PEMROGRAMAN BERORIENTASI OBJEK

Fungsi

Telkom University

Budhi Irawan, S.Si, M.T

PENDAHULUAN

- ***Fungsi*** merupakan ***kumpulan statemen*** yang ***dikelompokan*** menjadi satu ***bagian kode (blok program)*** untuk menyelesaikan ***tugas spesifik*** tertentu.
- Melalui cara seperti itu, ***kode (fungsi)*** hanya ***didefinisikan sekali***, namun dapat ***digunakan berulang kali*** tanpa harus menuliskan kembali kode yang sama.

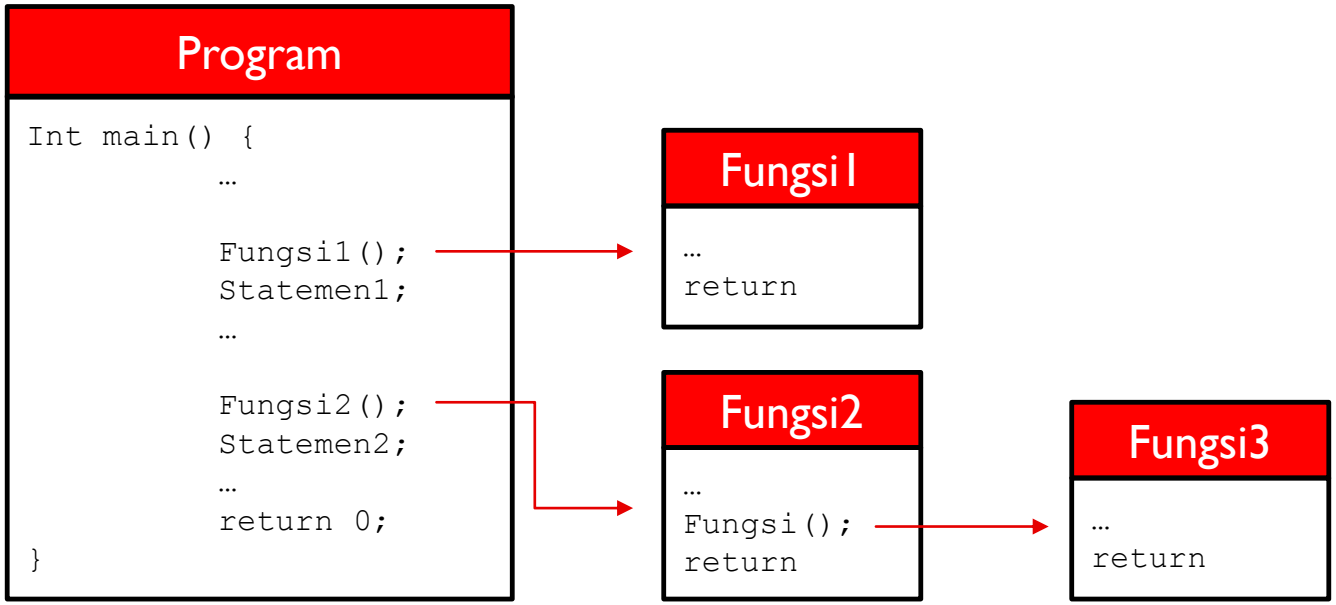
FUNGSI

- Dengan kata lain ***Fungsi*** merupakan ***subprogram*** dan berguna untuk menjadikan program untuk dapat lebih ***bersifat modular*** sehingga akan ***mudah difahami*** dan dapat ***digunakan kembali***, baik untuk program itu sendiri maupun untuk program lain yang ***memiliki proses yang sama***.

FUNGSI MAIN ()

- Dalam setiap bahasa pemrograman (termasuk C++), fungsi dibedakan menjadi 2 yaitu "***user-defined function***" dan "***built-in function***".
- ***User-define funtion*** adalah *fungsi-fungsi yang didefinisikan sendiri,* sedangkan ***built–in function*** *adalah fungsi-fungsi* ***'siap pakai'*** *yang telah disediakan oleh* ***compiler***.

FUNGSI DALAM PROGRAM



MENDEFINISIKAN FUNGSI

- Secara umum bentuk pendefinisian fungsi dalam C++ adalah sebagai berikut :

```
tipe_kembalian nama_fungsi(daftar_parameter) {
// badan fungsi
}
```

- Keterangan :
  - **`Tipe_kembalian`** : Fungsi bisa saja mempunyai nilai balik (nilai kembalian). Tipe Kembalian adalah tipe data dari nilai yang akan dikembalikan oleh fungsi
  - **`Nama_fungsi`** : Nama fungsi adalah nama yang akan digunakan dalam pemanggilan. Pemanggilan fungsi harus disesuaikan dengan daftar parameter yang disertakan pada saat pendefinisian fungsi.

MENDEFINISIKAN FUNGSI

- Keterangan :
  - **`Daftar parameter`** : Fungsi bisa saja memiliki satu atau lebih parameter. Parameter itu sendiri merupakan variabel yang berisi nilai yang akan disertakan kedalam proses yang dilakukan fungsi.
  - **`Badan_fungsi`** : kode atau statemen-statemen yang akan dilaksanakan didalam fungsi guna menyelesaikan tugas spesifik tertentu.

FUNGSI TANPA NILAI BALIK

- C++ tidak mengenal istilah ***prosedur (procedure)*** seperti pada saat melakukan pemrograman dengan menggunakan bahasa pascal.
- Dalam bahasa pascal, prosedur didefinisikan sebagai suatu proses yang tidak mengembalikan nilai.
- Untuk melakukan hal ini dalam C++, maka harus dibuat suatu fungsi dengan tipe **`void`**, yang berarti ***tidak memiliki nilai balik (return value)***

FUNGSI TANPA NILAI BALIK

- Bentuk umum dari ***pembuatan*** *fungsi tanpa nilai balik* ini sebagai berikut :

```
void nama_fungsi (parameter1, parameter2, …){
Statemen_yang_akan_dilakukan;
    …
    }
```

- Bentuk umum untuk ***pemanggilan*** *fungsi tanpa nilai balik* (yang sebelumnya telah didefinisikan) adalah sebagai berikut :

```
nama_fungsi(nilai_parameter1, nilai_parameter2, …);
```

FUNGSI TANPA NILAI BALIK

```
#include <iostream>

using namespace std;

// Membuat fungsi dengan nama Tulis10Kali()
void Tulis10Kali() {
  for (int C=0; C<10; C++) {
    cout<<"Saya Sedang Belajar PBO Menggunakan C++"<<endl;
  }
}

// Fungsi utama dalam program C++
int main() {

  // Memanggil fungsi Tulis10Kali() untuk dieksekusi
  Tulis10Kali();

  return 0;
}
```

FUNGSI DENGAN NILAI BALIK

- Berbeda dengan fungsi yang bertipe **`void`**, fungsi ini berguna untuk melakukan suatu proses yang dapat ***mengembalikan sebuah nilai***.
- Dalam membuat fungsi ini maka harus didefinisikan tipe data dari nilai yang akan dikembalikan :

```
tipe_data nama_fungsi (parameter1, parameter2, …){
Statemen_yang_akan_dilakukan;
…
return nilai_yang_akan_dikembalikan;
```

FUNGSI DENGAN NILAI BALIK

- Karena fungsi dengan nilai balik ini sebenarnya adalah sebuah ***nilai*** atau ***ekspresi***, maka pemanggilannya dapat dilakukan layakna seperti sebuah variabel
- Sebagai contoh, nilai yang dikembalikan dapat ditampung kedalam suatu variabel tertentu yang memiliki tipe data sama dengan tipe kembalian fungsi bersangkutan :

```
nama_variabel = nama__fungsi (parameter1, parameter2,...);
```

FUNGSI DENGAN NILAI BALIK

- Selain itu, pemanggilan fungsi juga bisa saja dilakukan langsung melalui perintah **`cout`** , jika keperluannya adalah ingin menampilkan nilai yang dihasilkan fungsi

```
cout<<nama_fungsi(parameter1, parameter2, …);
```

FUNGSI MENGEMBALIKAN TIPE STRING

- Berikut ini dibuat fungsi sederhana yang mengembalikan nilai bertipe ***string***.
- Dalam contoh ini, tidak akan digunakan parameter, maka dari itu setiap pemanggilan fungsi akan menghasilkan nilai yang sama.

FUNGSI MENGEMBALIKAN TIPE STRING

```
#include <iostream>

using namespace std;

// Membuat fungsi sederhana yang mengembalikan tipe string
char* FungsiString() {
  return (char*) "Ini adalah nilai dari fungsi";
}

// Fungsi utama
int main() {
  // Memanggil dan menampilkan hasil fungsi
  cout<<FungsiString();

  return 0;
}
```

FUNGSI MENGEMBALIKAN TIPE BILANGAN

- Selain mengembalikan tipe ***string***, fungsi dari C++ juga dapat mengembalikan nilai berupa ***karakter, numerik (bilangan), boolean, pointer*** maupun yang lainnya.
- Pada bagian ini akan kembali dibuat program yang memiliki definisi fungsi dengan nilai balik.
- Kali ini, fungsi yang didefinisikan akan mengembalikan nilai berupa ***bilangan.***

FUNGSI MENGEMBALIKAN TIPE BILANGAN

```
#include <iostream>

using namespace std;

// Membuat fungsi dengan nilai kembalian bertipe double
double FungsiBilangan() {
  return (3.14 * 3);
}

// Fungsi utama
int main() {

  cout<<"Nilai yang terdapat dalam fungsi : ";
  cout<<FungsiBilangan();

  return 0;
}
```

FUNGSI DENGAN PARAMETER

- Sejauh ini masih dibuat fungsi yang tidak memiliki parameter sehingga setiap kali pemanggilan fungsi tersebut, hasil yang didapatkan selalu bernilai ***tetap***.
- Berikutnya akan dibuat fungsi dengan parameter sehingga hasil yang diberikan dapat bersifat ***dinamis***, tentunya tergantung dari ***nilai parameter yang dimasukan***.

FUNGSI DENGAN PARAMETER

- Secara teori, *parameter yang terdapat pada pendefinisian sebuah fungsi* disebut dengan nama ***parameter formal***, sedangkan *parameter yang terdapat pada saat pemanggilan* disebut dengan ***parameter aktual***.
- Jumlah serta tipe data antara ***parameter formal*** dan ***parameter aktual*** haruslah *sesuai*, jika tidak maka compiler akan menampilkan *pesan kesalahan*.

FUNGSI DENGAN PARAMETER

- Ada juga referensi yang menyebut ***parameter formal*** dengan istilah ***parameter,*** sedangkan ***parameter aktual*** dengan istilah ***argumen***.

JENIS PARAMETER

- Terdapat 3 buah parameter yang dapat dilewatkan pada sebuah fungsi dalam C++, yaitu :
  a. Parameter Masukan
  b. Parameter Keluaran
  c. Parameter Massukan/Keluaran

PARAMETER MASUKAN

- Sesuai dengan namanya, parameter ini akan digunakan sebagai ***nilai masukan (input)*** dalam sebuah ***fungsi***.
- Nilai tersebut kemudian akan diproses oleh ***fungsi*** untuk menghasilkan sebuah ***nilai kembalian (return value).***

PARAMETER MASUKAN

```
#include <iostream>

using namespace std;

// Membuat fungsi dengan parameter bertipe masukan
int KaliDua(int X) {
  int hasil;
  hasil = X * 2;
  return hasil;
}

int main() {

  /* Mendeklarasikan variabel yang akan digunakan sebagai nilai parameter
     pada saat pemanggilan */
  int Bilangan, HASIL;

  cout<<"Masukkan sebuah bilangan bulat : ";
  cin>>Bilangan;

  HASIL = KaliDua(Bilangan);

  // Menampilkan nilai setelah diproses di dalam fungsi
  cout<<"Nilai Hasilnya : "<<HASIL;

  return 0;
}
```

PARAMETER KELUARAN

- ***Parameter keluaran*** adalah parameter yang berfungsi untuk ***menampung nilai yang akan dikembalikan***.
- Parameter tersebut berguna sebagai ***nilai keluaran (output)*** dari sebuah fungsi.
- ***Parameter keluaran*** harus berupa ***pointer*** maupun ***referensi*** yang pada umumnya terdapat pada fungsi – fungsi yang ***tidak mengembalikan nilai***.

PARAMETER MASUKAN/KELUARAN

- Jenis parameter ini adalah parameter yang digunakan sebagai ***masukan*** dan juga ***keluaran***.
- Karena akan diperankan sebagai ***parameter keluaran***, maka parameter jenis ini harus berupa ***pointer*** ataupun ***referensi***.

PARAMETER BERTIPE ARRAY

- ***Array*** dapat juga dijadikan sebagai ***parameter*** dalam sebuah ***fungsi***.
- Sebagai contoh, disini akan dibuat dua fungsi yang memiliki ***parameter*** bertipe ***array***.

PARAMETER BERTIPE POINTER

- Pada prakteknya, dalam program C++ ***parameter*** yang bertipe ***array*** ini sering diganti dengan parameter yang bertipe ***pointer***.
- Berikut ini contoh program yang memperlihatkan bahwa parameter dengan tipe ***array*** dapat diganti dengan tipe ***pointer***.

PARAMETER KONSTAN

- Sejauh ini hanya dibuat fungsi dengan ***parameter-parameter*** yang dapat diubah nilainya.
- ***Parameter*** sebuah ***fungsi*** dapat bersifat tetap (konstan), artinya didalam ***fungsi nilai parameter tidak dapat diubah***.
- Untuk melakukan hal ini, maka perlu ditambahkan kata kunci ***`const`*** didepan deklarasi parameter tersebut.

PARAMETER DENGAN NILAI DEFAULT

- Nilai dari ***parameter*** yang terdapat dalam sebuah fungsi dapat diinisialisasi dengan ***nilai awal (default)***.
- Dengan cara seperti ini, jika tidak akan mendefinisikan nilai pada saat pemanggilan fungsi, maka nilai yang akan digunakan oleh fungsi adalah ***nilai default***.
- Untuk melakukan hal ini maka perlu diisikan nilai parameter bersangkutan pada saat mendefinisikan fungsi.

PARAMETER DENGAN NILAI DEFAULT

- Bentuk umum dari pembuatan fungsi yang menggunakan nilai ***default*** pada parameternya adalah seperti yang tampak dibawah ini :

```
tipe_data nama_fungsi(tipe_data parameter1=nilai_default, …){
…
return nilai_kembalian
}
```

PARAMETER DENGAN NILAI DEFAULT

```
#include <iostream>
using namespace std;
// Mendefinisikan fungsi untuk menghitung volume balok
long VolumeBalok(int panjang, int lebar = 20, int tinggi = 5) {
  return (panjang * lebar * tinggi);
}
int main() {
  // Mendeklarasikan variabel p dengan nilai 50
  int p = 50;
  // Mendeklarasikan variabel l dengan nilai 10
  int l = 10;
  // Mendeklarasikan variabel t dengan nilai 2
  int t = 2;
  long hasil;
  // Memanggil fungsi dengan tiga parameter
  hasil = VolumeBalok(p, l, t); cout<<"Volume balok = "<<hasil<<endl;

  // Memanggil fungsi dengan dua parameter
  hasil = VolumeBalok(p, l); cout<<"Volume balok = "<<hasil<<endl;

  // Memanggil fungsi dengan satu parameter
  hasil = VolumeBalok(p); cout<<"Volume balok = "<<hasil;

  return 0;
}
```

POINTER KE FUNGSI

- Meskipun ***fungsi*** bukan sebuah ***variabel***, namun fungsi masih merupakan objek yang memiliki ***lokasi fisik*** di memori.
- Hal ini menunjukan bahwa nilai dari sebuah fungsi dapat diambil melalui pointer.
- Perlu sekali untuk diperhatikan bahwa cara untuk mendapatkan alamat dari sebuah fungsi adalah dengan menyebutkan nama fungsi tersebut ***tanpa tanda kurung*** maupun ***parameter***.

POINTER KE FUNGSI

- Sebagai contoh, misalnya ***pointer*** P menunjuk ke tipe **`int`** dan terdapat fungsi **`tambah()`** yang mengembalikan tipe **`int`**. dalam situasi seperti ini, dapat diperintahkan ***pointer P*** untuk menunjuk ke alamat dari fungsi **`tambah()`**.

MENGEMBALIKAN NILAI BERTIPE POINTER

- Dalam C++, kita dapat membuat fungsi yang mengembalikan nilai berupa pointer.
- Untuk melakukan hal ini tentu kita harus mendeklarasikan tipe data dari nilai kembalian dengan pointer.

MEMBUAT PROTOTIPE FUNGSI

- Dalam C++, ***fungsi-fungsi*** dapat dideklarasikan terlebih dahulu sebelum dilakukan ***pendefinisian***.
- Definisi fungsi baru dibuat setelah pembuatan fungsi utama, hal ini disebut dengan istilah ***prototipe fungsi***.
- Prototipe akan mempermudah kita dalam ***mengenali daftar fungsi*** yang tersedia atau akan didefinisikan didalam program.

MEMBUAT PROTOTIPE FUNGSI

- Bentuk umum dari pembuatan ***prototipe fungsi*** adalah seperti tampak dibawah ini.

```
Tipe_data nama_fungsi(parameter1, parameter2, …);
```

MEMBUAT PROTOTIPE FUNGSI

```
#include <iostream>
using namespace std;

// Membuat prototipe (deklarasi fungsi) Mendeklarasikan fungsi Kali()
int Kali(int X, int Y);
// Mendeklarasikan fungsi Tulis()
void Tulis(int S);

// Fungsi Utama
int main() {
  int Bilangan1, Bilangan2, HASIL;
  cout<<"Masukkan bilangan pertama : ";   cin>>Bilangan1;
  cout<<"Masukkan bilangan kedua   : ";   cin>>Bilangan2;
  cout<<endl;

  // Menggunakan fungsi Kali()
  HASIL = Kali(Bilangan1, Bilangan2);

  // Menggunakan fungsi Tulis()
  Tulis(HASIL);
  return 0;
}
// Membuat definisi fungsi Kali()
int Kali(int X, int Y) {
  return (X * Y);
}
// Membuat definisi fungsi Tulis()
void Tulis(int S) {
  cout<<S<<endl;
}
```

FUNGSI INLINE

- Pada saat didefinisikan sebuah ***fungsi***, ***compiler*** akan ***membuat satu set statemen didalam memori***.
- Ketika pemanggilan fungsi berjalan maka eksekusi program akan *meloncat ke statemen-statemen* tersebut.
- Kemudian setelah ***fungsi mengembalikan nilai***, maka eksekusi program akan kembali meloncat ke baris selanjutnya (baris setelah pemanggilan fungsi).

FUNGSI INLINE

- Apabila fungsi yang didefinisikan hanya terdiri dari sedikit statemen (misalnya 1 ata 2 baris), hal ini tentu akan mengurangi ***efisiensi*** program karena ***compiler*** harus meloncat keluar masuk untuk melakukan proses tersebut.
- Untuk menghindari hal ini C++, menyediakan fitur yang disebut dengan ***fungsi inline (inline function)***, yaitu menyertakan keyword **`inline`** didepan definisi fungsi.

FUNGSI INLINE

- Konsep dasar dari ***inline*** ini adalah proses ***penyalinan (copy)*** baris yang terdapat pada saat dilakukan pemanggilan fungsi tersebut
- Artinya jika dibuat fungsi inline, ***compiler*** *tidak menyimpannya ke dalam memori* melainkan akan *hanya membuat salinan kode dari fungsi* tersebut.
- Hal ini tentu tidak membutuhkan proses ***peloncatan statement*** sehingga *proses eksekusinya akan lebih cepat*.

FUNGSI INLINE

```
#include <iostream>
using namespace std;

// Mendefinisikan fungsi inline yang mengalikan bilangan dengan 2
inline int Kali2(int X) {
  return X * 2;
}

// Fungsi utama
int main() {
  int HASIL;

  // Melakukan pemanggilan fungsi inline untuk pertama kali
  HASIL = Kali2(10);
  cout<<"Hasil = "<<HASIL<<endl;

  // Melakukan pemanggilan fungsi inline untuk kedua kali
  HASIL = Kali2(20);
  cout<<"Hasil = "<<HASIL<<endl;

  // Melakukan pemanggilan fungsi inline untuk ketiga kali
  HASIL = Kali2(30);
  cout<<"Hasil = "<<HASIL<<endl;

  return 0;
}
```

REKURSI

- ***Rekursi*** adalah fungsi yang pada saat pendefinisiannya memanggil dirinya sendiri untuk melakukan proses didalamnya.
- Contoh paling sederhana untuk menunjukan proses rekursi adalah pada saat kita mebuat program untuk menghitung nilai faktorial dari sebuah bilangan bulat.

REKURSI

```
#include <iostream>

using namespace std;

// Mendefinisikan fungsi Faktorial()
int Faktorial(int X) {
  if (X==1) return(1);
  // Memanggil dirinya sendiri
  return X * Faktorial(X-1);
}

// Fungsi utama
int main() {
  int Bilangan, HASIL;
  cout<<"Masukkan bilangan yang akan dihitung : ";
  cin>>Bilangan;

  // Memanggil fungsi Faktorial()
  HASIL = Faktorial(Bilangan);

  // Menampilkan hasil
  cout<<Bilangan<<"! = "<<HASIL;

  return 0;
}
```