INF-103 Pemrograman II

FUNGSI C++

Dr. Taufik Fuadi Abidin, M.Tech
Irvanizam, M.Sc

Program Studi Informatika
FMIPA UNIVERSITAS KUALA

http://www.informatika.unsyiah.ac.id

Fungsi (*Function*)

- Eksekusi sebuah program dimulai dari fungsi `main()`

- Program biasanya dirancang dalam beberapa sub program yang disebut fungsi (*function*)

- Ketika program dieksekusi dan bertemu dengan pernyataan memanggil sebuah fungsi maka eksekusi program akan beralih ke fungsi tersebut. Setelah fungsi selesai dieksekusi, pernyataan dilanjutkan tepat pada bagian dimana fungsi itu dipanggil

Contoh

```
#include <iostream>

using namespace std;

void echo(const string str){
   cout << str << endl;
}

int main(){
   string kata;
   cout << "Ketik sebuah kata: ";
   cin >> kata;
   echo(kata);
}
```

Definisi Fungsi

```
function header{
    statements
}
```

- Pernyataan sebelum tanda kurung kurawal buka disebut sebagai *header* dari fungsi dan pernyataan setelah tanda kurung kurawal buka dan tutup disebut *body* dari fungsi

- Header dari sebuah fungsi ditulis dalam bentuk:

  tipe nama_fungsi (*daftar parameter*)

- Tipe adalah nilai yang akan dikembalikan oleh fungsi. Fungsi yang tidak mengembalikan nilai bertipe *void*.

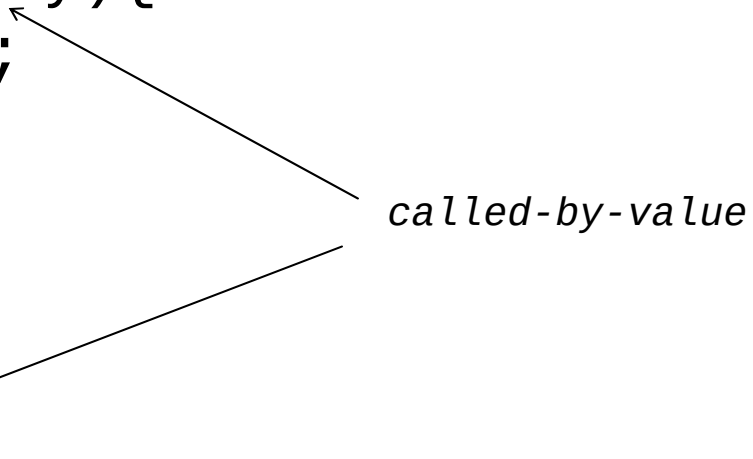
Fungsi: Pernyataan *return*

- Ketika pernyataan *return* dieksekusi, eksekusi sebuah fungsi akan berhenti dan nilai akan dikembalikan ke posisi dimana fungsi tersebut dipanggil

```
int maximum(int x, int y){
   if (x > y) return x;
   else return y;
}
```

atau

```
int maximum(int x, int y){
   return ((x > y)? x : y);
}
```



Fungsi: Called-by-Value

```
#include <iostream>
using namespace std;
int komulatif(int n){
  int sum = 0;
  for ( ; n > 0; --n)
     sum += n;
  return sum;
}
int main(){
  int n = 3, sum;
  cout << n << endl;
  sum = komulatif(n);
  cout << n << " : " << sum << endl;
}
```

Fungsi Rekursif

Fungsi rekursif (*recursive function*) memanggil dirinya sendiri dalam bagian *body*. Fungsi rekursif selalu memiliki bagian terminasi untuk menghentikan proses rekursif.

```
long factorial(int n){
  if (n <= 1)
    return;
  else
    return n * factorial(n - 1);
}
```

Fungsi Rekursif: Lanjutan

```
void countdown(unsigned int n){
  if (n <= 0)
    cout << "BLAST OFF" << endl;
  else {
    cout << "COUNT: " << n << endl;
    countdown(n - 1);
  }
}
```

Default Argumen

Default argumen merupakah fitur yang unik dari C++ yang ditulis dibagian *header* sebuah fungsi. Default argumen tidak ditemukan dalam C atau Java.

Contoh:

```
void author(string date, string ver,
     string programmer = "Taufik Abidin"){

  cout << programmer <<", ";
  cout << "v: " << ver <<", ";
  cout << date << endl;
}
```

Default Argumen


```
int main(){
  author("1/1/2005", "1.30");
  author("1/1/2008", "2.75", "Khalid");
}
```

Output

```
     Taufik Abidin, v:1.30, 1/1/2005
     Khalid, v: 2.75, 1/1/2008
```

```
void foo(int i, int j = 7);          // yes
void foo(int i=3, int j);            // no
void foo(int i, int j=1, int k=2); // yes
void foo(int i=1, int j=1, int k); // no
```

Default Argumen + Rekursif


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

int pangkat(int n, int k = 2){
  assert(k > 1);          // if false aborts
  if (k == 2) return (n * n);
  else return (pangkat(n, k - 1) * n);
}

int main(){
  int y = pangkat(2,3);
  cout << y << endl;
}
```

Fungsi sebagai Argumen

```
double f(double x){return (x * x + 1.0 / x);}

void plot(double fcn(double), double x0,
          double incr, int n){
  for (int i = 0; i < n; ++i) {
    cout << " x :" << x0 << "
    f(x) : " << fcn(x0) << endl;
    x0 += incr;
  }
}
int main(){
  cout << "Plot: " << endl;
  plot(f, 0.01, 0.01, 100);
}
```

Overloading

- *Overloading* adalah mekanisme membuat definisi sebuah fungsi dengan nama yang sama tetapi tipe argumen atau jumlah argumen berbeda

- Compiler akan menentukan fungsi untuk dieksekusi berdasarkan kecocokan terhadap tipe dan argumennya

Overloading: Contoh


```
double average(const int size, int& sum){

int data;
  cout << "\nInput " << size;
  cout << " int: " << endl;

  for (int i = 0; i < size; ++i) {
    cin >> data;
    sum += data;
  }

  return static_cast<double>(sum)/size;
}
```

Overloading: Contoh Lanjutan


```
double average(const int size, double& sum){

  double data;
  cout << "\nEnter " << size;
  cout << " doubles: " << endl;

  for (int i = 0; i < size; ++i) {
    cin >> data;
    sum += data;
  }

  return sum / size;
}
```

Overloading: Contoh Lanjutan


```
int main(){

  int isum = 0;
  double dsum = 0.0;
  cout << average(10, isum);
  cout << " int average" << endl;

  cout << average(10, dsum);
  cout << " double average" << endl;

}
```

Inline pada Fungsi

- Penggunaan kata kunci *inline* didepan definisi sebuah fungsi mengakibatkan pada saat fungsi tersebut dieksekusi, bagian *body* dari fungsi tersebut dicopy seluruhnya pada baris dimana fungsi tersebut dipanggil

- Fungsi tanpa *inline* akan mengakibatkan *function call invocation* (penundaan eksekusi dan men-stack proses) pada saat pemanggilan fungsi terjadi

- Penggunaan *inline* akan mempercepat proses eksekusi program

Inline: Contoh

```
inline double cube(double x){
  return (x * x * x);
}
```

Macro

```
#define SQR(X) ((X) * (X))
#define CUBE(X) (SQR(X)*(X))
```

Macro atau inline?

Argumen dalam *macro* tidak memiliki tipe sehingga tidak aman dan dapat menyebabkan kesalahan. Oleh karena itu, penggunaan *macro* tidak direkomendasikan dalam C++.

Inline: Contoh Lanjutan

```
inline int min (const int x, const int y){
  return (x < y ? x : y);
}
```

Pemanggilan fungsi *min* dalam program:

```
int main(){
  int a = 5 b = 3;
  int minimum = min(a,b);
}
```

Mengakibatkan pada saat kompilasi, baris 3 diganti menjadi:

```
  int minimum = (x < y ? x : y);
```

Tugas Bacaan:

Ira Pohl

C++ by Dissection: The Essentials of C++ Programming

Halaman:

99 – 104 (Pointer)