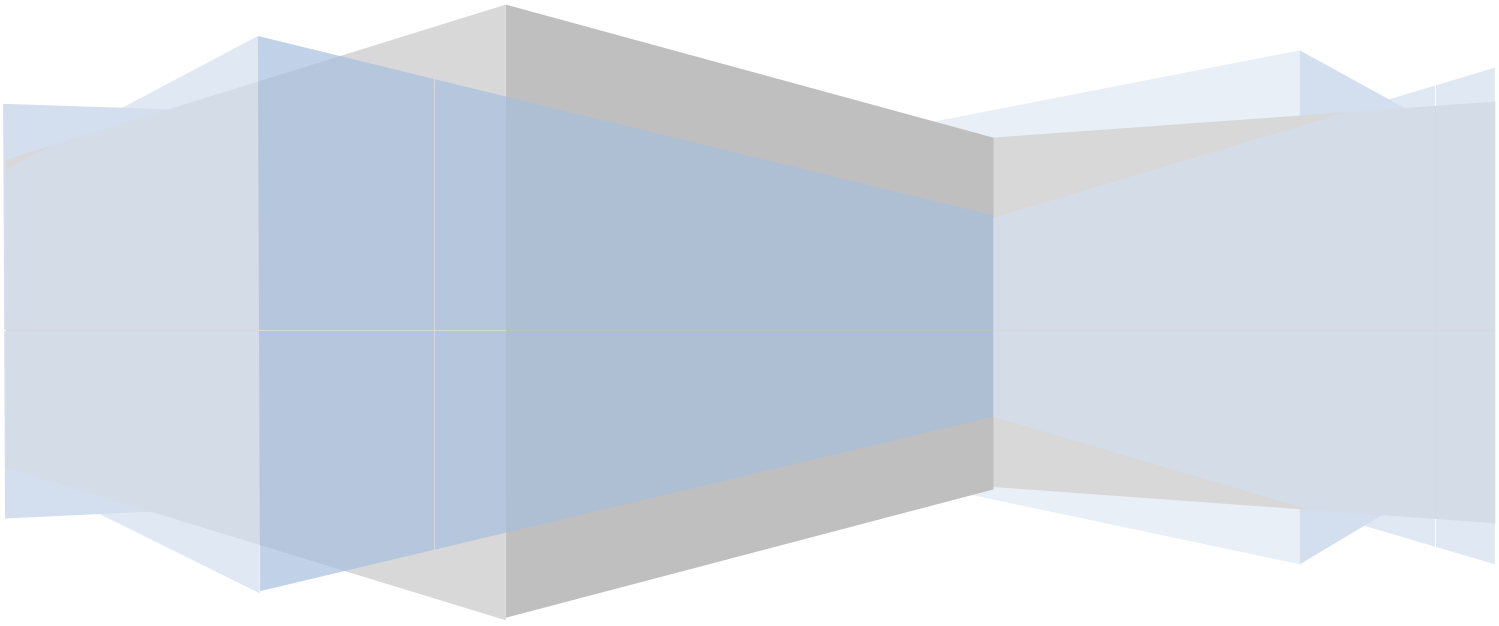
Universitas Komputer Indonesia

Modul Praktikum

Pemograman Berorientasi Objek dengan C++

S. Indriani Lestariningati, M.T



Daftar Isi

PENGENALAN C++ 1
INPUT & OUTPUT STANDAR 4
ELEMEN DASAR C++ 7
SELEKSI KONDISIONAL DAN PERULANGAN 14
FUNGSI 21
ARRAY 28
STRING 34
POINTER 38
STRUKTUR DAN KELAS 40
KONSEP BERORIENTASI OBJEK (1) 46
KONSTRUKTOR DAN DESTRUKTOR 50
KONSEP BERORIENTASI OBJEK (2) 54
KONSEP BERORIENTASI OBJEK (3) 57

MODUL 1

PENGENALAN C++

STRUKTUR BAHASA C++

Cara terbaik untuk belajar bahasa pemograman adalah dengan cara langsung mempraktekkannya.

Cobalah contoh program berikut:

```
// program pertamaku
#include<iostream.h>
int main()
{
      cout <<"selamat belajar c++";
}
```

Program diatas, misalnya dapat disimpan dengan nama *latih.cpp*.

Tekan tombol **Alt+F9** untuk mengkompile program (*compile program*), jika status dinyatakan sukses, dan tidak ada pesan kesalahan (error) maka tekan **Ctrl+F9** untuk menjalankan program tersebut (*run program*).

- `// program pertamaku`

  Merupakan sebuah baris komentar. Semua baris yang ditandai dengan dua buah tanda slash (//), akan diangkap 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 utnuk menyertakan pre-processor. Pernyataan ini bukan untuk dieksekusi. `#include<iostream.h>` berarti memerintahkan compiler untuk menyertakan file header `iostream.h`. Dalam file header ini, terdapat beberapa fungsi standar yang dipakai dalam proses input dan output. (yaitu perintah `cin` dan `cout`)

- `int main( )`

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

  Main diikuti dengan sebuah tanda kurung ( ), karena main merupakan sebuah fungsi. Dalam bahasa C++ sebuah fungsi harus diikuti dengan tanda ( ), yang nantinya dapat berisi sebuah argument.

- `{ }`

  Isi dari sebuah fungsi harus diawali dengan kurung kurawal buka ({) dan diakhiri dengan kurung kurawal tutup (})

- `cout<<"selamat belajar c++";`

  perintah ini merupakan hal yang akan dieksekusi oleh compiler dan merupakan perintah yang akan dikerjakan.

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

- `Return 0;`

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

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

```
// latihan keduaku
#include<iostream.h>
int main()
{
      cout <<"selamat belajar c++";
      cout<<"di Lab 7";
      return 0;
}
```

MODUL 2

INPUT & OUTPUT STANDAR

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 dari 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 variabel x ke layar
```

Operator << dikenal dengan *insertion operator* yang memberikan perintah kepada cout.

- Untuk contoh pertama, kalimat yang akan dicetak di layar di apit oleh tanda " " karena berupa string.
- Sedangkan pada contoh kedua dan ketiga, tanpa tanda " ", karena yang akan ditampilkan ke layar bukan berupa string ataupun karakter.

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

```
cout<<"hallo"<<"saya"<<"belajar c++;
```

Selanjutnya dapat juga dikombinasikan dengan variabel. Misalnya:

```
cout<<"hallo saya berusia"<<age<<"tahun";
```

Untuk menambahkan perintah ganti baris, ada dua perintah yang dapat dipakai: "\n" dan perintah **endl**.

Contoh:

- Penggunaan perintah “\n”

```
// program pertamaku
#include<iostream.h>
int main()
{
      cout <<"selamat belajar c++ \n";
      cout<<"di Lab 7";
      return 0;
}
```

- Penggunaan perintah **endl**

```
// program pertamaku
#include<iostream.h>
int main()
{
      cout <<"selamat belajar c++"<<endl;
      cout<<"di Lab 7";
      return 0;
}
```

2. Input (cin)

Untuk menerima inputan dengan perintah `cin`, maka operator yang akan digunakan adalah *overloaded operator* (>>) dan diikuti dengan variabel tempat menyimpan inputan data.

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.

Contoh program:

```
#include<iostream.h>
void main()
{
      int age;
      cout<<"masukkan umur anda = ";
      cin>>age;
      cout <<"umur saya sekarang= "<<age<<"tahun";
}
```

MODUL 3

ELEMEN DASAR C++

1. TIPE DATA

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

Tipe	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**.

Tipe	Keterangan
Bool	Isi bilangan Boolean (true dan false)
Wchar_t	Wide character

Contoh program mengetahui ukuran memory dari tipe data

```
#include <iostream.h>
void main()
{
      cout<<"ukuran char      : "<<sizeof(char)<<" byte\n";
      cout<<"ukuran short     : "<<sizeof(short)<<" byte\n";
      cout<<"ukuran long      : "<<sizeof(long)<<" byte\n";
      cout<<"ukuran float     : "<<sizeof(float)<<" byte\n";
}
```

2. VARIABEL

Berbeda dengan pendeklarasian variabel di bahasa pemograman 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 variabel 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;
```

contoh

```
#include<iostream.h>

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

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

      //cetak hasilnya:
      cout<<"hasil = "<<hasil;
}
```

3. KONSTANTA

Konstanta mirip dengan variabel, namun memiliki nilai tetap. Konstanta dapat berupa nilai interger, float, karakter, dan string.

Pendeklarasian konstanta dapat dilakukan dengan dua cara:

- Menggunakan (`#define`)

  Deklarasi konstanta dengan cara ini, lebih mudah dilakukan karena akan menyertakan `#define` sebagai preprocessor directive. Dan sintaksnya diletakkan bersama-sama dengan pernyataan `#include` (diatas main( ))

  Format penulisannya adalah:

  ```
  #define pengenal nilai
  ```

  Contoh penggunaan:

  ```
  #define phi 2.414159265
  #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 konstanta (const)

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

  Contoh:

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

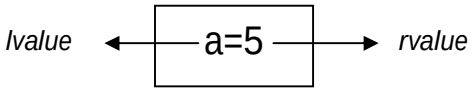
  Untuk contoh terakhir, deklarasi variabel zip yang tanpa tipe data, maka compiler akan secara otomatis memasukkannya kedalam tipe `int`

4. OPERATOR

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

- **Operator Assign (=)**

  Operator (=), akan memberikan nilai kedalam suatu variabel.



  Sebelah kiri tanda = dalam pernyataan diatas, dikenal dengan *lvalue* (left value) dan disebelah kanan tanda = dikenal dengan *rvalue* (right value).

  lvalue harus selalu berupa variabel, sedangkan *rvalue* dapat berupa variabel, nilai, konstanta, hasil operasi ataupun kombinasinya.

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

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

  Contoh:

  a=11%3, maka variabel a akan terisi nilai 2 karena sisa hasil bagi 11 dan 3 adalah 2.

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

  Operator penaikan/ *increment* (++) akan menaikkan atau menambahkan 1 nilai variabel. Sedangkan operator penurunan/ *decrement* (--) akan menurunkan atau mengurangi 1 nilai variabel.

  Misalnya:

```
a++;
a--;
```

  karakteristik dari operator ini adalah dapat dipakai diawal variabel (++a) atau diakhir variabel (a--). Hal ini akan berpengaruh pada hasilnya.

Contoh:

Pengaruh penempatan increment didepan:

```
#include <iostream.h>
void main()
{
      int r=2;
      int s;
      s=2 + ++r;
      cout<<s<<endl<<r;
}
```

Pengaruh penempatan increment dibelakang:

```
#include <iostream.h>
void main()
{
      int r=2;
      int s;
      s=2 + r++;
      cout<<s<<endl<<r;
}
```

- **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

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

void main()
{     int x = 2;
      clrscr();
      cout <<"x = "<< x <<endl;
      x+=3;
      cout <<"setelah x+=3, x menjadi " << x <<endl;
      x*=3;
      cout <<"setelah x*=3, x menjadi "<< x <<endl;
}
```

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

  Yang dihasilkan dari operator ini bukan berupa sebuah nilai, namun berupa bilangan bool yaitu benar dan 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

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

void main()
{     int nilai;

      clrscr();

      nilai = 3>2;      //hasil ungkapan: benar
      cout << "nilai =   " << nilai<<endl;

      nilai = 2>3;      //hasil ungkapan: salah
      cout << "nilai =   " <<nilai<<endl;
}
```

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

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

  ((5==5) && (3>6) hasilnya akan bernilai false, karena (true && false)

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

void main()
{     int x = 200;

      clrscr();

      cout<<"(x>=1)&&(x<=50) ->"<<((x>=1)&&(x<=50))<<endl;
      cout<<"(x>=1)||(x<=50) ->"<<((x>=1)||(x<=50))<<endl;
}
```

- **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

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

void main()
{
      int a,b, minim;

      clrscr();

      a = 53;
      b = 6;
      minim=a<b?a:b;

      cout<<"Bilangan terkecil="<<minim<<endl;
}
```

MODUL 4

SELEKSI KONDISIONAL DAN PERULANGAN

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 sintaksnya.

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

SELEKSI KONDISIONAL

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

Sintaksnya adalah:

```
if (kondisi)
      pernyataan1;
else
      pernyataan 2;
```

if

```
#include<iostream.h>
#include<conio.h>
void main()
{
      int usia;

      clrscr();

      cout<<“Berapa usia anda”;
      cin>>usia;
      if (usia<17)
            cout<<“Anda tidak diperkenankan menonton<<endl;
}
```

if_else

```
#include<iostream.h>
#include<conio.h>
void main()
{
      int usia;
      clrscr();
      cout<<"Berapa usia anda ";
      cin>>usia;
      if (usia<17)
            cout<<"Anda tidak diperkenankan menonton!"<<endl;
      else
            cout<<"Selamat menonton!"<<endl;
}
```

2. switch

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

Sintaksnya adalah:

```
switch (option)
{
case nilai 1:
      blok pernyataan 1
      break;
case nilai 2:
      blok pernyataan 2
      break;
-
-
default:
      blok pernyataan default
}
```

```
#include<iostream.h>
#include<conio.h>
void main()
{
      int usia;
      clrscr();
      cout<<"masukkan angka menurut keterangan dibawah ini:"<<endl;
      cout<<"-------------------------------------------"<<endl;
      cout<<" 1 = jika anda berusia dibawah 17 tahun"<<endl;
      cout<<" 2 = jika anda berusia diatas 17 tahun"<<endl;
      cout<<"-------------------------------------------"<<endl;
      cin>>usia;
      switch (usia)
      {
            case 1:
                  cout<<"Anda tidak diperkenankan menonton!";
                  break;
            case 2:
                  cout<<"Selamat menonton!";
                  break;
            default:
                  cout<<"tidak terdefinisi";
      }
}
```

PERULANGAN (loops)

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

1. **while**

   Sintaksnya adalah:

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

   Pernyataan akan dijalankan selama ekspresi bernilai true.

   Contoh:

```
#include<iostream.h>
#include<conio.h>
void main()
{
      int n;
      cout<<"Masukkan angka untuk mulai: ";
      cin>>n;
      while (n>0)
      {
            cout<<n<<" , ";
            --n;
      } cout<<"selesai";
}
```

Algoritma untuk pengulangan diatas adalah sebagai berikut:

1. User menginputkan sebuah nilai ke variabel n.
2. Pernyataan while akan melakukan pengecekan apakah (n=0)?
   Dalam kondisi ini, terdapat dua kemungkinan:
   a. True : lakukan pernyataan (langkah 3)
   b. False: lompati pernyataan (langkah 5)
3. Lakukan perintah:

```
cout<<n<<" , ";
--n;
```

4. Akhiri blok, kembali lagi ke langkah 2.
5. Lanjutkan program setelah blok while. Cetak SELESAI, dan akhiri program.

2. do…while

Sintaksnya:

```
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:

- menggunakan perintah while

```
#include<iostream.h>
#include<conio.h>
void main()
{
      int i;
      clrscr();
      i=0;
      while (i<10)
      {
            cout<<"C++"<<endl;
            i++;
      }
}
```

- menggunakan perintah do_while

```
#include<iostream.h>
#include<conio.h>
void main()
{
      int i;
      clrscr();
      i=0;
      do
      {
            cout<<"C++"<<endl;
            i++;
      }
      while (i<10);

}
```

3. for

Sintaksnya:

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

Pernyataan akan diulangi jika kondisi bernilai true (sama seperti while). Namun for menetapkan inisialisasi dan counter berada dalam ( ) tanda kurung .

Penjelasannya adalah sebagai berikut:

1. `inisialisasi;` akan dieksekusi. Biasanya merupakan variabel 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. `counter;` akan dieksekusi. Biasanya dapat terdiri dari sebuah instruksi atau blok instruksi yang berada diantara { dan }.

```
#include<iostream.h>
#include<conio.h>
void main()
{
      int i;
      clrscr();
      for(i=0;i<10;i++)
            {
                  cout<<"C++"<<endl;
            }
}
```

Nested Loops (Perulangan Bertumpuk)

Perulangan bertumpuk secara sederhanan dapat diartikan: terdapat satu atau lebih loop didalam 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 mendefiniskan baris, sedangkan loop dalam, digunakan untuk mendefinisikan kolom

Contoh:

```
#include<iostream.h>
#include<conio.h>
void main()
{
      clrscr();
      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

selesai

hasilnya dilayar akan muncul hasil dengan bentuk matriks 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 total.

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

  Dan dilayar akan tampak hasil sebagai berikut:

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

- Perintah **continue**

  Perintah ini akan melewati satu iterasi yang sesuai dengan syarat tertentu, dan melanjutkan ke iterasi selanjutnya.

  Contoh:

```
for (int n=10; n>0; n--)
{
      if (n==3) continue;
      cout<<n<<” , “;

}
cout<<penghitungan dihentikan!”;
```

  Dan dilayar akan tampak hasil sebagai berikut:

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

MODUL 5

FUNGSI

1. Pengertian Function (Fungsi)

Yang dimaksud dengan Function atau fungsi adalah sejumlah instruksi yang dikelompokkan menjadi satu, berdiri sendiri, yang berfungsi untuk menyelesaikan suatu pekerjaan tertentu. Bahasa C/C++ adalah suatu bahasa yang struktur penulisannya merupakan kumpulan dari fungsi-fungsi. Setiap fungsi mempunyai nama. Bahasa C/C++ minimal terdiri dari satu buah fungsi yang disebut fungsi main( ). Nama main adalah nama fungsi yang ditentukan oleh bahasa C/C++, yang tidak boleh diganti dengan nama lain. Fungsi-fungsi lain (yang dibuat sendiri bila diperlukan) namanya dapat dikarang sendiri.

Contoh:

Sebuah program yang didalamnya terdapat sebuah fungsi yang namanya HITUNG.

```
#include<iostream.h>

int HITUNG(int A, int B);
void main()
{
     int A, B, T;
     A=5; B=2; T=0;
     T=HITUNG(A,B);
     cout<<T;
}

int HITUNG(int A, int B)
{
     int T;
     A=A*2;
     B=B*2;
     T=A+B;
     return(T);
}
```

2. Mendeklarasikan dan mendefinisikan Fungsi

- Suatu fungsi mempunyai “Judul” yang minimal berisi Nama dan Tipe Fungsi tersebut. Menulis ‘Judul’ sebuah fungsi sebagai awal dari suatu fungsi disebut men-DEFINISIKAN fungsi.
- Bila sebuah fungsi ditulis ‘dibawah’ atau sesudah fungsi main( ), maka fungsi tersebut harus diperkenalkan atau ‘didaftarkan’ terlebih dahulu sebelum dapat digunakan. Menulis ‘pendaftaran’ fungsi ini disebut men-DEKLARASIKAN Fungsi, yang ditulis sebelum program induk main( ).

  Contoh: Program yang tidak menggunakan fungsi lain selain main function

```
#include<iostream.h>
void main()
{
     cout<<”Jakarta”;
}
```

  Contoh yang menggunakan fungsi:

```
#include<iostream.h>

void CETAK();
void main()
{
     CETAK();
}

void CETAK()
{
     cout<<"Jakarta";
}
```

- Sebuah fungsi memerlukan tipe sesuai dengan tipe nilai yang dikirimnya atau dikembalikan (*return*) ke bagian program atau fungsi yang memanggilnya. Bila sebuah fungsi tidak mengirimkan nilai balik, maka tipenya tidak diperlukan sehingga dapat dibuat sebagai **void**.
- Bila tipe tidak ditulis, bahasa C menganggap fungsi tersebut menggunakan tipe *default* yaitu tipe **int**.

- Format penulisan:

  Tipe nama(argumen1, argumen2, …) pernyataaan;

  Dimana:

  Tipe berisi tipe data yang akan dikembalikan oleh fungsi

  Nama merupakan pengenal utnuk memanggil fungsi

  Argumen (dapat didekalarasikan sesuai dengan kebutuhan). Tiap-tiap argumen terdiri dati tipe-tipe data yang diikuti oleh pengenalnya. Sama seperti mendeklarasikan variabel baru (contoh: int x)

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

- Fungsi yang tidak dideklarasikan terlebih dahulu, padahal Fungsi tersebut ditempatkan atau diDEFINISIKAN dibawah atau sesudah Fungsi main( ), maka fungsi tersebut tidak akan dikenal sehingga akan menyebabkan error.

```
#include<iostream.h>
void main()
{
     CETAK();
}
void CETAK()
{
     cout<<"Jakarta";
}
```

  Contoh jika fungsi CETAK( ) ditulis diatas atau sebelum fungsi main( ), maka tidak diperlukan lagi menDEKLARASIKAN Fungsi detak

```
#include<iostream.h>
void CETAK()
{
     cout<<"Jakarta";
}
void main()
{
     CETAK();
}
```

3. **Semua variabel yang dibuat dalam suatu Function, akan bersifat variabel LOKAL.**

```
#include<iostream.h>

void CETAK();
void main()
{
     CETAK();
}

void CETAK()
{
     int A, B, T;
     A=5; B=2;
     T=A+B;
     cout<<T;
}
```

4. **Variabel LOKAL dan variabel GLOBAL, CALLING Function, dan CALLED Function.**

```
#include<iostream.h>

void CETAK();
void main()
{
     int A, B, T;
     A=5; B=2;
     T=A+B;
     CETAK();
}
void CETAK()
{
     cout<<T;
}
```

Semua variabel yang ada dalam fungsi utama main(), juga bersifat LOKAL, hanya dikenal dalam fungsi main( ) saja, tidak dikenal didalam fungsi lain.

Error akan terjadi karena fungsi CETAK tidak mengenal variabel T, karena dalam fungsi CETAK tidak mengenal variabel T, karena dalam fungsi CETAK tidak ada perintah menyiapkan variabel T, seperti int T;

Contoh:

```
#include<iostream.h>

void CETAK();
int T;
void main()
{
     int A, B, T;
     A=5; B=2;
     T=A+B;
     CETAK();
}
void CETAK()
{
     cout<<T;
}
```

Dari contoh program diatas, main program atau main function disebut CALLING Function, yaitu program yang meng-CALL, dan Fungsi CETAK diebut CALLED Function karena dia merupakan fungsi yang di CALL.

Contoh:

```
#include<iostream.h>

void TAMBAH();
int A,B,T;
void main()
{
     A=5; B=2;
     TAMBAH();
     cout<<T;
}
void TAMBAH()
{
     T=A+B;
}
```

Dalam fungsi main () dan juga dalam fungsi TAMBAH( ), tidak disiapkan variabel, sehingga semua variabel diambil dari variabel GLOBAL.

Contoh:

```
#include<iostream.h>
void TAMBAH();
int A,B,T;
void main()
{
     int A, B;
     A=5; B=2;
     TAMBAH();
     cout<<T;
}
void TAMBAH()
{
     T=A+B;
}
```

Dalam fungsi main() ada disiapkan variabel A dan B, sehingga yang diisi dengan 5 dan 2 adalah A dan B yang ada dalam fungsi main( ), variabel lokal yang dimilikinya sendiri.

Fungsi TAMBAH( ) tidak menyiapkan variabel A, B, dan T, sehingga variabel A, B, dan T diambil dari variabel GLOBAL. Dalam kelompok variabel GLOBAL memang ada variabel A, B, dan T, sehingga proses dalam fungsi TAMBAH( ) tidak error, hanya isi variabel A, B, dan T yang ada dalam kelompok variabel GLOBAL dan nilainya tidak diketahui (berisi nilai interger sembarang)

Perintah mencetak T dalam fungsi main( ) adalah mencetak variabel T yang ada dalam kelompok variabel GLOBAL, yang isinya sembarang. Sehingga program diatas akan mencetak nilai interger sembarang.

5. Passing variabel (passing by Value)

Variabel Parameter dan variabel Argument

Calling function dapat mengirimkan (passing) suatu nilai ke Called Function. Bila nilai yang dikirim adalah nilai atau data yang akan diproses., maka pengiriman nilai tersebut disebut **passing by value**. Bila nilai yang dikirim berupa nilai pointer (bukan data) yang merefer ke suatu data, maka pengiriman tersebut disebut **passing by reference**.

Contoh-contoh berikut mengenai passing by value:

```
#include<iostream.h>
void CETAK(int T);
void main()
{
     int A, B, T;
     A=5; B=2;
     T=A+B;
     CETAK(T);
}
void CETAK(int T)
{
     cout<<T;
}
```

Pengalokasian variabel dalam memori untuk program diatas, dapat diilustrasikan sebagai berikut:

Nama variabel **argument** boleh berbeda dengan nama **variabel parameter**.

```
#include<iostream.h>
void CETAK(int T);
void main()
{
     int A, B, T;
     A=5; B=2;
     T=A+B;
     CETAK(T);
}
void CETAK(int X)
{
     cout<<X;
}
```

Semua nilai yang akan dioperasikan dibawa kedalam fungsi CETAK.

```
#include<iostream.h>
void CETAK(int X, int Y);
void main()
{
     int A, B;
     A=5; B=2;
     CETAK(A,B);
}
void CETAK(int X, int Y)
{
     int T;
     T=X+Y;
     cout<<T;
}
```

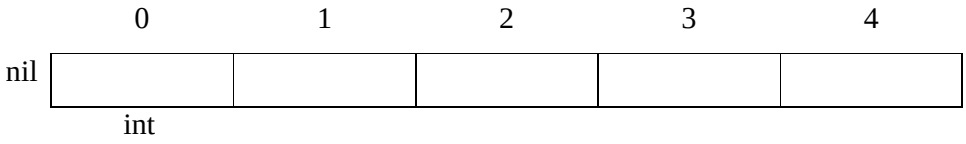
MODUL 6

ARRAY

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

Array merupakan koleksi data dimana setiap elemen memakai nama yang sama dan bertipe sama, 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 indeks pertama selalu dimulai dengan 0

Deklarasi array

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

Sintaksnya adalah:

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

Contoh, untuk pendeklarasian array dengan nama **nil** diatas adalah:

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

Inisialisasi Array

Nilai suatu variabel array dapat juga diinisialisasi secara langsng pada saat deklarasi, misalnya: `int nil [5] = { 1, 3, 6, 12, 24 };`

Maka di penyimpanan ke dalam arraynya akan 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 sintaks:

```
nama [indeks]
```

Pada contoh diatas, variabel nilai 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	**1**	**3**	**6**	**12**	**24**

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:

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

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

      for (n=0;n<4;n++)
      {
            hasil +=nil[n];
      }
      cout<<hasil;
      return 0;
}
```

Array dua dimensi

Struktur array yang dibahas diatas, 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 matriks B berukuran 2x3 dapat dideklarasikan sebagai berikut:

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

	0	1	2
0	2	4	1
1	3	5	7

Contoh program dengan dua dimensi

```
#include<iostream.h>
void main()
{
      int i,j;
      int matriks[2][3]={{2,4,1},{5,3,7}};

      for (i=0;i<2;i++)
      {
            for (j=0;j<3;j++)
            {
                  cout<<matriks[i][j];
            }
      cout<<endl;
      }
}
```

Beberapa Operasi dengan Array

- Memperoleh bilangan terbesar
- Mencari suatu data pada array
- Mengurutkan data

❖ **Memperoleh Bilangan terbesar**

- Pada program berikut, mula-mula array diisi dengan bilangan acak. Kemudian program menampilkan isi array dan sekaligus memperoleh bilangan yang terbesar. Setelah menampilkan isi seluruh array, nilai terbesar ditampilkan

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
#include<time.h>
#include<math.h>

void main()
{
      clrscr();

      const MAKS = 10;
      int data[MAKS];
      int maks;

      //memperoleh data secara acak
      randomize();
      for(int i=0; i<MAKS; i++)
      data[i]=rand();

      //menampilan data dan mencari
      //data terbesar

      cout<<"DATA : "<<endl;
      cout<<data[0]<<endl;
      maks = data[0]; // isi dengan data pertama

      for(i=1 ; i<MAKS ; i++)
      {
            cout<<data[i]<<endl;

            if (data[i]>maks)
            {
                  maks = data[i];
            }
      }
      cout<<"Data terbesar="<<maks<<endl;
}
```

❖ **Mencari Suatu Data**

Program berikut memberikan gambaran cara untuk mencari suatu data didalam array. Mula-mula data yang akan dicari perlu dimasukkan dari keyboard. Kemudian, data ini dibandingkan dengan elemen-elemen array. Jika ada yang sama, program melaporkan posisi elemen array yang cocok dengan data yang dicari.

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

void main()
{
      int i, x, ketemu;

      clrscr();

      int data[]={5,100,20,31,77,88,99,20,55,1};

      cout<<"Data yang anda cari: ";
      cin>>x;

      ketemu = 0;
      for (i=0; i<sizeof(data)/sizeof(int); i++)
      {
            if (data[i] == x)
            {
                  ketemu=!ketemu;   //ubah menjadi benar
                  break;            //keluar dari for
            }
      }
      if (ketemu)
            cout<<"Data tersebut ada pada posisi ke-";
            cout<<i+1<<endl;
            else
            cout<<"Data tersebut tidak ada!"<<endl;
}
```

❖ **Mengurutkan data**

- Ada berbagai teknik untuk mengurutkan data, salah satu diantaranya adalah metoda *bubble sort*. (metoda ini terkenal karena kesederhanaannya).
- Pengurutan dilakukan dengan membandingkan setiap elemen dengan seluruh elemen yang terletak sesudah posisinya.

```
#include<iostream.h>
#include<conio.h>
#include<iomanip.h>
void main()
{
      int i, j, tmp, jumdata;
      clrscr();
      int data[]={5,100,20,31,77,88,99,20,55,1};
      jumdata=sizeof(data)/sizeof(int);

      //menampilkan data
      cout<<"data semula:"<<endl;
      for (i=0;i<jumdata;i++)
            cout<<setw(4)<<data[i];
            cout<<endl;

      //mengurutkan data
      for (i=0;i<jumdata-1;i++)
            for(j=i+1;j<jumdata;j++)
            if (data[i]>data[j])
            {
                  tmp=data[i];
                  data[i]=data[j];
                  data[j]=tmp;
            }

      //menampilkan data
      cout<<"data setelah diurutkan:"<<endl;
      for (i=0;i<jumdata;i++)
            cout<<setw(4)<<data[i];
      cout<<endl;
}
```

MODUL 7

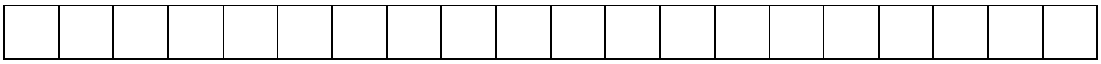
STRING

String

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

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

Maka dari pernyataan diatas, dapat digambarkan sebagai deklarasi sebuah variabel string (array dari karakter) dengan panjang maksimum 20 karakter, termasuk diakhiri dengan karakter null.



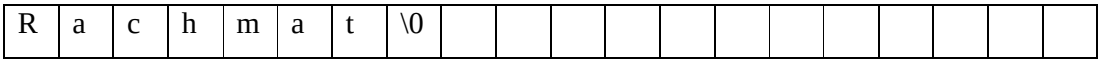
namaku

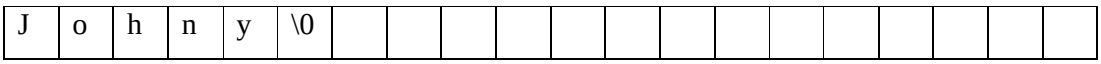
Ukuran maksimum 20 karakter untuk pernyataan diatas, dalam pengisiannya tidak harus penuh. Contoh, variabel namaku, dapat diisi dengan string “Rachmat” yang panjangnya 7 karakter atau dapat digantikan dengan string “Johny” yang memiliki panjang 5 karakter.

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 kedalam variabel string sedangkan tanda (“) berarti menginputkan sebuah nilai string kedalam variabel string.

Untuk memberikan nilai kepada sebuah variabel string, biasanya digunakan perintah input stream (`cin`) dan diikuti dengan `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;
}
```

Fungsi-fungsi untuk memanipulasi string

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

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

dengan cara seperti diatas, maka isi dari string2 akan dikopikan ke dalam string 1.

```
//penggunaan strcpy
#include<iostream.h>
#include<string.h>

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

Hasil outputnya adalah:

```
Ayu
```

Contoh penggunaan perintah manipulasi string yang lain:

Perintah	Kegunaan
Strcpy	Menyalin isi string
Strln	Mengetahui panjang string
Strcat	Menggabungkan string
Strcmp	Membandingkan dua buah string

strln

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

void main()
{
      char bunga[25]="mawar";
      char kosong[15]="";
      clrscr();
      cout<<strlen(bunga)<<endl;
      cout<<strlen(kosong)<<endl;
}
```

strcat

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

void main()
{
      char st[25]="satu dua";
      clrscr();
      cout<<"Isi st semula    : "<<st<<endl;
      strcat(st," tiga empat lima");
      cout<<"Isi st semula    : "<<st<<endl;
}
```

strcmp

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
void main()
{
      char st[128];
      char cpp[]="C++";
      clrscr();
      cout<<"Masukkan sembarang string "<<endl;
      cin.getline(st, sizeof(st));
      int hasil=strcmp(st, cpp);
      if (hasil==0)
            cout<<st<<"=="<<cpp<<endl;
      else if (hasil<0)
            cout<<st<<"<"<<cpp<<endl;
      else
      cout<<st<<">"<<cpp<<endl;
}
```

MODUL 8

POINTER

Setiap kali computer 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: bil 1 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

```
#include<iostream.h>
#include<conio.h>
void main()
{
      int a=5;
      float b=7.5;
      double c=17.777;

      clrscr();

      cout<<"isi variabel:"<<endl;

      cout<<"a = "<<a<<endl;
      cout<<"b = "<<b<<endl;
      cout<<"c = "<<c<<endl;

      cout<<"Alamat variabel : "<<endl;

      cout<<"a = "<<&a<<endl;
      cout<<"b = "<<&b<<endl;
      cout<<"c = "<<&c<<endl;
}
```

Contoh lain:

```
//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;
}
```

MODUL 9

STRUKTUR DAN KELAS

Struktur

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 berguna.

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

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

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

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

Untuk memberi nilai panjang dan lebar kotak, salah satu caranya adalah seperti diatas. Cara lain untuk member nilai panjang dan lebar adalah dengan membentuk suatu fungsi. Karena fungsi ini hanya digunakan untuk member nilai data panjang dan lebar suatu kotak, tentunya fungsi ini khusus milik objek kotak, sehingga harus dianggap sebagai anggota struktur kotak. C++ sebagai bahasa pemograman 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 member nilai panjang dan lebar hanya dengan memanggil 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<conio.h>
#include<iostream.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;
}
```

Kelas

Bentuk program diatas adalah contoh gaya pemograman berorientasi prosedur (terstruktur) yang sudah mengubah pola pikirnya menjadi berorientasi objek. Dalam pemograman 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 class. Jika objek kota 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 diakes dari luar 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 member 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.

Dapat dilihat dari sebuah 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;
```

Sebagai contoh:

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

MODUL 10

KONSEP BERORIENTASI OBJEK (1)

Pengantar

Pemograman 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. Polimophisme (*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.

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 didalam kelas dikendalikan. Pengendalian ini tidak hanya berupa data tapi 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*)

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 public. 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.

```
//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 diatas dengan nama PersegiPJ.cpp

Kemudian lanjutkan pembuatan program berikut pada halaman baru:

```
//penggunaan kelas persegi panjang
#include "PersegiPJ.cpp"
#include<conio.h>
void main()
{
      PersegiPanjang pp1(10,5);
      pp1.Skala(2);
      pp1.TulisInfo();
      getch();
}
```

MODUL 11

KONSTRUKTOR DAN DESTRUKTOR

Konstruktor

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 didalam kelas, tetapi jika diperlukan konstruktor dapat lebih dari satu.

Tiga jenis konstruktor:

1. *Konstruktor default*: tidak dapat menerima argument, 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:

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

class Kompleks
{
      private:
            double re;
            double im;

      public:
            Kompleks();
            void info();
};
void main()
{
      clrscr();

      Kompleks a;
      a.info();

      Kompleks b;
      b.info();
}

//Definisi fungsi anggota kelas Kompleks;
Kompleks::Kompleks()
{
      cout<<"Konstruktor dijalankan..."<<endl;
      re=5.2;
      im=3.6;
}

void Kompleks::info()
{
      cout<<"\nBilangan kompleks: "<<endl;
      cout<<"Real    = "<<re<<endl;
      cout<<"Imajiner   = "<<im<<endl;
}
```



Destruktor

Destruktor adalah fungsi anggota kelas yang akan dijalankan secara otomatis pada saat objek akan sirna, nama destruktor sama seperti nama konstruktor, hanya saja diawali dengan sebuah karakter tak berhingga (~)

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

class Bilangan
{
      private:
            int i;

      public:
            Bilangan();
            ~Bilangan();
            void info_data();
};
void main()
{
      clrscr();
      Bilangan x;
      x.info_data();

      Bilangan y;
      y.info_data();

      cout<<"Fungsi main() berakhir..."<<endl;
}

Bilangan::Bilangan()
{
      cout<<"Konstruktor dijalankan..."<<endl;
      i=55;
}

Bilangan::~Bilangan()
{
      cout<<"Destruktor dijalankan..."<<endl;
}

void Bilangan::info_data()
{
      cout<<"i= "<<i<<endl<<endl;
}
```

Tampak bahwa destruktor dari masing-masing objek dengan sendirinya dijalankan saat fungsi main( ) berakhir;

Konstruktor, Destruktor & Pengalokasian Dinamis

Konstruktor dan destruktor sangat bermanfaat untuk melakukan pengalokasian memori secara dinamis dan pendealokasiannya (membebaskan memori). Pada saat objek diciptakan, kosntruktor mengalokasian memori di *heap* dan saat objek akan sirna, destructor melakukan pembebasan memori yang ada pada *heap*.

Contoh program untuk menggambarkan penggunaan konstruktor dan destruktor yang berkaitan dengan memori dinamis dapat dilihat di bawah ini.

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
class Mobil
{
      private:
            char *nama;
            int cc_mesin;
            int jumlah_pintu;

      public:
            Mobil (char *nama_mobil, int cc, int jum_pintu);
            ~Mobil();

            void keterangan();
};
void main()
{     clrscr();
      Mobil sedan("Ford", 1300, 4);
      Mobil APV("Daihatsu", 1000, 4);

      sedan.keterangan();
      APV.keterangan();
}
Mobil::Mobil(char *nama_mobil, int cc, int jum_pintu)
{     nama = new char[25];
      strcpy(nama, nama_mobil);
      cc_mesin=cc;
      jumlah_pintu=jum_pintu;
}
Mobil::~Mobil()
{     delete[]nama;
}
void Mobil::keterangan()
{     cout<<"Informasi mobil:"<<endl;
      cout<<"Nama       :"<<nama<<endl;
      cout<<"CC Mesin   :"<<cc_mesin<<endl;
      cout<<"Jumlah Pintu     :"<<jumlah_pintu<<endl;
      cout<<endl;
}
```

MODUL 12

KONSEP BERORIENTASI OBJEK (2)

PEWARISAN

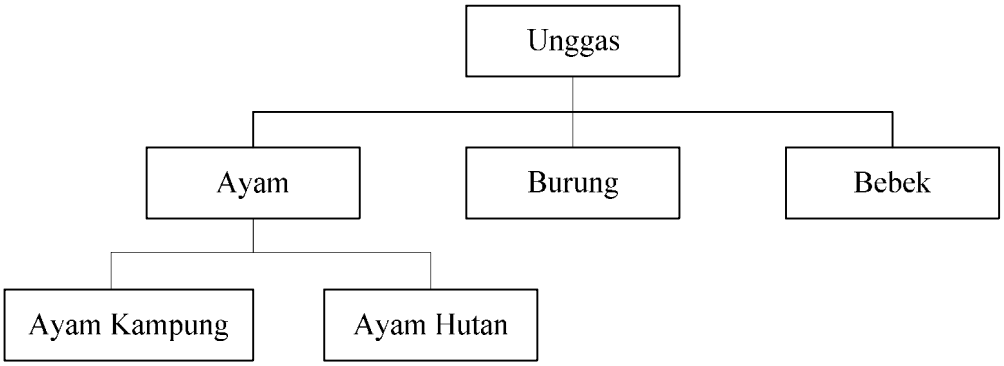
Pewarisan (Inheritance)

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 yang sama dengan kelas lain tanpa harus menulis ulang bagian-bagian yang sama. Perwarisan merupakan unsur penting dalam pemograman berorientasi objek dan merupakan blok bangunan dasar pertama penggunaan kode ulang (*code reuse*)

Yang perlu menjadi catatan disini 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 pemograman 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, Bebek. Tiga kelas turunan ini mewarisi sifat kelas dasar Unggas yaitu bertelur dan bersayap. Selain mewarisi sifat kelas dasar, masing-masing kelas turunannya mempunyai kelas khusus, Ayam berkokok, Burung terbang, dan Bebek berenang. Kelas ayam punya kelas turunan yaitu

Ayam kampung dan Ayam hutan. Dua kelas ini mewarisi sifat Ayam yang berkokok. Tapi 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 dan Halus"<<endl; }
};
class AyamHutan:public Ayam
{
      public:
      void Berkokok_Pendek_Kasar()
            {     cout<<"Berkokok Pendek dan Kasar"<<endl; }
};
void main()
{
      cout<<"Sifat Bebek adalah: "<<endl;
      Bebek bk;
      bk.Bertelur();
      bk.Berenang();
      cout<<"Sifat Ayam adalah: "<<endl;
      Ayam ay;
      ay.Bertelur();
      ay.Berkokok();
      cout<<"Sifat Ayam Kampung adalah"<<endl;
      AyamKampung ayk;
      ayk.Bertelur();
      ayk.Berkokok();
      ayk.Berkokok_Panjang_Halus();
      getch();
}
```

MODUL 13

KONSEP BERORIENTASI OBJEK (3)

POLIMORPHISME

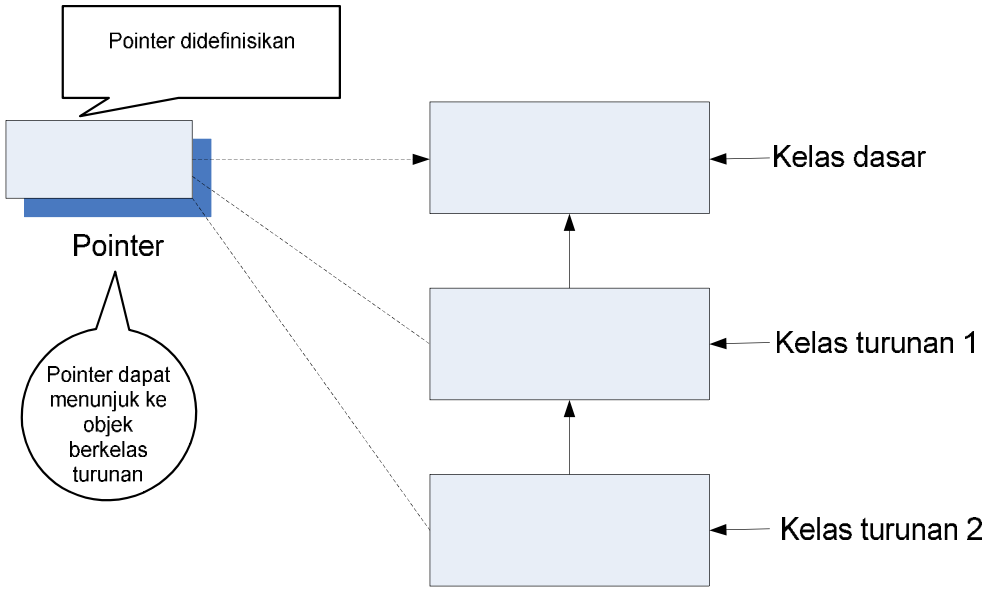
Polimorfisme

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

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

Fungsi virtual

Suatu fungsi anggota dari suatu kelas dapat dijadikan sebagai fungsi virtual. Jika fungsi seperti ini dideklarasikan kembali pada kelas turunan dan suatu pointer yang menunjuk kelas dasar diciptakan, pointer dapat memilih objek yang tepat sekiranya fungsi anggota tersebut dipanggil via pointer. Gambar dibawah memberikan ilustrasi suatu pointer seperti itu.



```
//contoh untuk memperlihatkan efek fungsi virtual
#include<iostream.h>
#include<conio.h>

class Makhluk
{
      public:
            void informasi()
            {     cout<<"informasi() pada Makhluk..."<<endl;
            }

            virtual void keterangan()
            {     cout<<"keterangan() pada Makhluk..."<<endl;
            }
};
class Mamalia : public Makhluk
{
      public:
            void informasi()
            {     cout<<"informasi () pada Mamalia..."<<endl;
            }

            void keterangan()
            {     cout<<"keterangan() pada Mamalia..."<<endl;
            }
};
class Sapi : public Mamalia
{
      public:
      void informasi()
            {     cout<<"informasi () pada sapi..."<<endl;
            }

            void keterangan()
            {     cout<<"keterangan() pada sapi..."<<endl;
            }
};
void main()
{
      clrscr();
      Mamalia mamalia;
      Sapi sapi_sumba;
      Makhluk *binatang;

      binatang = &mamalia;
      binatang->informasi();
      binatang->keterangan();

      cout<<"-----------------"<<endl;

      binatang = &sapi_sumba;
      binatang->informasi();
      binatang->keterangan();
}
```

```
//*contoh program Polimorphisme
#include<iostream.h>
#include<conio.h>

class Mahluk
{
      public:
      void informasi()
      {     cout<<"Informasi pada Mahluk..."<<endl;
      }
      virtual void keterangan()
      {     cout<<"Keterangan pada Mahluk..."<<endl;
      }
};
class mamalia:public Mahluk
{
      public:
      void informasi()
      {     cout<<"Informasi pada mahluk..."<<endl;
      }
      void keterangan()
      {     cout<<"Keterangan pada mamalia.."<<endl;
      }
};
class sapi:public mamalia
{
      public:
      void informasi()
      {     cout<<"informasi pada sapi..."<<endl;
      }
      void keterangan()
      {     cout<<"keterangan pada sapi..."<<endl;
      }
};
void main()
{
      clrscr();
      mamalia mamalia;
      sapi sapi_sumba;
      Mahluk*binatang;

      binatang=&mamalia;
      binatang->informasi();
      binatang->keterangan();

      cout<<"--------------"<<endl;

      binatang=&sapi_sumba;
      binatang->informasi();
      binatang->keterangan();

      getch();
}
```