# BAB V
# FUNGSI

**Tujuan :**

1. Memecah program dalam fungsi fungsi yang sederhana.
2. Menjelaskan tentang pemrograman terstruktur.
3. Mengetahui perbedaan antara variabel lokal, eksternal, statis dan register

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.

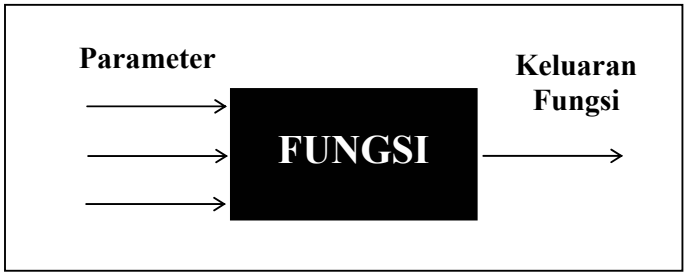
## 5.1 Dasar Fungsi

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 5.1 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 :

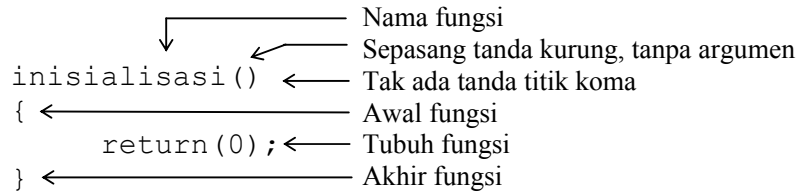
- **tipe-keluaran-fungsi**, dapat berupa salah satu tipe data C, misalnya *char* atau *int* . Kalau penentu tipe tidak disebutkan maka dianggap bertipe *int* (secara *default*).
- **tubuh fungsi** berisi deklarasi variabel (kalau ada) dan statemen-statemen yang akan melakukan tugas yang akan diberikan 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 5.2 Penjelasan definisi sebuah fungsi

**5.2 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);
```

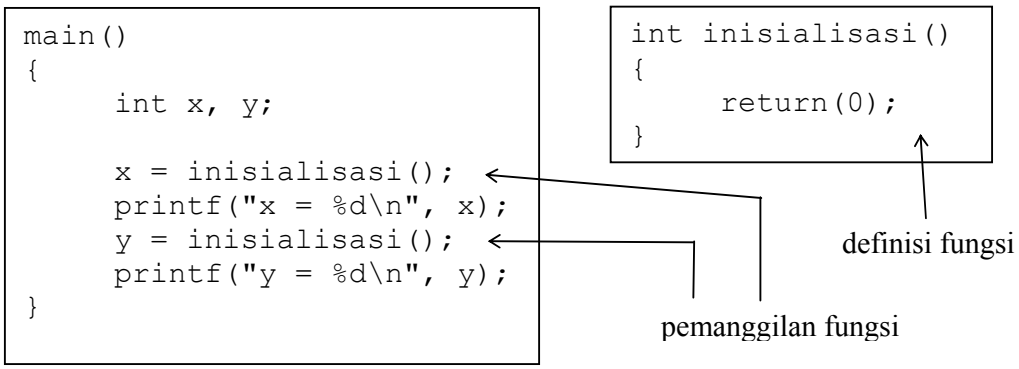
merupakan pernyataan untuk memberikan nilai keluaran fungsi berupa nol. Selengkapnya perhatikan program di bawah ini

```
/* 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 5.3 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
```

### 5.3 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.......
================================
```

### 5.4 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.

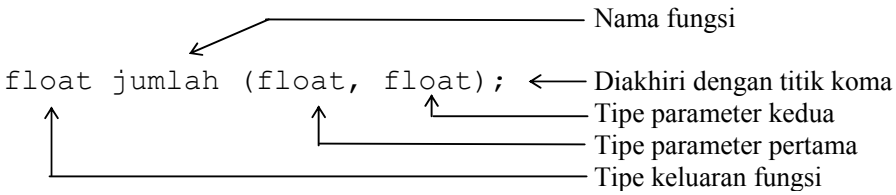
Contoh prototipe fungsi;

```
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);
```

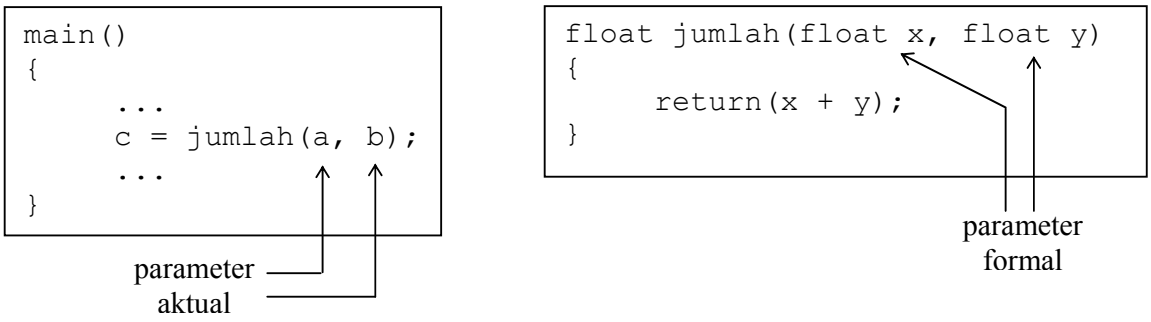
menyatakan bahwa **info_program()** tidak memiliki parameter

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**.

## 5.5 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.

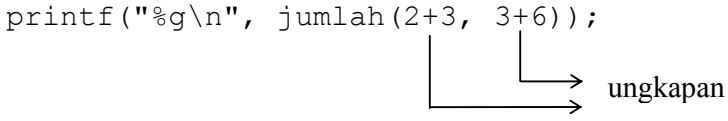



Gambar 5.5 Paramater formal dan parameter aktual

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 :



### 5.6 Cara Melewatkan Parameter

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

- Pemanggilan dengan nilai *(call by value)*
- Pemanggilan dengan referensi *(call by reference)*

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 **tukar()** pada contoh berikut ini.;

```
/* File program : tukar1.c
Untuk melihat pengaruh pemanggilan nilai pada fungsi untuk
penukaran dua data */

#include <stdio.h>

void tukar (int, int);

main()
{
     int a,b;

     a = 88;
     b = 77;

     printf("Nilai sebelum pemanggilan fungsi\n");
     printf("a = %d    b = %d\n", a, b);

     tukar(a,b);

     printf("\nNilai setelah pemanggilan fungsi\n");
     printf("a = %d    b = %d\n", a, b);
}

void tukar(int x, int y)
{
     int z;

     z = x;
     x = y;
     y = z;

     printf("\nNilai di akhir fungsi tukar()\n");
     printf("x = %d    y = %d\n", x, y);
```

```
}
```

**Contoh eksekusi :**

```
Nilai sebelum pemanggilan fungsi
a = 88    b = 77

Nilai di akhir fungsi tukar()
x = 77    y = 88

Nilai setelah pemanggilan fungsi
a = 88    b = 77
```

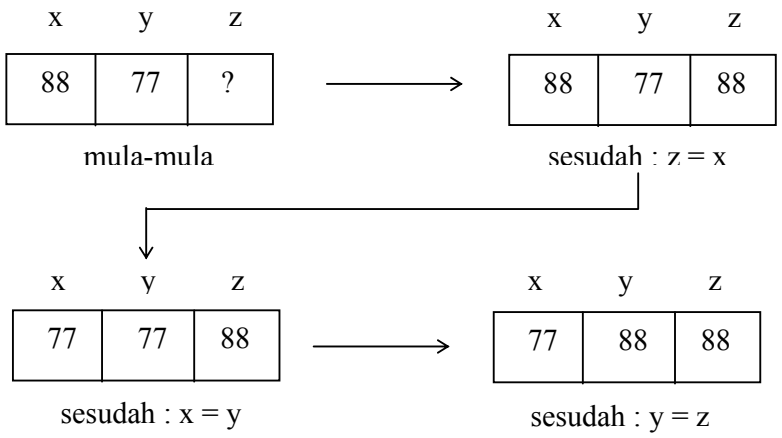
Tampak bahwa sekeluarnya dari pemanggilan fungsi **tukar()**, variabel **a** dan **b** (yang dilewatkan ke fungsi **tukar()** tidak berubah, walaupun pada fungsi **tukar()** telah terjadi penukaran antara parameter **x** dan **y** . Mengapa hal ini bisa terjadi ? Sebab **x** hanyalah salinan dari **a** dan **y** adalah salinan dari **b** (Lihat gambar 5.6 di bawah ini). Pada saat pemanggilan fungsi, maka :

- **x** bernilai 88 (nilai **a**)
- **y** bernilai 77 (nilai **b**)

Sesudah pernyataan-pernyataan berikut dijalankan, maka :

```
z = x;
x = y;
y = z;
```

**x** akan bernilai 77 dan **y** bernilai 88.




Gambar 5.6 Proses penukaran nilai

Gambar 5.6 menjelaskan bahwa **a** dan **b** tidak berubah. Yang berubah hanyalah parameter **x** dan **y**.

Pemanggilan dengan referensi *(call by reference)* merupakan upaya untuk melewatkan alamat dari suatu variabel ke dalam fungsi. Cara ini dapat dipakai untuk mengubah isi suatu variabel di luar fungsi dengan pelaksanaan pengubahan dilakukan di dalam fungsi. Sebagai contoh perhatikan program **tukar2.c** yang merupakan modifikasi dari **tukar1.c**. Perubahan yang pertama terletak dalam definisi fungsi, yang kini berupa

```
void tukar(int *px, int *py)
{
     int z;

     z = *px;
     *px = *py;
     *py = z;

     printf("\nNilai di akhir fungsi tukar()\n");
     printf("x = %d    y = %d\n", *px, *py);
}
```

Adapun perubahan dalam parameter aktualnya menjadi :

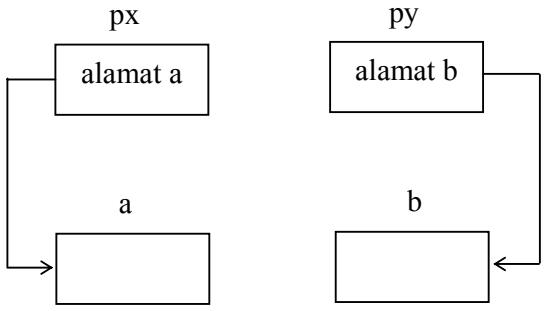
```
tukar(&a,&b);            /* alamat a dan alamat b */
```

Dalam deklarasi parameter

```
int *px, int *py
```

menyatakan bahwa **px** dan **py** adalah suatu variabel pointer. Yang dimaksudkan sebagai variabel pointer adalah suatu variabel yang menunjuk ke variabel lain. Lebih jelasnya, variabel pointer berisi alamat dari variabel lain.

Adapun pada pemanggilan fungsi, **&a** dan **&b** masing-masing berarti "alamat a" dan "alamat b". Dengan pemanggilan seperti ini, hubungan antara variabel pointer **px** dan **py** dengan variabel **a** dan **b** adalah seperti ditunjukkan pada gambar 5.7. Dalam hal ini, **px** dikatakan menunjuk variabel **a** dan **py** menunjuk variabel **b**.



Gambar 5.7
Variabel pointer **px** menunjuk variabel **a** dan variabel pointer **py** menunjuk variabel **b**

```
/* File program : tukar2.c
Untuk melihat pengaruh pemanggilan nilai pada fungsi untuk
penukaran dua data */

#include <stdio.h>

void tukar (int *px, int *py);     /* prototype fungsi */

main()
{
     int a,b;

     a = 88;
     b = 77;

     printf("Nilai sebelum pemanggilan fungsi\n");
     printf("a = %d    b = %d\n", a, b);

     tukar(&a,&b);            /* alamat a dan alamat b */

     printf("\nNilai setelah pemanggilan fungsi\n");
     printf("a = %d    b = %d\n", a, b);
}

void tukar(int *px, int *py)
{
     int z;

     z = *px;
     *px = *py;
     *py = z;

     printf("\nNilai di akhir fungsi tukar()\n");
     printf("x = %d    y = %d\n", *px, *py);
}
```

**Contoh eksekusi :**

```
Nilai sebelum pemanggilan fungsi
a = 88    b = 77
```

```
Nilai di akhir fungsi tukar()
x = 77    y = 88

Nilai setelah pemanggilan fungsi
a = 77    b = 88
```

---

Setelah **px** menunjuk **a** dan **py** menunjuk **b**, proses penukaran isi **a** dan **b** dilakukan dengan cara sebagai berikut :

```
z = *px;            /* 1 */
*px = *py;          /* 2 */
*py = z;            /* 3 */
```

Pertama variabel **z** diisi dengan nilai yang ditunjuk oleh **px**. Kedua, yang ditunjuk oleh **px** diisi dengan yang ditunjuk oleh **py** (berarti **a** diisi dengan **b**). Ketiga, yang ditunjuk oleh **py** diberi nilai **z**. Dengan melalui tiga pernyataan di atas, nilai **a** dab **b** dapat diubah di dalam fungsi.

Catatan : Pembahasan lebih lanjut mengenai pointer dapat dilihat pada bab VIII.

## 5.7 Penggolongan Variabel berdasarkan Kelas Penyimpanan

Suatu variabel, di samping dapat digolongkan berdasarkan jenis/tipe data juga dapat diklasifikasikan berdasarkan kelas penyimpanan *(storage class)*. Penggolongan berdasarkan kelas penyimpanan berupa :

- variabel lokal
- variabel eksternal
- variabel statis
- variabel register

### 5.7.1 Variabel Lokal

Variabel lokal adalah variabel yang dideklarasikan dalam fungsi, dengan sifat :

- secara otomatis diciptakan ketika fungsi dipanggil dan akan sirna (lenyap) ketika eksekusi terhadap fungsi berakhir.
- Hanya dikenal oleh fungsi tempat variabel tersebut dideklarasikan
- Tidak ada inisialisasi secara otomatis (saat variabel diciptakan, nilainya tak menentu).

Dalam banyak literatur, variabel lokal disebut juga dengan variabel otomatis. Variabel yang termasuk dalam golongan ini bisa dideklarasikan dengan menambahkan kata kuci *auto* di depan tipe-data variabel. Kata kunci ini bersifat opsional, biasanya disertakan sebagai penjelas saja. Contoh variabel lokal ditunjukkan pada gambar 5.8.

```
void fung_x(void)
{
    int x;
    .    ↑
    .    └──────────  x adalah variabel lokal bagi
    .                 fungsi fung_x()
    .
}
```

Gambar 5.8 Variabel lokal

Pada **fung_x()**, deklarasi

```
int x;
```

dapat ditulis menjadi

```
auto int x;
```

Penerapan variabel lokal yaitu bila variabel hanya dipakai oleh suatu fungsi (tidak dimaksudkan untuk dipakai oleh fungsi yang lain). Pada contoh berikut, antara variabel **i** dalam fungsi **main()** dan **fung_1()** tidak ada kaitannya, sebab masing-masing merupakan variabel lokal.

---

```
/* File program : lokal.c */
#include <stdio.h>

void fung_1(void);

main()
{
    int i = 20;

    fung_1();
    printf("nilai i di dalam main()   = %d\n", i);
}

void fung_1(void)
{
    int i = 11;
```

```
    printf("nilai i di dalam fung_1() = %d\n", i);
}
```

**Contoh eksekusi :**

```
nilai i di dalam fung_1() = 11
nilai i di dalam main()   = 20
```

---

### 5.7.2 Variabel Eksternal

Variabel eksternal merupakan variabel yang dideklarasikan di luar fungsi, dengan sifat :

- dapat diakses oleh semua fungsi
- kalau tak diberi nilai, secara otomatis diinisialisasi dengan nilai sama dengan nol.

Contoh variabel eksternal ada pda program **ekstern1.c** yaitu berupa variabel **i**. Pada pendeklarasian

```
int i = 273;
```

menyatakan bahwa **i** merupakan variabel eksternal dan diberi nilai awal sama denan 273. Nilai dari variabel **i** selanjutnya dapat diubah oleh fungsi **tambah()** maupun **main()**. Setiap fungsi **tambah()** dipanggil maka nilai **i** akan bertambah satu.

---

```
/* File program : ekstern1.c
Contoh program dengan variabel eksternal */

#include <stdio.h>

int i = 273;              /* variabel eksternal */

void tambah(void);

main()
{
     printf("Nilai awal i = %d\n", i);
     i += 7;
     printf("Nilai i kini = %d\n", i);
     tambah();
     printf("Nilai i kini = %d\n", i);
```

```
     tambah();
     printf("Nilai i kini = %d\n", i);
}

void tambah(void)
{
     i++;
}
```

**Contoh eksekusi :**

```
Nilai awal i = 273
Nilai i kini = 280
Nilai i kini = 281
Nilai i kini = 282
```

---

Pada contoh di atas, terlihat bahwa **i** hanya dideklarasikan di bagian atas program, dan tak dideklarasikan lagi dalam fungsi **main()** maupun **tambah()**. Oleh karena **i** merupakan variabel eksternal maka dapat digunakan oleh kedua fungsi tsb. Namun ada satu hal yang perlu diketahui, variabel eksternal haruslah dideklarasikan sebelum definisi fungsi yang akan mempergunakannya.

Untuk memperjelas bahwa suatu variabel dalam fungsi merupakan variabel eksternal, di dalam fungsi yang menggunakannya dapat mendeklarasikan variabel itu kembali dengan menambahkan kata kunci *extern* di depan tipe data variabel. Sebagai contoh, program **ekstern1.c** ditulis kembali menjadi seperti pada **ekstern2.c**.

---

```
/* File program : ekstern2.c
Contoh program yang menggunakan variabel eksternal dan
memakai kata kunci extern */

#include <stdio.h>

int i = 273;              /* variabel eksternal */

void tambah(void);

main()
{
     extern int i;        /* variabel eksternal */

     printf("Nilai awal i = %d\n", i);
     i += 7;
```

```
     printf("Nilai i kini = %d\n", i);
     tambah();
     printf("Nilai i kini = %d\n", i);
     tambah();
     printf("Nilai i kini = %d\n", i);
}

void tambah(void)
{
     extern int i;       /* variabel eksternal */

     i++;
}
```

**Contoh eksekusi :**

```
Nilai awal i = 273
Nilai i kini = 280
Nilai i kini = 281
Nilai i kini = 282
```

---

Kalau dalam suatu program terdapat suatu variabel eksternal, suatu fungsi bisa saja menggunakan nama variabel yang sama dengan variabel eksternal, namun diperlakukan sebagai variabel lokal. Untuk lebih jelasnya perhatikan contoh program di bawah ini.

---

```
/* File program : ekstern3.c
Contoh program yang menggunakan variabel eksternal dan
variabel lokal dengan nama yang sama */

#include <stdio.h>

int i = 273;                  /* variabel eksternal */

void tambah(void);

main()
{
     extern int i;            /* variabel eksternal */

     printf("Nilai awal i = %d\n", i);
     i += 7;
     printf("Nilai i kini = %d\n", i);
     tambah();
     printf("Nilai i kini = %d\n", i);
     tambah();
     printf("Nilai i kini = %d\n", i);
```

```
}

void tambah(void)
{
     int i;                    /* variabel lokal */

     i++;
}
```

**Contoh eksekusi :**

```
Nilai awal i = 273
Nilai i kini = 280
Nilai i kini = 280
Nilai i kini = 280
```

---

Pada program di atas, bagi fungsi **main() i** adalah variabel eksternal. Namun bagi fungsi **tambah()**, **i** merupakan variabel lokal, sebab pada fungsi ini **i** dideklarasikan tanpa kata kunci *extern*. Hal ini terlihat jelas dengan mengamati hasil eksekusi program. Pernyataan:

```
     i++;
```

Pada fungsi **tambah()** tidak mempengaruhi nilai **i** yang ditampilkan pada fungsi **main()** (bandingkan dengan hasil eksekusi pada **ekstern2.c**).

### 5.7.3 Variabel Statis

Variabel statis dapat berupa variabel internal (didefinisikan di dalam fungsi) maupun variabel eksternal. Sifat variabel ini :

- Kalau variabel statis bersifat internal, maka variabel hanya dikenal oleh fungsi tempat variabel dideklarasikan
- Kalau variabel statis bersifat eksternal, maka variabel dapat dipergunakan oleh semua fungsi yang terletak pada file yang sama, tempat variabel statis dideklarasikan
- Berbeda dengan variabel lokal, variabel statis tidak akan hilang sekeluarnya dari fungsi (nilai pada variabel akan tetap diingat).
- Inisialisasi akan dilakukan hanya sekali, yaitu saat fungsi dipanggil yang pertama kali. Kalau tak ada inisialisasi oleh pemrogram secara otomatis akan diberi nilai awal nol

Variabel statis diperoleh dengan menambahkan kata kunci *static* di depan tipe data variabel. Sebagai contoh perhatikan program di bawah ini.

```
/* File program : statis.c
Contoh variabel statis */

#include <stdio.h>

void fung_y(void);

main()
{
    int y = 20;

    fung_y();
    fung_y();
    printf("Nilai y dalam main()   = %d\n", y);
}

void fung_y(void)
{
    static int y;

    y++;
    printf("Nilai y dalam fung_y() = %d\n", y);
}
```

**Contoh eksekusi :**

```
Nilai y dalam fung_y() = 1
Nilai y dalam fung_y() = 2
Nilai y dalam main()   = 20
```

### 5.7.4 Variabel Register

Variabel register adalah variabel yang nilainya disimpan dalam register dan bukan dalam memori RAM. Variabel yang seperti ini hanya bisa diterapkan pada variabel yang lokal atau parameter formal, yang bertipe *char* atau *int*. Variabel register biasa diterapkan pada variabel yang digunakan sebagai pengendali *loop*. Tujuannya untuk mempercepat proses dalam *loop*. Sebab variabel yang dioperasikan pada register memiliki kecepatan

yang jauh lebih tinggi daripada variabel yang diletakkan pada RAM. Contoh pemakaiannya bisa dilihat pada program di bawah ini.

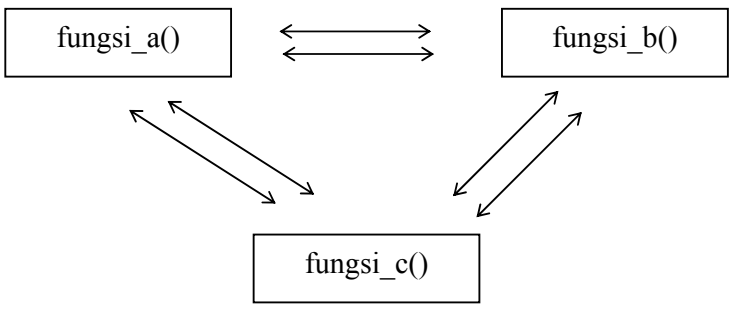
```
/* File program : var_reg.c
Contoh variabel register */

#include <stdio.h>

main()
{
    register int i;                 /* variabel register */
    int jumlah = 0;

    for(i = 1; i <= 100; i++)
        jumlah = jumlah + i;

    printf("1 + 2 + 3 + ... + 100 = %d\n", jumlah);
}
```

**Contoh eksekusi :**

```
1 + 2 + 3 + ... + 100 = 5050
```

### 5.8 Menciptakan Sejumlah Fungsi

Pada C, semua fungsi bersifat sederajat. Suatu fungsi tidak dapat didefinisikan di dalam fungsi yang lain. Akan tetapi suatu fungsi diperbolehkan memanggil fungsi yang lain, dan tidak tergantung kepada peletakan definisi fungsi pada program. Komunikasi antara fungsi dalam C ditunjukkan dalam gambar 5.9. Gambar tersebut menjelaskan kalau suatu fungsi katakanlah **fungsi_a()** memanggil **fungsi_b()**, maka bisa saja **fungsi_b()** memanggil **fungsi_a()**. Contoh program yang melibatkan fungsi yang memanggil fungsi yang lain ada pada program **kom_fung.c**, yaitu **fungsi_1()** dipanggil dalam **main()**, sedangkan **fungsi_2()** dipanggil oleh **fungsi_1()**.



Gambar 5.9 Komunikasi antar fungsi dalam C

```
/* File program : kom_fung.c
contoh fungsi yang memanggil fungsi yang lain */

#include <stdio.h>

void fungsi_1(void);
void fungsi_2(void);

main()
{
     fungsi_1();
}

void fungsi_1()
{
     puts("fungsi 1 dijalankan");
     fungsi_2();
}

void fungsi_2()
{
     puts("fungsi 2 dijalankan");
}
```

**Contoh eksekusi :**

```
fungsi 1 dijalankan
fungsi 2 dijalankan
```

## 5.9 Rekursi

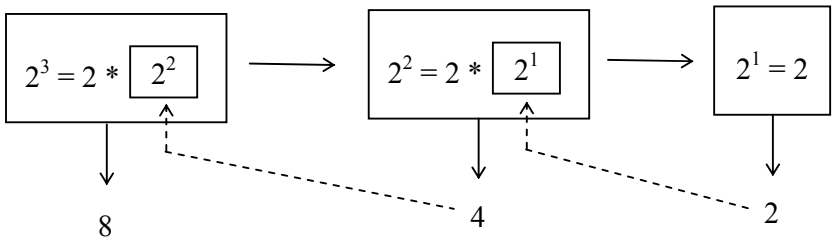
Fungsi dalam C dapat dipakai secara rekursi, dalam artian suatu fungsi dapat memanggil dirinya sendiri. Sebagai contoh penerapan fungsi rekursi yaitu untuk menghitung nilai

$x^n$

dengan n berupa bilagnan bulat positif. Solusi dari persoalan ini dapat berupa :

- Jika n = 1, maka $x^n = x$
- Selain itu maka $x^n = x * x^{n-1}$

Misalnya x = 2 dan n = 3, proses pemecahannya seperti diuraikan pada gambar 5.10.




Gambar 5.10 Pemecahan secara rekursi

Penuangan dalam bentuk program

```
/* File program : faktor.c
Contoh penerapan rekursi untuk memperoleh nilai factorial */

#include <stdio.h>

int faktorial(int);

main()
{
    int x;

     puts("MENCARI FAKTORIAL DARI X!");
     printf("Masukkan nilai x (bulat positif) : ");
     scanf("%d", &x);

     printf("Faktorial dari %d = %d\n", x, faktorial(x));
}

int faktorial(int m)
{
     if(m == 1)
          return(1);
     else
          return(m * faktorial(m-1));
}
```

**Contoh eksekusi :**

```
MENCARI FAKTORIAL DARI X!
Masukkan nilai x (bulat positif) : 4
Faktorial dari 4 = 24
```

Rekursi jarang dipakai, di antaranya disebabkan :

- Biasanya rekursi akan menjadikan fungsi sulit dimengerti. Hanya cocok untuk persoalan tertentu saja (misalnya pada *binary tree* atau pohon biner). Untuk fungsi rekursi pada program faktor.c di atas misalnya, akan lebih mudah dipahami kalau ditulis menjadi :

```
int faktorial(int m)
{
    int i, fak;

    fak = 1;
    for(i = 1; i <= m; i++)
        fak = fak * i;
    return(fak);
}
```

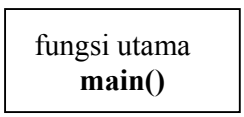
- Memerlukan *stack* dengan ukuran yang lebih besar. Sebab setiap kali fungsi dipanggil, variabel lokal dan parameter formal akan ditempatkan ke stack dan adakalanya akan menyebabkan *stack* tak cukup lagi *(stack overflow)*.

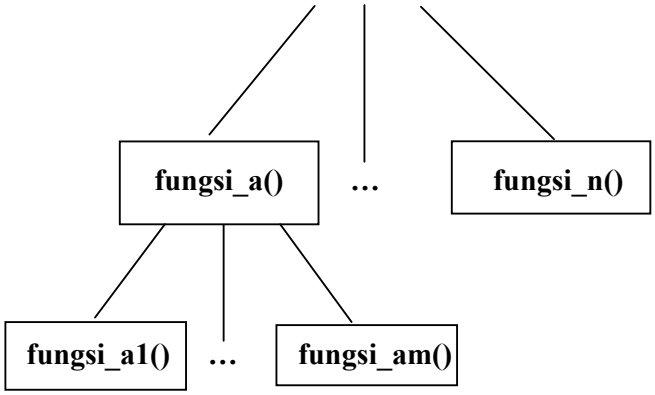
## 5.10 Pengenalan Konsep Pemrograman Terstruktur

Fungsi sangat bermanfaat untuk membuat program yang terstruktur. Suatu program yang terstruktur dikembangkan dengan menggunakan *"top-down design"* (rancang atas bawah). Pada C suatu program disusun dari sejumlah fungsi dengan tugas tertentu. Selanjutnya masing masing fungsi dipecah-pecah lagi menjadi fungsi yang lebih kecil. Pembuatan program dengan cara ini akan memudahkan dalam pencarian kesalahan ataupun dalam hal pengembangan dan tentu saja mudah dipahami/ dipelajari.

Dalam bentuk diagram, model suatu program C yang terstruktur adalah seperti yang tertera pada bagan berikut ini. Namun sekali lagi perlu diketahui, bahwa pada C semua fungsi sebenarnya berkedudukan sederajat.

Fungsi **main()** terdiri dari **fungsi_a()** sampai dengan **fungsi_n()**, menegaskan bahwa dalam program fungsi **main()** akan memanggil **fungsi_a()** sampai dengan **fungsi_n()**. Adapun fungsi-fungsi yang dipanggil oleh fungsi **main()** juga bisa memanggil fungsi-fungsi yang lain.






Gambar 5.11 Model terstruktur Program C

**Kesimpulan**

- Fungsi digunakan untuk memecah program yang besar menjadi program-program kecil sesuai dengan fungsi masing-masing.
- Fungsi bisa memberikan nilai balik dan bisa tanpa memberikan nilai balik kepada fungsi yang memanggilnya.
- Fungsi yang memberikan nilai balik harus memiliki tipe dan ditulis didepan nama fungsi.
- Bila fungsi tidak memberikan nilai balik maka fungsi tersebut bertipe "void " dan ditulis didepan nama fungsi.

**Latihan :**

**Buatlah potongan program untuk soal-soal di bawah ini**

1. Buatlah sebuah fungsi yang berfungsi untuk menampilkan sebuah string (di layar) = "`Pilihan Menu`" (misalkan nama fungsinya = **menu**). Fungsi tersebut tidak memiliki nilai kembalian *(return value)* dan juga tidak menerima parameter masukan apapun.

2. Tulislah prototipe fungsi untuk fungsi pada soal nomor 1 di atas.

3. Buatlah sebuah fungsi (misalkan nama fungsinya = **cetak**) yang berfungsi untuk menampilkan sebuah string (di layar). Fungsi tersebut tidak memiliki nilai kembalian *(return value),* tetapi menerima parameter masukan berupa string yang akan dicetak (catatan : string merupakan array karakter).

4. Tulislah prototipe fungsi untuk fungsi pada soal nomor 3 di atas.

5. Buatlah sebuah fungsi (misalkan nama fungsinya = **total**) yang berfungsi untuk menjumlah total nilai dari array integer yang dikirim sebagai parameter masukan fungsi tsb. Fungsi tersebut memberikan nilai kembalian *(return value)* bertipe integer yang berisi total hasil perhitungannya. Dalam hal ini fungsi tsb memiliki 2 parameter masukan berupa array integer dan sebuah variabel integer yang menunjukkan jumlah elemen dari array tsb.

6. Tulislah prototipe fungsi untuk fungsi pada soal nomor 5 di atas.