BAB 6
FUNGSI

TUJUAN PRAKTIKUM

1. Praktikan mengerti dan dapat menggunakan fungsi dalam program sederhana.
2. Praktikan dapat membedakan antara variabel lokal, variabel global, register & variabel statik
3. Praktikan dapat membedakan pengiriman parameter secara nilai dan secara acuan.
4. Praktikan dapat membuat program sederhana dengan fungsi rekursif.

TEORI PENUNJANG

6.1 Pengertian Fungsi

Fungsi merupakan suatu bagian dari program yang dimaksudkan untuk mengerjakan suatu tugas tertentu dan letaknya terpisah dari program yang memanggilnya. Fungsi merupakan elemen utama dalam bahasa C karena bahasa C sendiri terbentuk dari kumpulan fungsi-fungsi. Dalam setiap program bahasa C, minimal terdapat satu fungsi yaitu fungsi main(). Fungsi banyak diterapkan dalam program-program C yang terstruktur. Keuntungan penggunaan fungsi dalam program yaitu program akan memiliki struktur yang jelas (mempunyai readability yang tinggi) dan juga akan menghindari penulisan bagian program yang sama.

Dalam bahasa C fungsi dapat dibagi menjadi dua, yaitu fungsi pustaka atau fungsi yang telah tersedia dalam Turbo C dan fungsi yang didefinisikan atau dibuat oleh programmer.

6.2 Mendefinisikan Fungsi

Suatu fungsi secara umum mempunyai dua buah komponen utama, yaitu definisi fungsi dan tubuh fungsi. Definisi fungsi berisi dengan tipe dari fungsi, nama dari fungsi dan argumen-argumennya jika digunakan. Tubuh fungsi berisi dengan statemen-statemen yang akan melakukan tugas yang akan diberikan kepada fungsi yang bersangkutan. Bentuk umum dari fungsi :

tipe_fungsi nama-fungsi(argumen1,argumen2,...) → *Definisi fungsi*

```
{
.....
.....
.....
}
```

→ *Badan fungsi*

Definisi fungsi ditulis sebelum tubuh fungsi tanpa diakhiri dengan titik koma. Tipe di definisi fungsi menunjukkan tipe dari fungsi. Tipe dari fungsi tergantung dari tipe data hasil balik yang akan diberikan oleh fungsi. Misalnya hasil balik dari fungsi berupa nilai numerik pecahan, maka tipe dari fungsi dapat dibuat float atau double atau long double tergantung pada ketepatan yang diinginkan. Jika fungsi tidak memberikan nilai balik, maka tipenya adalah void. Tipe dari fungsi ini tidak dapat ditulis jika tipenya adalah char atau int. Jika suatu fungsi didefinisikan tanpa menggunakan tipenya, maka akan dianggap bertipe integer.

Contoh definisi fungsi dengan menggunakan argumen atau parameter :

```
int Fungsi_Ku (float A, int B, char C)
```

Hal-hal yang perlu diperhatikan dalam penggunaan fungsi:

- Kalau tipe fungsi tidak disebutkan, maka akan dianggap sebagai fungis dengan nilai keluaran bertipe integer.
- Untuk fungsi yang memiliki keluaran bertipe bukan integer, maka diperlukan pendefinisian penentu tipe fungsi.
- Untuk fungsi yang tidak mempunyai nilai keluaran maka dimasukkan ke dalam tipe void

- Pernyataan yang diberikan untuk memberikan nilai akhir fungsi berupa pernyataan return.
- Suatu fungsi dapat menghasilkan nilai balik bagi fungsi pemanggilnya.

6.2.1 Deklarasi Fungsi

Suatu fungsi yang memberikan hasil balik selain tipe integer perlu dideklarasikan sebelum digunakan. Deklarasi fungsi ditulis sebelum fungsi tersebut digunakan.

Bentuk umum dari deklarasi fungsi :

tipe nama_fungsi(argumen1, argumen2, ...);

Jika bagian dari program yang menggunakan fungsi diletakkan sebelum definisi fungsi, maka deklarasi fungsi diperlukan. Akan tetapi jika bagian program yang menggunakan fungsi terletak setelah definisi dari fungsi, maka deklarasi fungsi boleh tidak dapat dituliskan. Jika suatu fungsi memberikan hasil balik, maka nilai hasil balik yang diberikan oleh fungsi dapat dilakukan oleh statement **return** yang diikuti oleh nilai hasil baliknya yang ditulis tanda kurung. Contoh : return(F);

Contoh program :

```
#include < iostream.h >
double Absolut ( double X );/* prototype fungsi Absolut
*/
main()
{
float Nilai;
Nilai = -123.45;
cout << Nilai << “Nilai mutlaknya adalah “ << Absolut (
Nilai );
}
/* --- Fungsi untuk memutlakkan nilai negatif --- */
double Absolut ( double X )/* definisi fungsi */
{
if ( X < 0 ) X = -X;
return ( X );
}
```

Output :

```
-123.45 nilai mutlaknya adalah 123.45
```

6.2.2 Parameter Formal dan Parameter Aktual

- Parameter Formal adalah variabel yang ada pada daftar parameter dalam definisi fungsi.
- Parameter Aktual adalah variabel (parameter) yang dipakai dalam pemanggilan fungsi.
- Dalam contoh program pertambahan di atas parameter formal terdapat pada pendefinisian fungsi :

```
float tambah(float x, float y) //parameter
formal

{ return (a+b);
}
Sedangkan parameter aktual terdapat pada
pemanggilan fungsi :
void main()
{ ..............
  ..............
  c = tambah(a, b); //parameter aktual
  ..............
}
```

6.2.3 Ruang Lingkup Variabel

Terdapat tiga macam bentuk variabel yang mempunyai ruang lingkup berbeda, yaitu variabel lokal, variabel global, dan variabel statik.

6.2.3.1 Variabel Lokal

Merupakan variabel yang namanya dan nilainya hanya dikenal di suatu blok statemen tertentu saja atau di dalam suatu fungsi. Variabel lokal akan dihapus dari memori jika proses sudah meninggalkan blok statemen letak variabel lokalnya.

Sifat-sifat variabel lokal :

- Secara otomatis akan diciptakan ketika fungsi dipanggil dan akan lenyap ketika proses eksekusi terhadap fungsi berakhir.
- Hanya dikenal oleh fungsi tempat variabel dideklarasikan
- Tidak ada inisialisasi secara otomatis (saat variabel diciptakan nilainya random).
- Dideklarasikan dengan menambahkan kata “auto” (opsional).

Contoh program :

```
#include <stdio.h>
float Tambah(float A, float B);
main()
{
float Hasil;
Hasil=Tambah(2,3);
printf(“%f\n”,Hasil);
}
/*-------Fungsi menghitung pertambahan -------------
*/
float Tambah(float A,float B)
{
float C;
                        C = 1000;
}
return(C);
}
```

Output :

```
5.000000
```

Fungsi Tambah( ) akan menghasilkan nilai 5, yaitu hasil dari nilai 2 ditambah dengan nilai 3. Di dalam fungsi ini, terdapat suatu blok statemen yang merubah nilai variabel C dengan nilai 1000. Karena variabel C ini juga dideklarasikan sendiri, maka juga bersifat lokal untuk blok statemen tersebut dan dianggap berbeda dengan variabel C yang dideklarasikan di luar blok statemennya.

6.2.3.2 Variabel Global

Variabel global (eksternal) adalah variabel yang dideklarasikan di luar fungsi.

Sifat-sifat variabel global :

- Dikenal (dapat diakses) oleh semua fungsi.
- Jika tidak diberi nilai awal secara otomatis berisi nilai nol.
- Dideklarasikan dengan menambahkan kata “extern” (opsional).

Contoh Program :

```
#include “stdio.h”
#include “conio.h”
void tampil(void);
int i = 25; /* variabel global */
void main()
{ clrscr();
printf(“Nilai variabel i dalam fungsi main() adalah
%i\n\n”, i);
tampil();
i = i * 4; /* nilai i yang dikali 4 adalah 25
(global) bukan 10 */
printf(“\nNilai variabel i dalam fungsi main()
sekarang adalah %i\n\n”, i);
getch();
}
void tampil(void)
{ int i = 10; /* variabel lokal */
printf(“Nilai variabel i dalam fungsi tampil()
adalah %i\n\n”, i);
}
```

6.2.3.3 Variabel Register

Variabel Register adalah variabel yang nilainya disimpan dalam resister dan bukan dalam memori RAM.

Sifat-sifat variabel register :

- Hanya dapat diterapkan pada variabel lokal yang bertipe int dan char.
- Digunakan untuk mengendalikan proses perulangan (looping).

- Proses perulangan akan lebih cepat karena variabel register memiliki kecepatan yang lebih tinggi dibandingkan variabel biasa.
- Dideklarasikan dengan menambahkan kata “register”.

Contoh Program :

```
#include “stdio.h”
#include “conio.h”
void main()
{ register int x; /* variabel register */
int jumlah;
clrscr();
for(i=1; i<=100; i++)
jumlah = jumlah + I;
printf(“1+2+3+...+100 = %i\n”, jumlah);

getch();
```

6.2.3.4 Variabel Statik

Variabel statis adalah variabel yang nilainya tetap dan bisa berupa variabel lokal (internal) dan variabel global (eksternal).

Sifat-sifat variabel statis :

- Jika bersifat internal (lokal), maka variabel hanya dikenal oleh fungsi tempat variabel dideklarasikan.
- Jika bersifat eksternal (global), maka variabel dapat dipergunakan oleh semua fungsi yang terletak pada program yang sama.
- Nilai variabel statis tidak akan hilang walau eksekusi terhadap fungsi telah berakhir.
- Inisialisasi hanya perlu dilakukan sekali saja, yaitu pada saat fungsi dipanggil pertama kali.
- Jika tidak diberi nilai awal secara otomatis berisi nilai nol.
- Dideklarasikan dengan menambahkan kata “static”.

6.3 Pengiriman Parameter

6.3.1 Pengiriman Parameter Secara Nilai (*Call by Value*)

- Yang dikirim adalah nilai dari datanya, bukan alamat memori letak datanya.
- Perubahan nilai di fungsi tidak akan merubah nilai asli di bagian program yang memanggil fungsi walaupun keduanya menggunakan nama variabel yang sama.
- Merupakan pengiriman searah, yaitu dari bagian program yang memanggil fungsi ke fungsi yang dipanggil.

Contoh program :

```
#include <stdio.h>
void Secara_Nilai(float A, float B, char C);
main()
{
char C='a';
float A = 25, *Alamat_A;
Alamat_A=&A;
Secara_Nilai(A,A/3,C);
printf("Di fungsi utama setelah memanggil fungsi
Secara_Nilai:\n");
printf("Nilai A adalah %f di alamat %p\n", A,
Alamat_A);
printf("Nilai A/3 adalah %f\n", A/3);
printf("Nilai karakter C adalah %c\n", C);
}
void Secara_Nilai(float A, float B, char C);
{
float *Alamat_A;
Alamat_A=&A;
A=7;
printf("Di fungsi Secara_Nilai:\n");
printf("Nilai A adalah 5f di alamat
%p\n",A,Alamat_A);
printf("Nilai B adalah %f\n",B);
printf("Nilai karakter C adalah %c\n\n",C);
}
```

Output :

```
Di fungsi Secara_Nilai:
Nilai A adalah 7.000000 di alamat FFCA
Nilai B adalah 8.333333
```

```
Nilai karakter C adalah a
Di fungsi utama setelah memanggil Secara_Nilai;
Nilai A adalah 25.000000 di alamat FFD8
Nilai A/3 adalah 8.333333
Nilai karakter C adalah a
```

Keterangan :

- Parameter-parameter nyata yang dikirimkan adalah datanya, yaitu untuk variabel A, A/3, dan C sebagai berikut :

```
Secara_Nilai(A, A/3, C);
```

- Variabel A dan C menempati memori yang berbeda untuk fungsi utama yang memanggil fungsi dan yang digunakan di fungsi `Secara_Nilai()`. Untuk fungsi utama, variabel A disimpan di alamat FFD8, sedangkan di fungsi `Secara_Nilai()` nilai variabel A disimpan dalam alamat memori FFCA.
- Perubahan nilai variabel A di fungsi `Secara_Nilai()` menjadi bernilai 7 tidak merubah nilai variabel A ini di fungsi utama yang akan tetap bernilai 25.
- Pengiriman suatu nilai merupakan pengiriman satu arah sebagai berikut :

```
Secara_Nilai(A, A/3, C);


void Secara_Nilai(float A,float B, Char C)
```

- Pengiriman secara nilai dapat mengirimkan suatu ungkapan, yaitu A/3.

6.3.2 Pengiriman parameter secara acuan (*Call by reference*)

- Yang dikirim adalah alamat memori letak datanya, bukan nilai dari datanya.
- Perubahan nilai di fungsi akan merubah nilai asli di bagian program yang memanggil fungsi.
- Merupakan pengiriman dua arah, yaitu dari bagian program yang memanggil fungsi ke fungsi yang dipanggil dan sebaliknya.

Contoh di bawah ini akan menunjukkan bahwa perubahan nilai di fungsi untuk parameter yang dikirimkan secara acuan akan merubah juga nilai di bagian program yang memanggilnya. Karena sifat dua arah ini, maka pengiriman parameter secara acuan dapat juga dipakai sebagai pengganti hasil balik dari fungsi. Di contoh ini, parameter nyata C akan mengirimkan alamatnya ke fungsi Tambah( ). Setelah proses di Tambah( ) selesai, alamat ini akan berisi dengan hasil proses fungsi, sehingga dapat digunakan dibagian program yang memanggilnya sebagai hasil yang diperoleh dari fungsi yang dipanggil.

```
#include<stdio.h>
void Tambah(int *A, int *B, int *C);
main()
{
int A=, B=3, C;
Tambah(&A, &B, &C);
printf("%d ditambah %d adalah %d\n", A, B, C);
}

/*---------------Fungsi Acuan -----------------*/
void Tambah(int *A, int *B,int *C)
{
C=*A+*B;
return;
}
```

Output :

```
2 ditambah 3 adalah 5
```

6.3.3 Pengiriman Parameter Berupa Larik

Pengiriman parameter berupa larik (yang dibahas larik dimensi satu) sebenarnya merupakan pengiriman secara acuan, karena sebenarnya yang dikirimkan adalah alamat dari elemen pertama lariknya, bukan seluruh nilai-nilai elemennya. Alamat elemen pertama dari larik dapat ditunjukkan oleh nama lariknya yang tidak

ditulis dalam indeksnya. Bentuk ini akan tampak sebagai argumen di parameter nyata untuk bagian yang memanggil fungsi.

Contoh :

```
#include<stdio.h>
void Cetak_Mundur(char S[]);
main()
{
       char S[]=”Ini string”;
       Cetak_Mundur(S);
}
void Cetak_Mundur(char S[])
{
       int I,N;
       for(N=0; S[N];N++);
       for (I=N-1;I>=0;I--) printf(“%c”,S[I]);
       printf(“\n”);
}
```

Output :

```
gnirtS ialiN inI
```

6.4 Rekursi

Rekursi adalah suatu proses dari fungsi yang memanggil dirinya sendiri secara berulang-ulang.

Contoh program :

```
#include < stdio.h >
long int Fak_Rekursif ( int N );/* prototype fungsi */
main()
{
  int N ;

  N = 5;
  printf(“%d faktorial = %ld\n”, N, Fak_Rekursif(N));
}
long int Fak_Rekursif ( int N )/* definisi fungsi */
{
  long int F;
  if ( N <= 1 ) return( 1 ) ;
  else
  {
```

```
        F = N * Fak_Rekursif( N – 1);
       return(F);
     }
  }
```

Output :

```
5 faktorial = 120
```

Penjelasan :

- Fungsi utama memanggil fungsi `Fak_Rekursif` dengan mengirimkan nilai 5 untuk parameter nyata N, yang maksudnya akan dilakukan perhitungan sebanyak 5 faktorial.
- Jika nilai dari N pertama kali yang diberikan oleh fungsi utama bernilai kurang atau sama dengan satu, maka hasil faktorial yang akan diberikan adalah bernilai 1.
- Jika N pertama kali yang diberikan oleh fungsi utama lebih besar dari 1,maka proses rekursi akan dilakukan. Misalnya nilai N adalah 5, maka proses rekursi yang pertama adalah :

  ```
  F=5*Fak_Rekursif(4);
  ```

  Proses ini akan memanggil kembali fungsi dirinya sendiri dengan mengirimkan nilai 4 sebagai nilai N yang baru. Karena nilai N masih lebih besar dari 1, maka proses rekursi yang kedua akan dilakukan dengan hasil adalah `4*Fak_Rekursif(3)`. Begitu selanjutnya sampai nilai N adalah 1. Untuk nilai N =1 ini, statemen `return(1)` akan mengembalikan proses ke bagian yang memanggilnya, yaitu statemen setelah statemen `F=5*Fak_Rekursif(N-1)`. Statemen `return(F)` kemudian baru akan mengembalikan proses ke fungsi utama.