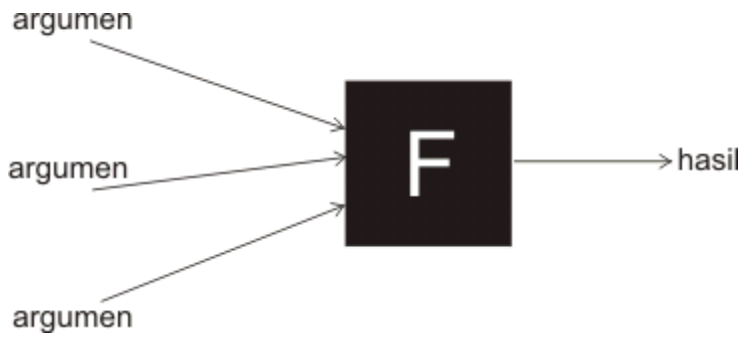
Bab 10
Fungsi & Prosedur

Fungsi adalah sekumpulah perintah komputasi. Sebuah fungsi dapat menerima satu atau lebih argumen, lalu mengembalikan sebuah nilai.

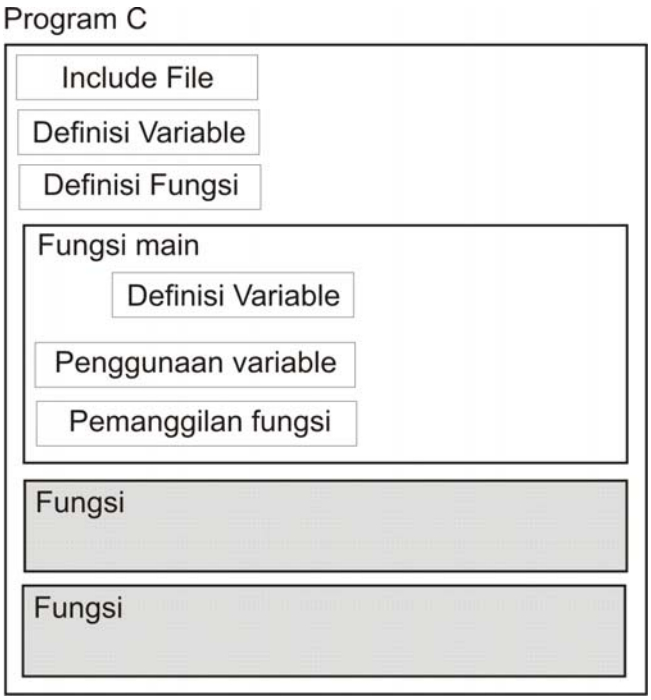


Sebuah fungsi memiliki nama, tipe keluaran, dan tipe-tipe argumen jika ada. Seperti yang sudah kita bahas pada bab 1, contoh fungsi adalah `int main()`. Fungsi ini memiliki nama `main`, tipe keluarannya `int`, dan tidak memiliki argumen.

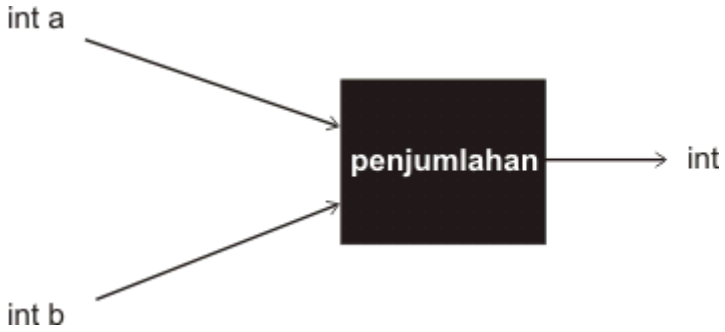
Membuat Fungsi

Dalam bahasa C, kita dapat menggunakan fungsi-fungsi standar yang sudah disediakan, seperti `printf` dan `scanf`. Kita juga dapat membuat fungsi-fungsi sendiri sesuai keperluan.

Sesuai dengan aturan umum pemrograman bahasa C, apa yang akan kita gunakan pada program, harus diberitahu (dideklarasikan) terlebih dahulu. Karena itu, kita perlu mendeklarasikan fungsi-fungsi yang akan kita buat/pakai. Selanjutnya barulah kita tuliskan isi dari fungsi tersebut. Penulisan isi fungsi biasanya diletakan setelah fungsi `main`, namun tidak menutup kemungkinan diletakan ditempat lain.



Contoh fungsi :



Deklarasi :

```
int penjumlahan(int a, int b);
```

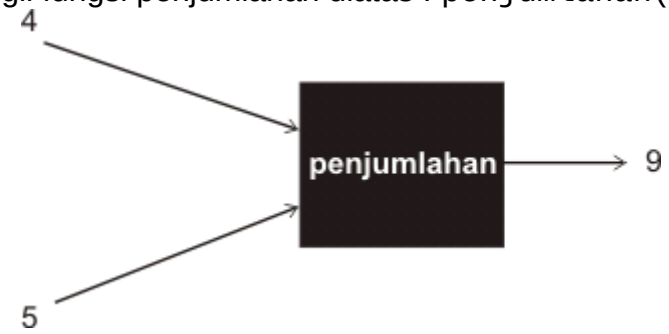
Isi Fungsi :

```
int penjumlahan(int a, int b)
{
  return (a+b);
}
```

Fungsi diatas bernama `penjumlahan`, memiliki tipe keluaran `int`, dan memiliki 2 buah argumen. Argumen pertama adalah a, bertipe `int`. Argumen kedua adalah b, bertipe `int`.
Fungsi ini mengerjakan perintah `return(a+b);`. Perintah `return` adalah perintah untuk mengembalikan nilai fungsi.

Pemanggilan Fungsi
Untuk menggunakan fungsi yang sudah dibuat, kita perlu memanggilnya. Pemanggilan fungsi dilakukan dengan menuliskan namanya, kemudian tanda kurung buka-kurung tutup. Jika fungsi tersebut menggunakan argumen, kita perlu mengisikan nilai-nilai tersebut.

Contoh memanggil fungsi penjumlahan diatas : `penjumlahan(4,5)`



Contoh pemanggilan fungsi :

```
#include<stdio.h>

int penjumlahan(int a, int b);

int main()
{
  int hasil;
  hasil = penjumlahan(4,5);
  printf("hasil = %d\n", hasil);

  return 0;
}

int penjumlahan(int a, int b)
{
  return (a+b);
}
```

Prosedur
Pada bahasa C, prosedur dapat dikatakan "sebuah fungsi yang tidak mengembalikan nilai". Sama seperti fungsi, prosedur juga boleh menerima argumen.
Karena prosedur tidak mengembalikan nilai, maka tipe keluarannya adalah `void`.

Contoh :

```
#include<stdio.h>

void deret_genap(int n);
```

```
int main()
{
  deret_genap(10);
  return 0;
}

void deret_genap(int n)
{
  int i;

  for(i=0; i<n; i++)
    printf("%d ", i*2);
}
```

Passing Argument

Passing Argument artinya memasukan argumen kedalam sebuah fungsi/prosedur. Pada kedua contoh diatas kita sudah mencoba memasukan argumen berupa konstanta integer (4, 5, 10). Sebenarnya argumen tidak terbatas pada konstanta saja, kita juga dapat memasukan variable kedalam fungsi. Cobalah contoh berikut ini :

```
#include<stdio.h>

float luas_lingkaran(float r);

int main()
{
  float jari2, luas;
  printf("masukan jari-jari : ");
  scanf("%f", &jari2);

  luas = luas_lingkaran(jari2);
  printf("Luas = %.2f\n", luas);

  return 0;
}

float luas_lingkaran(float r)
{
  return (r*r*3.14);
}
```

Perhatikan pemanggilan fungsi pada cotoh diatas !

```
luas_lingkaran(jari2)
```

Argumen `jari2` merupakan sebuah variable. Perlu diperhatikan disini, variable yang boleh di-*passing* hanya variable-variable yang tipenya cocok. Pada contoh diatas, fungsi `luas_lingkaran` menerima argumen `(float r)`, sehingga variable `jari2` yang bertipe float dapat di-*passing*.

Passing by Value

Passing by Value artinya memasukan argumen berupa nilainya saja. Contoh diatas, `luas_lingkaran(jari2)`, sebenarnya menggunakan cara *passing by value*. Cara ini menyebabkan variable yang di-*passing* tidak akan berubah nilainya. Cobalah program dibawah ini :

```
#include<stdio.h>

void coba(int a);

int main()
{
  int x;

  x = 10;
  coba(x);
  printf("nilai x : %d\n", x);

  return 0;
}

void coba(int a)
{
  a = a+10;
}
```

Pada contoh diatas, nilai x setelah melewati fungsi `coba` tetap 10.

Passing by Address

Berbalikan dengan *passing by value*, cara ini menyebabkan nilai variable dapat berubah sesuai dengan apa yang terjadi di dalam fungsi. Untuk mem-*pass* variable dengan cara ini, gunakan tanda * pada fungsi, dan tanda & pada pemanggilan. Cobalah contoh berikut :

```
#include<stdio.h>

void coba(int *a);

int main()
{
  int x;
```

```
  x = 10;
  coba(&x);
  printf("nilai x : %d\n", x);

  return 0;
}

void coba(int *a)
{
  *a = *a+10;
}
```