PRAKTIKUM 7

FUNGSI 1

A. Tujuan

1. Menjelaskan pengertian fungsi
2. Membuat Fungsi
3. Mengerti parameter dalam fungsi
4. Memahami cara melewatkan parameter ke dalam fungsi

B. DASAR TEORI

Fungsi adalah suatu bagian dari program yang dirancang untuk melaksanakan tugas tertentu dan letaknya dipisahkan dari program yang menggunakannya. Elemen utama dari program bahasa C berupa fungsi-fungsi, dalam hal ini program dari bahasa C dibentuk dari kumpulan fungsi pustaka (standar) dan fungsi yang dibuat sendiri oleh pemrogram. Fungsi banyak digunakan pada program C dengan tujuan :

a. Program menjadi terstruktur, sehingga mudah dipahami dan mudah dikembangkan. Dengan memisahkan langkah-langkah detail ke satu atau lebih fungsi-fungsi, maka fungsi utama (*main()*) menjadi lebih pendek, jelas dan mudah dimengerti.
b. dapat mengurangi pengulangan (duplikasi) kode. Langkah-langkah program yang sama dan dipakai berulang-ulang di program dapat dituliskan sekali saja secara terpisah dalam bentuk fungsi-fungsi. Selanjutnya bagian program yang membutuhkan langkah-langkah ini tidak perlu selalu menuliskannya, tetapi cukup memanggil fungsi-fungsi tersebut.

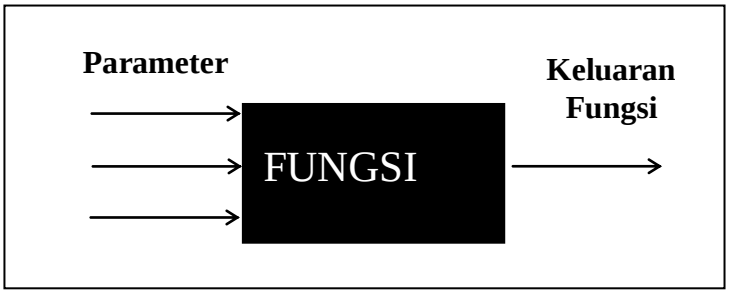
Dasar Fungsi

Fungsi standar C yang mengemban tugas khusus contohnya adalah ;

- *printf()* , yaitu untuk menampilkan informasi atau data ke layar.
- *scanf()* , yaitu untuk membaca kode tombol yang diinputkan.

Pada umumnya fungsi memerlukan nilai masukan atau parameter yang disebut sebagai argumen. Nilai masukan ini akan diolah oleh fungsi. Hasil akhir fungsi berupa sebuah nilai (disebut sebagai *return value* atau nilai keluaran fungsi). Oleh karena itu

fungsi sering digambarkan sebagai "kotak gelap" seperti ditunjukkan pada gambar di bawah ini.



Gambar 8.1 Fungsi sebagai sebuah kotak gelap

Penggambaran sebagai kotak gelap di antaranya menjelaskan bahwa bagian dalam fungsi bersifat pribadi bagi fungsi. Tak ada suatu pernyataan di luar fungsi yang bisa mengakses bagian dalam fungsi, selain melalui parameter (atau variabel eksternal yang akan dibahas belakangan). Misalnya melakukan *goto* dari pernyataan di luar fungsi ke pernyataan dalam fungsi adalah tidak diperkenankan.

Bentuk umum dari definisi sebuah fungsi adalah sebagai berikut ;

```
tipe-keluaran-fungsi  nama-fungsi (deklarasi argumen)
{
        tubuh fungsi
}
```

Keterangan :

- **tipe-keluaran-fungsi**, dapat berupa salah satu tipe data C, misalnya *char* atau *int* . Kalau penentu tipe tidak disebutkan maka dianggap bertipe *int* (secara *default*).
- **tubuh fungsi** berisi deklarasi variabel (kalau ada) dan statemen-statemen yang akan melakukan tugas yang akan diberikan kepada fungsi yang bersangkutan. Tubuh fungsi ini ditulis di dalam tanda kurung kurawal buka dan kurung kurawal tutup.

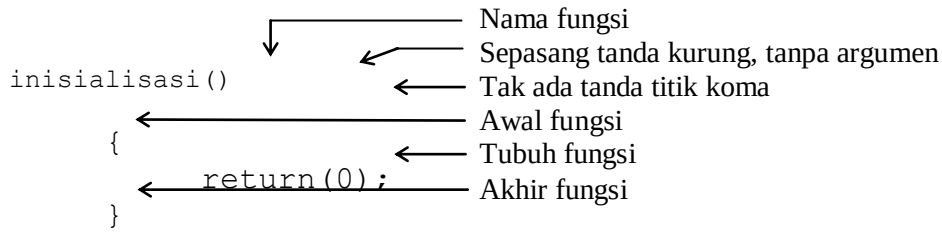
Sebuah fungsi yang sederhana bisa saja tidak mengandung parameter sama sekali dan tentu saja untuk keadaan ini deklarasi parameter juga tidak ada. Contoh :

```
int inisialisasi()

      {
            return(0);
      }
```

```
inisialisasi()

{
     return(0);
}
```

Pada fungsi di atas :

- tipe keluaran fungsi tidak disebutkan, berarti keluaran fungsi ber tipe *int*.
- `inisialisasi` adalah nama fungsi
- Tanda `()` sesudah nama fungsi menyatakan bahwa fungsi tak memiliki parameter.
- Tanda `{` dan `}` adalah awal dan akhir fungsi
- `return(0)` merupakan sebuah pernyataan dalam tubuh fungsi.



Gambar 8.2 Penjelasan definisi sebuah fungsi

Memberikan Nilai Keluaran Fungsi

Suatu fungsi dibuat untuk maksud menyelesaikan tugas tertentu. Suatu fungsi dapat hanya melakukan suatu tugas saja tanpa memberikan suatu hasil keluaran atau melakukan suatu tugas dan kemudian memberikan hasil keluaran. Fungsi yang hanya melakukan suatu tugas saja tanpa memberikan hasil keluaran misalnya adalah fungsi untuk menampilkan hasil di layar.

Dalam tubuh fungsi, pernyataan yang digunakan untuk memberikan nilai keluaran fungsi berupa *return.* Sebagai contoh, pada fungsi **inisialisasi()** di atas terdapat pernyataan

```
return(0);
```

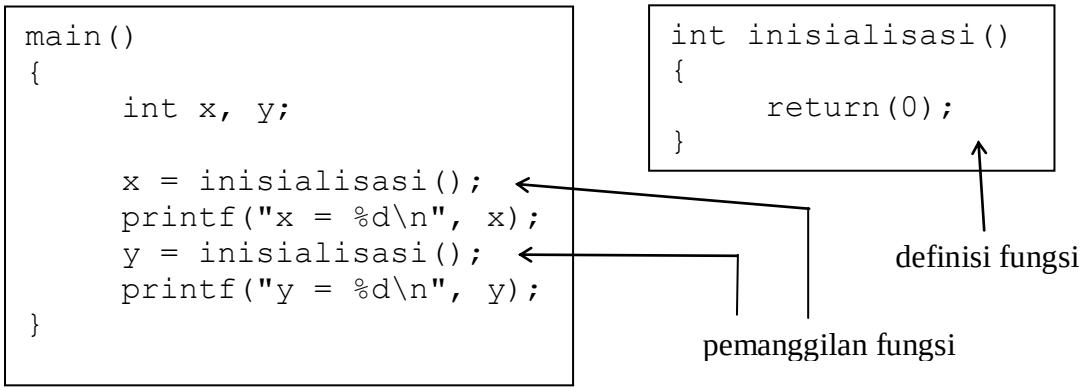
merupakan pernyataan untuk memberikan nilai keluaran fungsi berupa nol. Selengkapnya perhatikan program di bawah ini

```
/* File program : inisial.c
Contoh pembuatan fungsi */

int inisialisasi();

#include <stdio.h>

main()
{
     int x, y;
     x = inisialisasi();
     printf("x = %d\n", x);
     y = inisialisasi();
     printf("y = %d\n", y);
}

int inisialisasi()
{
     return(0);
}
```

Contoh eksekusi :

```
x = 0
y = 0
```

Program di atas sekaligus menjelaskan bahwa suatu fungsi cukup didefinisikan satu kali tetapi bisa digunakan beberapa kali. Pada keadaan semacam ini seandainya tubuh fungsi mengandung banyak pernyataan, maka pemakaian fungsi dapat menghindari duplikasi kode dan tentu saja menghemat penulisan program maupun kode dalam memori.



Gambar 8.3 Proses pemanggilan fungsi

Misalnya pada saat pernyataan

```
x = inisialisasi();
```

dijalankan, mula-mula eksekusi akan diarahkan ke fungsi **inisialisasi()**, selanjutnya suatu nilai keluaran (hasil fungsi) akhir fungsi diberikan ke **x**. Proses yang serupa, dilakukan untuk pernyataan

```
y = inisialisasi();
```

Bagi suatu fungsi, jika suatu pernyataan *return* dieksekusi, maka eksekusi terhadap fungsi akan berakhir dan nilai pada parameter *return* akan menjadi keluaran fungsi. Untuk fungsi yang tidak memiliki pernyataan *return*, tanda } pada bagian akhir fungsi akan menyatakan akhir eksekusi fungsi.

Di bawah ini diberikan contoh sebuah fungsi yang mengandung dua buah pernyataan *return*. Fungsi digunakan untuk memperoleh nilai minimum di antara 2 buah nilai yang menjadi parameternya.

```
int minimum(int x, int y)
{
     if (x < y)
          return(x);
     else
          return(y);
}
```

Pada fungsi di atas terdapat dua buah parameter berupa **x** dan **y**. Oleh karena itu fungsi juga mengandung bagian untuk mendeklarasikan parameter, yang menyatakan **x** dan **y** bertipe *int*. Adapun penentuan nilai keluaran fungsi dilakukan pada tubuh fungsi, berupa pernyataan

```
if (x < y)
     return(x);
else
     return(y);
```

yang menyatakan :

- jika x < y maka nilai keluaran fungsi adalah sebesar nilai x.
- untuk keadaan lainnya (x >= y) maka keluaran fungsi adalah sebesar y.

Selengkapnya perhatikan program di bawah ini.

```
/* File program : minimum1.c */

#include <stdio.h>

int minimum (int, int);

main()
{
     int a, b, kecil;

     printf("Masukkan nilai a : ");
     scanf("%d", &a);
     printf("Masukkan nilai b : ");
     scanf("%d", &b);

     kecil = minimum(a, b);
     printf("\nBilangan terkecil antara %d dan %d adalah
          %d\n\n", a, b, kecil);
}

minimum(int x, int y)
{
     if (x < y)
          return(x);
     else
          return(y);
}
```

Contoh eksekusi :

```
Masukkan nilai a = 4
Masukkan nilai b = 2

Bilangan terkecil antara 4 dan 2 adalah 2
```

Fungsi Dengan Keluaran Bukan Integer

Untuk fungsi yang mempunyai keluaran bertipe bukan integer, maka fungsi haruslah didefiniskan dengan diawali tipe keluaran fungsinya (ditulis di depan nama fungsi). Sebagai contoh untuk menghasilkan nilai terkecil di antara dua buah nilai real, maka definisinya berupa :

```
float minimum(float x, float y)
{
     if (x < y)
          return(x);
     else

                    return(y);
}
```

Perhatikan, di depan nama **minimum** diberikan tipe keluaran fungsi berupa *float*. Seluruh parameter sendiri juga didefinisikan dengan tipe *float*. Selengkapnya adalah sebagai berikut :

```
/* File program : minimum2.c */

#include <stdio.h>

float minimum (float, float);

main()
{
     float a, b, kecil;

     printf("Masukkan nilai a : ");
     scanf("%f", &a);
     printf("Masukkan nilai b : ");
     scanf("%f", &b);

     kecil = minimum(a, b);
     printf("\nBilangan terkecil antara %g dan %g adalah
          %g\n\n", a, b, kecil);
}

float minimum(float x, float y)
{
     if (x < y)
          return(x);
     else
          return(y);
}
```

Contoh eksekusi :

```
Masukkan nilai a = 5.5
Masukkan nilai b = 6.23
```

```
Bilangan terkecil antara 5 dan 6.23 adalah 5.5
```

Khusus untuk fungsi yang dirancang tanpa memberikan nilai keluaran (melainkan hanya menjalankan suatu tugas khusus) biasa didefinisikan dengan diawali kata kunci *void* (di depan nama fungsi). Sebagai contoh perhatikan program di bawah ini.

```
/* File program : void.c
Contoh fungsi tanpa nilai keluaran (pamakaian void) */

#include <stdio.h>

void info_program();          /* deklarasi fungsi */
main()
{
     info_program();          /* pemanggilan fungsi */
}

void info_program()           /* definisi fungsi */
{
     puts("================================");
     puts("Progam dibuat oleh Moh. Izzuddin  ");
     puts("Tanggal : 12 Juni 2001            ");
     puts("                                  ");
     puts("Selamat menggunakannya.......     ");
     puts("================================");
}
```

Contoh eksekusi :

```
================================
Progam dibuat oleh Moh. Izzuddin
Tanggal : 12 Juni 2001

Selamat menggunakannya.......
================================
```

Prototipe Fungsi

Prototipe fungsi digunakan untuk menjelaskan kepada kompiler mengenai :

- tipe keluaran fungsi
- jumlah parameter
- tipe dari masing-masing parameter.

Bagi kompiler, informasi dalam prototipe akan dipakai untuk memeriksa keabsahan (validitas) parameter dalam pemanggilan fungsi. Salah satu keuntungannya adalah, kompiler akan melakukan konversi seandainya antara tipe parameter dalam fungsi dan parameter saat pemanggilan fungsi tidak sama, atau akan menunjukan kesalahan bila jumlah parameter dalam definisi dan saat pemanggilan berbeda.

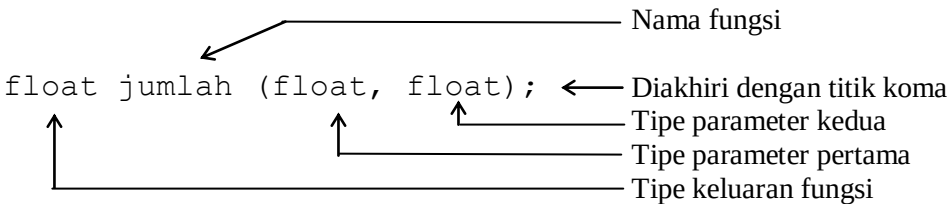
Contoh prototipe fungsi;

```
float jumlah (float x, float y);
```

atau

```
float jumlah (float, float);
```

Penjelasannya adalah sbb :



Gambar 5.4 Prototipe fungsi

Perhatikan contoh program di bawah ini.

```
/* File program : jumlah.c
contoh pemakaian prototipe fungsi */

#include <stdio.h>

float jumlah(float, float);        /* prototipe fungsi */

main()
{
     float a, b,c;

     printf("Masukkan nilai a : ");
     scanf("%f", &a);
     printf("Masukkan nilai b : ");
     scanf("%f", &b);

     c = jumlah(a, b);
     printf("\nHasil penjumlahan a + b = %g\n", c);
}
```

```
float jumlah(float x, float y)          /* definisi fungsi */
{
     return(x + y);
}
```

Contoh eksekusi :

```
Masukkan nilai a : 4.5
Masukkan nilai b : 7.65

Hasil penjumlahan a + b = 12.15
```

Untuk fungsi yang tidak memiliki argumen (contoh program **void.c**), maka deklarasinya adalah

```
void info_program(void);
```

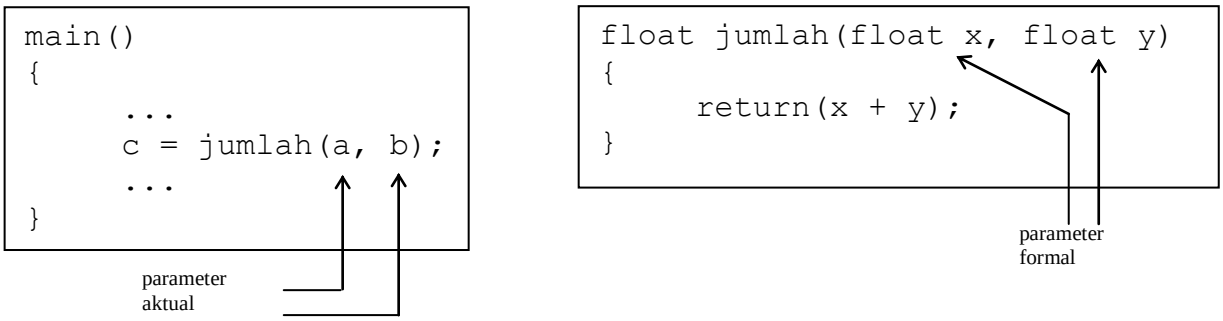
menyatakan bahwa **info_program()** tidak memiliki parameter

Catatan :

- Untuk fungsi-fungsi pustaka, prototipe dari fungsi-fungsi berada di file-file judulnya *(header file)*. Misalnya fungsi pustaka *printf()* dan *scanf()* prototipenya berada pada file dengan nama **stdio.h**
- Untuk fungsi pustaka pencantuman pada prototipe fungsi dapat dilakukan dengan menggunakan *preprocessor directive* **#include**.

Parameter Formal dan Parameter Aktual

Parameter formal adalah variabel yang ada pada daftar parameter dalam definisi fungsi. Pada contoh program di atas misalnya, maka dalam fungsi **jumlah()** variabel **x** dan **y** dinamakan sebagai parameter formal. Adapun parameter aktual adalah parameter (tidak selalu berupa variabel) yang dipakai dalam pemanggilan fungsi.



Gambar 5.5 Paramater formal dan parameter aktual

Pada pernyataan :

```
x = jumlah(a, b);
y = jumlah(20.1, 45.6);
```

a dan **b** merupakan parameter aktual dari fungsi **jumlah()** dalam hal ini parameter berupa variabel. Demikian juga **20.1** dan **45.6** adalah parameter aktual, dalam hal ini berupa konstanta. Bahkan bisa juga parameter aktual berupa ungkapan yang melibatkan operator, misalnya :

```
printf("%g\n", jumlah(2+3, 3+6));
```

ungkapan

Cara Melewatkan Parameter

Ada dua cara untuk melewatkan parameter kedalam fungsi, yaitu berupa ;

- Pemanggilan dengan nilai *(call by value)*
- Pemanggilan dengan pointer *(call by pointer)*

Pemanggilan dengan nilai merupakan cara yang dipakai untuk seluruh fungsi buatan yang telah dibahas didepan. Pada pemanggilan dengan nilai, nilai dari parameter aktual akan disalin ke parameter formal. Dengan cara ini nilai parameter aktual tidak bisa dirubah sekalipun nilai parameter formal berubah. Untuk lebih jelasnya lihat pada fungsi pada contoh berikut ini.;

```
#include <stdio.h>
void fungsi_nilai (int );
main()
{
    int a;
    a = 10;
    printf("nilai a sebelum fungsi = %d\n", a);
    fungsi_nilai (a);
    printf("nilai a setelah fungsi = %d\n", a);
}
void fungsi_nilai (int b)
{
   b = b + 5;
   printf ("nilai a di fungsi = %d\n",b);
}
```

Contoh Eksekusi:

Nilai a sebelum fungsi = 10;

Nilai a setelah fungsi = 10;

Nilai a di fungsi = 15;

Dengan menggunakan *call by pointer* maka program tersebut menjadi :

```
#include <stdio.h>
void fungsi_nilai (int *b );
main()
{
    int a;
    a = 10;
    printf("nilai a sebelum fungsi = %d\n", a);
    fungsi_nilai (&a);
    printf("nilai a setelah fungsi = %d\n", a);
}
void fungsi_nilai (int *b)
{
   *b = *b + 5;
   printf ("nilai a di fungsi = %d\n",*b);
}
```

Contoh Eksekusi:

Nilai a sebelum fungsi = 10;

Nilai a setelah fungsi = 15;

Nilai a di fungsi = 15;

Dari hasil eksekusi tersebut terlihat bahwa dengan menggunakan *call by pointer*, nilai pada parameter aktual bisa berubah.

C. TUGAS PENDAHULUAN

1. Buatlah program untuk menghitung luas dan keliling lingkaran dengan menggunakan fungsi:
   a. yang tidak memiliki input dan output
   b. yang memiliki input dan output
   c. yang memiliki input dan tidak memiliki output.

D. PERCOBAAN

1. Buatlah program untuk menjumlahkan n buah data, tetapi yang dijumlahkan hanya data genap dengan menggunakan fungsi.
2. Buatlah program untuk menjumlahkan n buah data kemudian hitunglah rata-ratanya dengan menggunakan fungsi.
3. Dengan menggunakan *call by pointer*, buatlah program untuk menukarkan dua buah bilangan
4. Dengan menggunakan fungsi, buatlah program untuk menentukan nilai terbesar dari 3 buah inputan

```
Tampilan:
Masukkan 3 buah bilangan
6 9 8
Nilai terbesar adalah 9
```

E. LAPORAN RESMI

1. Buatlah flowchart dari percobaan yang telah dilakukan.
2. Buatlah suatu fungsi `ganjil()` yang mengembalikan nilai 1 jika argumen yang diberikan adalah bilangan ganjil dan mengembalikan nilai 0 jika argumen tsb bukan bilangan ganjil.
3. Apa hasil eksekusi dari program berikut :

```
#include <stdio.h>

void ubah(int);

main()
{
int x;

printf("Masukkan nilai x : ");
scanf("%d", &x);
ubah(x);
printf("x = %d\n", x);
}

void ubah(int y)
{
y = 85;
}
```