhir, t3 diciptakan dengan menyalin objek dari t2, mengakibatkan pemberian nilai kepada anggota data sama dengan objek t2.

Destruktor

Destruktor adalah pasangan konstruktor. Pada saat program menciptakan objek secara otomatis konstruktor akan dijalankan, yang biasanya dimaksudkan untuk memberi nilai awal variabel private. Sejalan dengan ini, C++ menyediakan fungsi destruktor (penghancur atau pelenyap) yang secara otomatis akan dijalankan pada saat berakhirnya kehidupan objek. Fungsi destruktor adalah untuk mendealokasikan memori dinamis yang diciptakan konstruktor. Nama destruktor sama dengan nama kelas ditambah awalan karakter tilde(~).

Perhatikan contoh berikut:

```
// Contoh Konstruktor dan Destruktor
#include<iostream.h>
class Tpersegi
{
int *lebar, *panjang;
public:
      Tpersegi (int, int);
      ~Tpersegi();
      int Luas() {return (*lebar * *panjang);}
};

Tpersegi::Tpersegi(int a, int b)
{
      lebar = new int;
      panjang = new int;
      *lebar = a;
      *panjang = b;
}
```

```
Tpersegi::~Tpersegi()
{
      delete lebar;
      delete panjang;
}
int main()
{
      Tpersegi pers(3,4), persg(5,6);
      cout<< “Luas pers = ”<<pers.Luas()<<endl;
      cout<< “Luas persg = ”<<persg.Luas()<<endl;
      return 0;
}
```

MODUL 10

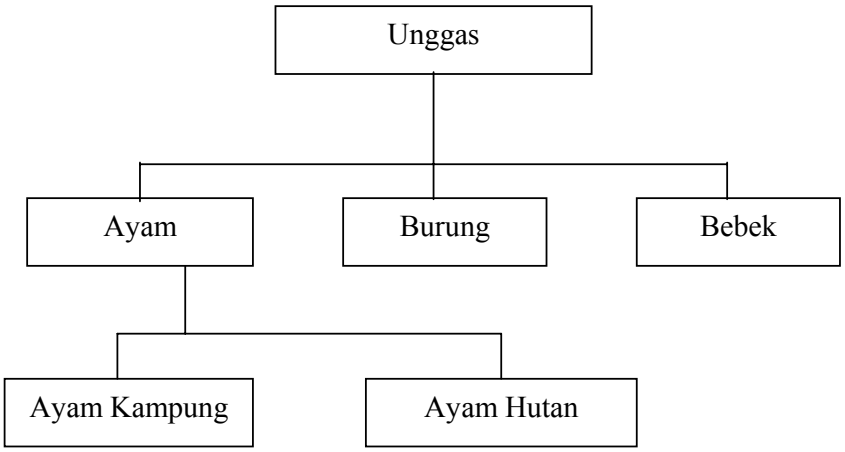
Waktu : 2 x pertemuan

Pewarisan (Inheritance)

Suatu kelas dapat diciptakan berdasarkan kelas lain. Kelas baru ini mempunyai sifat-sifat yang sama dengan kelas pembentuknya, ditambah sifat-sifat khusus lainnya. Dengan pewarisan kita dapat menciptakan kelas baru yang mempunyai sifat sama dengan kelas lain tanpa harus menulis ulang bagian-bagian yang sama. Pewarisan merupakan unsur penting dalam pemrograman berorientasi objek dan merupakan blok bangunan dasar pertama penggunaan kode ulang (code reuse).

Jika tidak ada fasilitas pewarisan ini, maka pemrograman dalam C++ akan tidak banyak berbeda dengan pemrograman C, hanya perbedaan dalam pengkapsulan saja yang menggunakan kelas sebagai pengganti struktur. Yang perlu menjadi catatan di sini adalah bahwa data dan fungsi yang dapat diwariskan hanya yang bersifat public dan protected. Untuk data dan fungsi private tetap tidak dapat diwariskan. Hal ini disebabkan sifat protected yang hanya dapat diakses dari dalam kelas saja.

Sifat pewarisan ini menyebabkan kelas-kelas dalam pemrograman berorientasi objek membentuk hirarki kelas mulai dari kelas dasar, kelas turunan pertama, kelas turunan kedua dan seterusnya. Sebagai gambaran misalnya ada hirarki kelas unggas.



Sebagai kelas dasar adalah Unggas. Salah satu sifat Unggas adalah bertelur dan bersayap. Kelas turunan pertama adalah Ayam, Burung dan Bebek. Tiga kelas turunan ini mewarisi sifat kelas dasar Unggas yaitu bertelur dan bersayap. Selain mewarisi sifat kelas dasar, masing-masing kelas turunan mempunyai sifat khusus, Ayam berkokok, Burung terbang dan Bebek berenang. Kelas Ayam punya kelas turunan yaitu Ayam Kampung dan Ayam Hutan. Dua kelas ini mewarisi sifat kelas Ayam yang berkokok. Tetapi dua kelas ini juga punya sifat yang berbeda, yaitu: Ayam Kampung berkokok panjang halus sedangkan Ayam hutan berkokok pendek dan kasar.

Jika hirarki kelas Unggas diimplementasikan dalam bentuk program, maka secara sederhana dapat ditulis sebagai berikut:

```
//Program Kelas Unggas
#include<iostream.h>
#include<conio.h>
class Unggas
{
public:
      void Bertelur() {cout<<"Bertelur"<<endl; }
};
class Ayam : public Unggas
{
public:
      void Berkokok() {cout<<"Berkokok"<<endl; }
};
class Burung : public Unggas
{
public:
      void Terbang() {cout<<"Terbang"<<endl; }
};
class Bebek : public Unggas
{
public:
      void Berenang() {cout<<"Berenang"<<endl; }
};
class AyamKampung : public Ayam
{
public:
      void Berkokok_Panjang_Halus() {cout<<"Berkokok Panjang Halus"<<endl; }
};
class AyamHutan : public Ayam
{
public:
      void Berkokok_Pendek_Kasar() {cout<<"Berkokok Pendek Kasar"<<endl; }
};

void main()
{
      cout <<"Sifat bebek adalah:"<<endl;
      Bebek bk;
      bk.Bertelur();
      bk.Berenang();
      cout<<endl;
      cout<<"Sifat ayam adalah:"<<endl;
      Ayam ay;
      ay.Bertelur();
      ay.Berkokok();
      cout<<endl;
      cout<<"Sifat ayam kampung adalah:"<<endl;
      AyamKampung ayk;
      ayk.Bertelur();
      ayk.Berkokok();
      ayk.Berkokok_Panjang_Halus();
      getch();
}
```

Dapat dilihat, bahwa kelas Ayam dan kelas Bebek dapat menjalankan fungsi Bertelur() yang ada dalam kelas Unggas meskipun fungsi ini bukan merupakan anggota kelas Ayam dan kelas Bebek.

Kelas AyamKampung dapat menjalankan fungsi Berkokok() yang ada dalam kelas Ayam walaupun dua fungsi tersebut bukan merupakan anggota kelas AyamKampung. Sifat-sifat di atas yang disebut dengan pewarisan (*inheritance*).

Polimorfisme

Polimorfisme merupakan fitur pemrograman berorientasi objek yang penting setelah pengkapsulan dan pewarisan. Polimorfisme berasal dari bahasa Yunani, ***poly***(banyak) dan ***morphos*** (bentuk). Polimorfisme menggambarkan kemampuan kode C++ berperilaku berbeda tergantung situasi pada waktu run (program berjalan).

Konstruksi ini memungkinkan untuk mengadakan **ikatan dinamis** (juga disebut ikatan tunda, atau ikatan akhir). Kalau fungsi-fungsi dari suatu kelas dasar didefinisikan ulang atau ditindih pada kelas turunan, maka objek-objek yang dihasilkan hirarki kelas berupa objek polimorfik. Polimorfik artinya mempunyai banyak bentuk atau punya kemampuan untuk mendefinisi banyak bentuk. Perhatikan kelas sederhana berikut :

```
#include <iostream.h>
class TX
{
public:
double FA(double p) { return p*p; }
double FB(double p) { return FA(p) / 2; }
};
class TY : public TX
{
public:
double FA(double p) { return p*p*p; }
};
void main()
{
TY y;
cout << y.FB(3) << endl;
}
```

Kelas TX berisi fungsi FA dan FB, dimana fungsi FB memanggil fungsi FA. Kelas TY, yang merupakan turunan dari kelas TX, mewarisi fungsi FB, tetapi mendefinisikan kembali fungsi FA. Yang perlu diperhatikan di sini adalah fungsi TX::FB yang memanggil fungsi TY::FA dalam konteks perilaku polimorfisme.

Berapakah output dari program ini? Jawabnya adalah 4.5 dan bukan 13.5! Mengapa?

Jawabannya adalah karena kompiler memecahkan ungkapan Y.FB(3) dengan menggunakan fungsi warisan TX::B, yang akan memanggil fungsi TX::FA.

Oleh karena itu fungsi TY::FA ditinggalkan dan tidak terjadi perilaku polimorfisme.

C++ menetapkan persoalan ini dan mendukung perilaku polimorfisme dengan menggunakan fungsi virtual. Fungsi virtual, yang mengikat kode pada saat runtime, dideklarasikan dengan menempatkan kata kunci virtual sebelum tipe hasil fungsi.

Untuk program di atas, kode double FA(double p) tulis menjadi virtual double FA(double p) baik di kelas TX maupun yang ada di kelas TY. Dengan perubahan ini maka output dari program di atas menjadi 13.5.