Pertemuan 5

5. Function

Obyektif Praktikum :

1. Mengerti konsep dasar penggunaan Function
2. Memahami Definisi Fungsi, Fungsi tanpa nilai balik dan Ruang lingkup variabel yang digunakan.

P.5.1 Function

Function adalah satu blok instruksi yang dieksekusi ketika dipanggil dari bagian lain dalam suatu program. Format dari *function* :

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

Dimana :

- ***type***, adalah tipe dari data yang akan dikembalikan/dihasilkan oleh *function*.
- ***name***, adalah nama yang memungkinkan kita memanggil function.
- ***arguments*** (dispesifikasikan sesuai kebutuhan). Setiap argumen terdiri dari tipe data diikuti identifier, seperti deklarasi variable (contoh, `int x`) dan berfungsi dalam function seperti variable lainnya. Juga dapat melakukan *passing parameters* ke function itu ketika dipanggil. Parameter yang berbeda dipisahkan dengan koma.
- ***statement***, merupakan bagian badan suatu *function*. Dapat berupa instruksi tunggal maupun satu blok instruksi yang dituliskan diantara kurung kurawal **{}**.

Program diatas, ketika dieksekusi akan mulai dari fungsi **main**. **main** function memulai dengan deklarasi variabel **z** dengan tipe **int**. Setelah itu instruksi pemanggilan fungsi **addition**. Jika diperhatikan, ada kesamaan antara sruktur pemanggilan dengan deklarasi fungsi itu sendiri, perhatikan contoh dibawah ini :

```
int addition (int a, int b)
                 ↑     ↑
z = addition (  5  ,   3  );
```

Instruksi pemanggilan dalam fungsi **main** untuk fungsi **addition**, memberikan 2 nilai : **5** dan **3** mengacu ke parameter **int a** dan **int b** yang dideklarasikan untuk fungsi **addition**.

Saat fungsi dipanggil dari **main**, kontrol program beralih dari fungsi **main** ke fungsi **addition**. Nilai dari kedua parameter yg diberikan (**5** dan **3**) di-*copy* ke variable local ; **int a** dan **int b**.
Fungsi **addition** mendeklarasikan variable baru (**int r;**), kemudian ekspresi **r=a+b;**, yang berarti **r** merupakan hasil penjumlahan dari **a** dan **b**, dimana **a** dan **b** bernilai **5** dan **3** sehingga hasil akhirnya **8**. perintah selanjutnya adalah :

```
return (r);
```

Merupakan akhir dari fungsi **addition**, dan mengembalikan kontrol pada fungsi **main**. Statement **return** dikuti dengan variabel **r** (**return (r);**), sehingga nilai dari **r** yaitu 8 akan dikembalikan :

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

Dengan kata lain pemanggilan fungsi (**addition (5,3)**) adalah menggantikan dengan nilai yang akan dikembalikan (**8**).

Perhatikan penulisan pemanggilan *function,* format penulisan pada dasarnya sama.
Contoh 1 :

```
z = subtraction (7,2);
cout << "The first result is " << z;
```

Contoh 2 :

```
cout << "The second result is " << subtraction
(7,2);
```

Contoh 3 :

```
cout << "The third result is " << subtraction (x,y);
```

Hal lain dari contoh diatas, parameter yang digunakan adalah variable, bukan konstanta. Contoh diatas memberikan nilai dari **x** dan **y**, yaitu **5** dan **3**, hasilnya **2**.

contoh 4:

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

Atau dapat dituliskan :

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

Akan memberikan hasil akhir yang sama. Perhatikan, pada setiap akhir ekspresi selalu diberi tanda semicolon (**;**).

***Function* tanpa tipe (Kegunaan *void*)**

Deklarasi fungsi akan selalu diawali dengan tipe dari fungsi, yang menyatakan tipe data apa yang akan dihasilkan dari fungsi tersebut. Jika tidak ada nilai yang akan dikembalikan, maka dapat digunakan tipe **void**,

Walaupun pada C++ tidak diperlukan men-spesifikasikan **void**, hal itu digunakan untuk mengetahui bahwa fungsi tersebut tidak mempunyai argumen, atau parameter dan lainnya. Maka dari itu pemanggilan terhadap fungsinya dituliskan :

```
dummyfunction ();
```

Argument passed *by value* dan *by reference*.

Parameter yang diberikan ke fungsi masih merupakan *passed by value*. Berarti, ketika memanggil sebuah fungsi, yang diberikan ke fungsi adalah nilainya, tidak pernah men-spesifikasikan variablenya. Sebagai Contoh, pemanggilan fungsi **addition, menggunakan perintah berikut :**

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

Yang berarti memanggil fungsi **addition** dengan memberikan nilai dari **x** dan **y**, yaitu **5** dan **3**, bukan variabelnya.

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

Tetapi, dapat juga memanipulasi dari dalam fungsi, nilai dari variable external. Untuk hal itu, digunakan argument *passed by reference*

Perhatikan deklarasi **duplicate**, tipe pada setiap argumen diakhiri dengan tanda *ampersand* (**&**), yang menandakan bahwa variable tersebut biasanya akan *passed by reference* dari pada *by value*.
Ketika mengirimkan variable *by reference*, yang dikirimkan adalah variabelnya dan perubahan apapun yang dilakukan dalam fungsi akan berpengaruh pada variable diluarnya.

```
void duplicate (int& a,int& b,int& c)
                  x     y     z
     duplicate (  x  ,  y  ,  z  );
```

Atau dengan kata lain, parameter yang telah ditetapkan adalah **a**, **b** dan **c** dan parameter yang digunakan saat pemanggilan adalah **x**, **y** dan **z**, maka perubahan pada **a** akan mempengaruhi nilai **x**, begitupun pada **b** akan mempengaruhi **y**, dan **c** mempengaruhi **z**.

Itu sebabnya mengapa hasil output dari program diatas adalah nilai variable dalam main dikalikan 2. jika deklarasi fungsi tidak diakhiri dengan tanda ampersand (&), maka variable tidak akan *passed by reference*, sehingga hasilnya akan tetap nilai dari **x**, **y** dan **z** tanpa mengalami perubahan.

Passing by reference merupakan cara efektif yang memungkinkan sebuah fungsi mengembalikan lebih dari satu nilai.

Nilai *Default* dalam argument

Ketika mendeklarasikan sebuah fungsi, dapat diberikan nilai *default* untuk setiap parameter. nilai ini akan digunakan ketika parameter pemanggil dikosongkan. Untuk itu cukup dideklarasikan pada saat deklarasi fungsi, Dapat dilihat dalam fungsi **divide**. Instruksi 1:

```
divide (12)
```

Instruksi 2 :

```
divide (20,4)
```

Fungsi Overloaded function

Dua fungsi yang berbeda dapat memiliki nama yang sama jika prototype dari argumen mereka berbeda, baik jumlah argumennya maupun tipe argumennya,

inline Function

Directive inline dapat disertakan sebelum deklarasi fungsi, untuk menspesifikasikan bahwa fungsi tersebut

harus di-compile sebagai suatu kode saat dipanggil. Sama halnya dengan deklarasi *macro*. Keuntungannya dapat terlihat pada fungsi sederhana yaitu hasil yang diberikan akan lebih cepat. (jika terjadi *stacking of arguments*) dapat dihindari. Format deklarasi :

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

Pemanggilannya, sama dengan pemanggilan fungsi pada umumnya. Tidak diperlukan penulisan keyword *inline* pada setiap pemanggilan.

Recursivity Function

Rekursif merupakan kemempuan sebuah fungsi untuk memanggil dirinya sendiri. Sangat berguna untuk pengerjaan sorting atau perhitungan factorial. Contoh, format perhitungan factorial :

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

Misalkan, 5! (5 faktorial), akan menjadi :

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

Prototyping function.

Format :

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

Hampir sama dengan deklarasi fungsi pada umumnya, **kecuali** :

- Tidak ada *statement* fungsi yang biasanya dituliskan dalam kurung kurawal { }.
- Diakhiri dengan tanda semicolon (;).
- Dalam argumen dituliskan tipe argumen, bersifat optional.

P.5.2 Contoh Kasus

- *function example* :

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

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

int main ()
{
  int x=5, y=3, z;
  z = subtraction (7,2);
  cout << "The first result is " << z << '\n';
  cout << "The second result is " << subtraction (7,2) << '\n';
```

```
  cout << "The third result is " << subtraction (x,y) << '\n';
  z= 4 + subtraction (x,y);
  cout << "The fourth result is " << z << '\n';
  return 0;
}
```

Output :

```
The first result is 5
The second result is 5
The third result is 2
The fourth result is 6
```

Fungsi diatas melakukan pengurangan dan mengembalikan hasilnya. Jika diperhatikan dalam fungsi **main**, dapat dilihat beberapa cara pemanggilan fungsi yang berbeda.

- *overloaded function*

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

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

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

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

Output :

```
2
2.5
```

Contoh diatas mempunyai nama fungsi yang sama, tetapi argumennya berbeda. Yang pertama bertipe **int** dan lainnya bertipe **float**. Kompiler mengetahuinya dengan memperhatikan tipe argumen pada saat pemanggilan fungsi.

- *factorial calculator*

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

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

int main ()
{
  long l;
  cout << "Type a number: ";
  cin >> l;
  cout << "!" << l << " = " << factorial (l);
  return 0;
}
```

Output :

```
Type a number: 9
!9 = 362880
```

- *Prototyping*

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

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

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

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

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

Output :

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

Contoh diatas tidak menjelaskan tentang efektivitas program tetapi bagaimana prototyping dilaksanakan. Perhatikan prototype dari fungsi **odd** dan **even**:

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

Memungkinkan fungsi ini dipergunakan sebelum didefinisikan. Hal lainnya mengapa program diatas harus memiliki sedikitnya 1 fungsi prototype, karena fungsi dalam **odd** terdapat pemanggilan fungsi **even** dan dalam **even** terdapat pemanggilan fungsi **odd**. Jika tidak satupun dari fungsi tersebut dideklarasikan sebelumnya, maka akan terjadi error.

P.5.3. Latihan

1. Apa output untuk program dibawah ini :

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

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

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

2. Apa output untuk program dibawah ini :

```
// void function example
#include <iostream.h>
void dummyfunction (void)
{
  cout << "I'm a function!";
}

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

3. Apa output untuk program dibawah ini :

```
// more than one returning value
#include <iostream.h>

void prevnext (int x, int& prev, int& next)
{
  prev = x-1;
  next = x+1;
}
int main ()
{
  int x=100, y, z;
  prevnext (x, y, z);
  cout << "Previous=" << y << ", Next=" << z;
  return 0;
}
```

4. Apa output untuk program dibawah ini :

```
// default values in functions
#include <iostream.h>
int divide (int a, int b=2)
{
  int r;
  r=a/b;
  return (r);
}
int main ()
{
  cout << divide (12);
  cout << endl;
  cout << divide (20,4);
```

```
  return 0;
}
```

P. 5.4 Daftar Pustaka

1. Ayuliana, modul pengenalan bahasa C++, Gunadarma Jakarta, February 2004

2. Hari, Konsep Dasar Objek Oriented Programming, FTI budiluhur Jakarta, 2003

3. r.hubbard, John , schaum's outline of theory and problems of programming with C++ second edition, mcgraw-hill, New York 2000

4. http://www.cplusplus.com/

5. http://cs.binghamton.edu/~steflik/

6. http://en.wikipedia.org/wiki/c++