PRAKTIKUM 8

FUNGSI 2

A. Tujuan

1. Mengetahui perbedaan antara variabel lokal, eksternal, dan variabel statis
2. Memahami dalam menciptakan sejumlah fngsi

B. DASAR TEORI

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 :

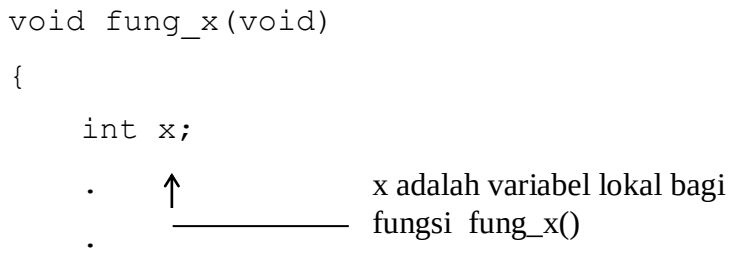
- variabel lokal
- variabel eksternal
- variabel statis
- variabel register

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 9.1.



Gambar 9.1 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
```

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**).

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.

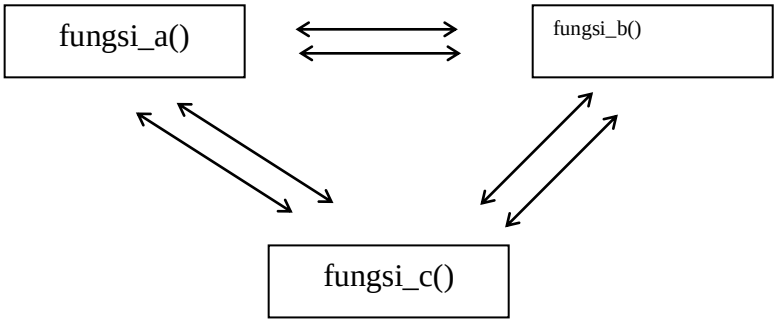
```
/* File program : statis.c
Contoh variabel statis */

#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
```

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 9.2 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 :

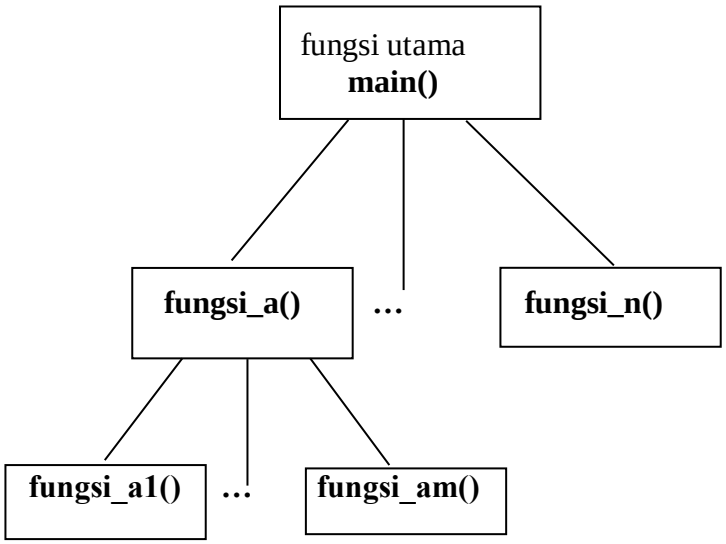
```
fungsi 1 dijalankan
fungsi 2 dijalankan
```

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 9.4 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.

C. TUGAS PENDAHULUAN

1. Apa hasil dari program dibawah ini, dan berikan penjelasan:

```
main()
{
      int total,i;
      for(i=0;i<5;i++)
      total=total+i;
      printf("jadi nilainya adalah %d",total);
}
```

D. PERCOBAAN

1. Trace secara manual semua program di bawah ini baris per barisnya, dan tampilkan nilai semua variabel pada setiap baris prosesnya. Selain itu, tebaklah tampilkan keluaran programnya

a)

```
int OddEvenTest(int);

main()
{
  int a, hasil;

  a = 5;
  hasil = OddEvenTest(a);
  printf("a=%d; hasil=%d\n",a,hasil);
}

int OddEvenTest(int b)
{
  int a;

  a = b % 2;
  return a;
}
```

Gambarkan hubungan antara bagian program yang memanggil fungsi, parameter actual, parameter formal dan hasil balik dari fungsi

b)

```
void demo(void);

main()
{
   int i=0;

   while(i < 3) {
        demo();
        i++;
   }
}

void demo(void)
{
   auto int var_auto = 0;
   static int var_static = 0;

   printf("auto = %d, static = %d\n",
                 var_auto, var_static);
   var_auto++;
   var_static++;
}
```

Apa yang terjadi bila
- var_auto++
- var_static++

diletakkan sebelum fungsi printf( )

c)

```
void fung_a(void);
void fung_b(void);

int x = 20;

main()
{
   x += 2;
  printf("\nNilai x dalam main() = %d\n\n",    x);
   fung_a();
  fung_a();
}

void fung_a(void)
{
   static x = 5;

   x++;
   printf("Nilai x dalam fung_a() = %d\n", x);
   fung_b();
}

void fung_b(void)
{
  int x=8;
   x--;
   printf("Nilai x dalam fung_b() = %d\n", x);
}
```

2. Definisikanlah fungsi-fungsi sebagai berikut :

- Fungsi f_to_i() untuk mengubah ukuran dari satuan kaki *(feet)* ke inci
- Fungsi i_to_cm() untuk mengubah ukuran dari satuan inci ke centimetre
- Fungsi c_to_m()untuk mengubah ukuran dari satuan centimeter ke meter

Dalam main() mintalah masukan ukuran dalam satuan kaki *(feet)* kemudian lakukan konversi sampai mendapatkan keluaran berupa ukuran dalam meter. Tentukan jumlah dan tipe parameter dan return value yang dibutuhkan

Keterangan : 1 kaki = 12 inchi, 1 inchi = 2.54 cm, 100 cm = 1 meter

D. LAPORAN RESMI

1. Buatlah flowchart dari seluruh percobaan yang telah dilakukan.