Function

Function (Fungsi) merupakan suatu blok yang berisi kode program yang dirancang untuk melaksanakan tugas khusus.

Pada intinya fungsi berguna untuk :

- Mengurangi pengulangan penulisan program yang berulangan atau sama.
- Program menjadi terstruktur, sehingga mudah dipahami dan dikembangkan.

1. Struktur Fungsi

Sebuah fungsi sederhana mempunyai bentuk penulisan sebagai berikut :

```
nama_fungsi(argumen)
{
        … pernyataan / perintah;
        … pernyataan / perintah;
        … pernyataan / perintah;
}
```

Contoh pembuatan fungsi sederhana

```
/* pembuatan fungsi garis()  */
garis()
{
  cout <<“----------------------”<<endl;
}
/* program utama */
void main()
{
   clrscr();
   garis();
   cout<<“SELAMAT BELAJAR BAHASA C++"<<endl;;
   garis();
  getche();
}
```

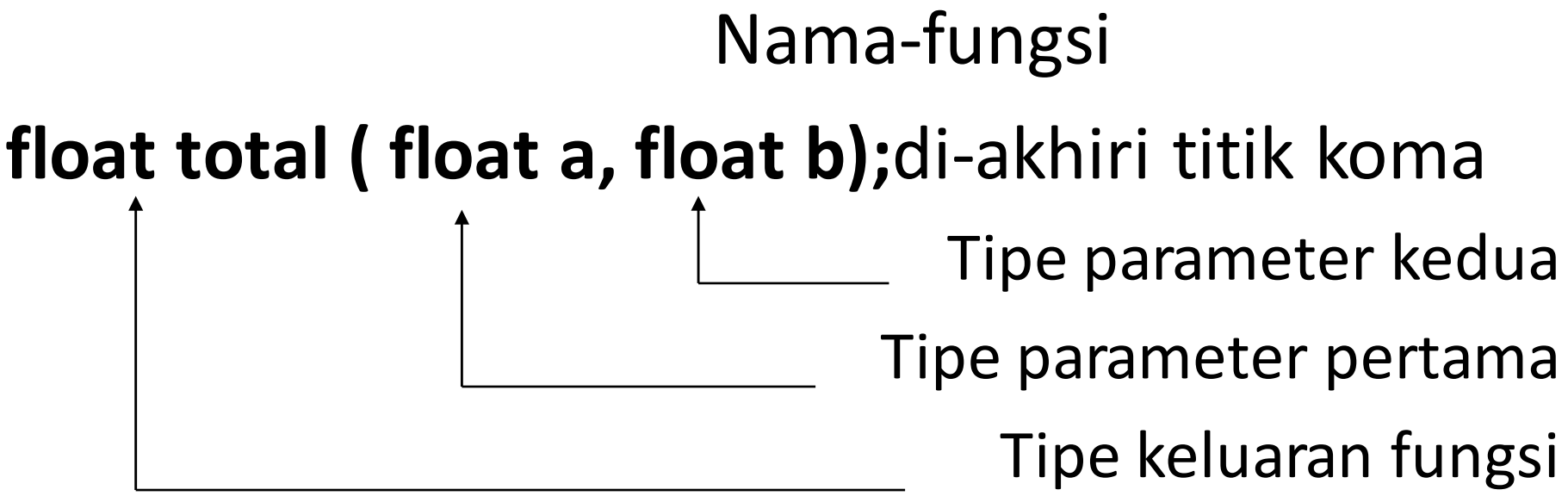
2. Prototipe Fungsi

Prototipe fungsi digunakan untuk menjelaskan kepada kompiler mengenai :

- Tipe keluaran fungsi.
- Jumlah parameter.
- Tipe dari masing-masing parameter.

Salah satu keuntungan pemakai prototipe, kompiler akan melakukan konversi antara tipe parameter dalam definisi dan parameter saat pemanggilan fungsi tidak sama atau akan menunjukkan kesalahan jika jumlah parameter dalam definisi dan saat pemanggilan berbeda.

Contoh prototipe fungsi :

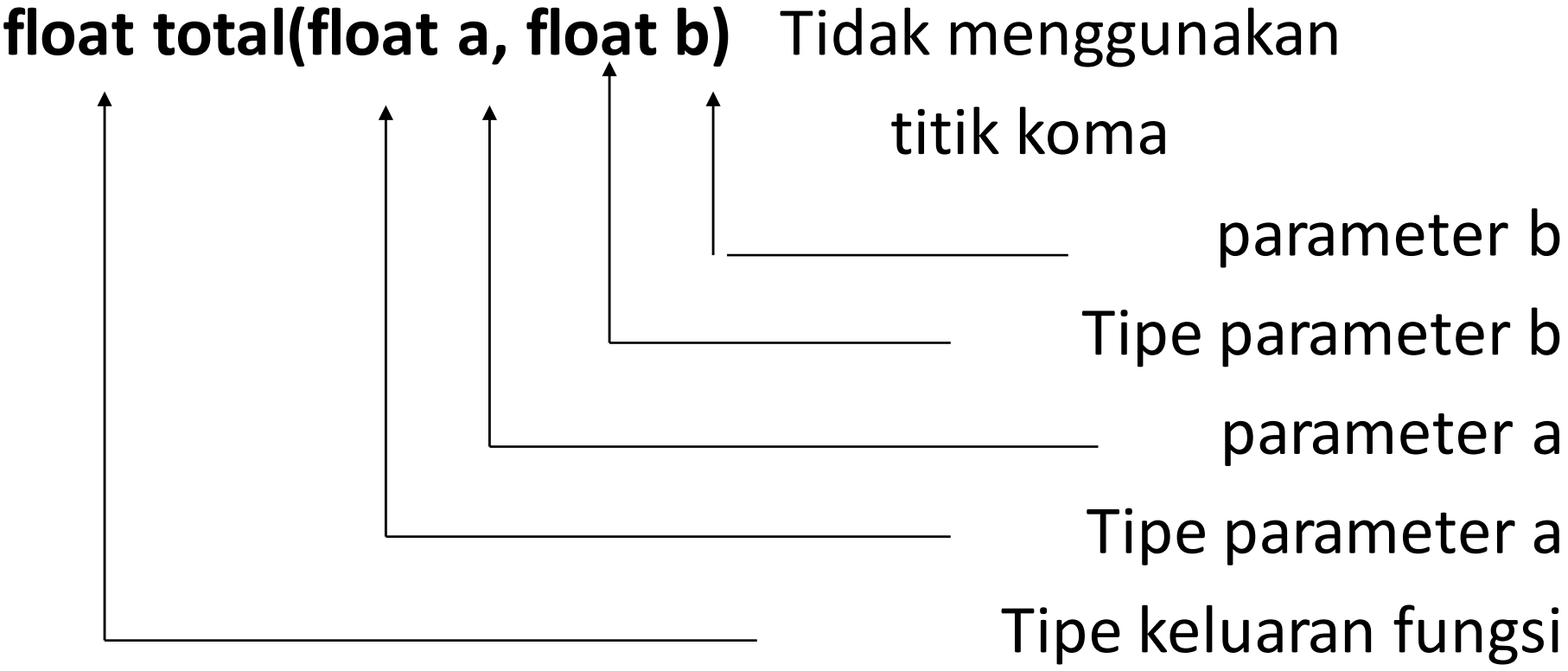


Jika dalam penggunaan fungsi yang dideklarasikan dengan menggunakan prototipe, maka bentuk definisi harus diubah. Sebagai contoh pada pendefinisian berikut :

float total(a, b)
float a, y;

Bentuk pendefinisian diatas harus diubah menjadi bentuk modern pendefinisian fungsi :

Nama fungsi

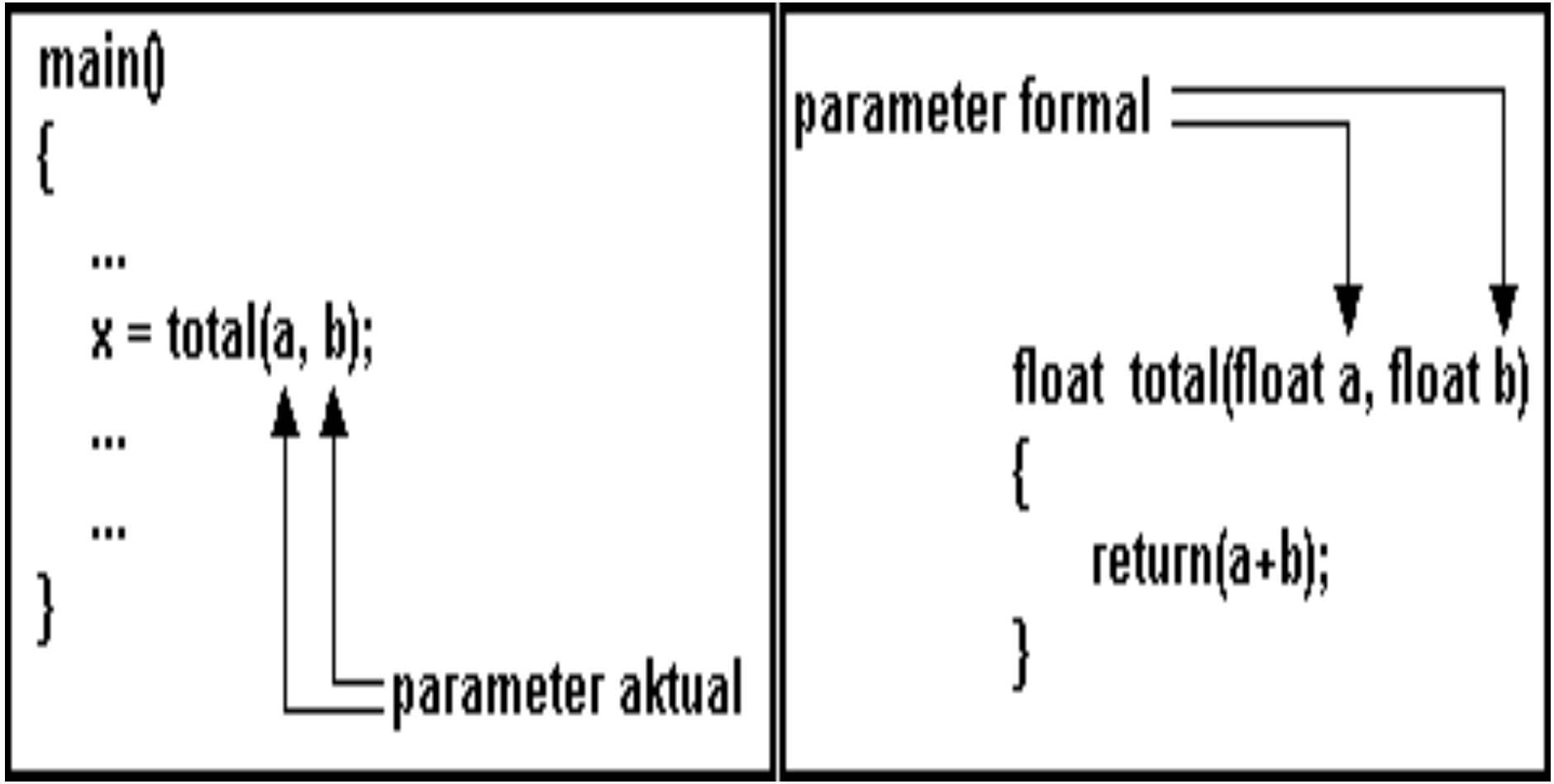


3. Parameter Fungsi

Terdapat dua macam para parameter fungsi, yaitu :

a. **Parameter formal** adalah *variabel yang ada pada daftar parameter dalam definisi fungsi*.
b. **Parameter Aktual** adalah *variabel yang dipakai dalam pemanggilan fungsi.*

Bentuk penulisan Parameter Formal dan Parameter Aktual.



a. Pemanggilan dengan nilai (*Call by Value*)

Pemanggilan dengan nilai merupakan cara yang dipakai untuk seluruh fungsi buatan yang telah dibahas didepan. Pada pemanggilan dengan nilai, nilai dari parameter aktual akan ditulis keparameter formal. Dengan cara ini nilai parameter aktual tidak bisa berubah, walaupun nilai parameter formal berubah.

```
/* Penggunaan Call By Value
   Program Pertukaran Nilai
   ---------------------------  */
#include<conio.h>
#include<stdio.h>
#include<iostream.h>
tukar(int x, int y);
void main()
{
int a, b;
cout<<"MASUKAN  BILANGAN PERTAMA :"; cin >>
      a;
cout<<"MASUKAN  BILANGAN KEDUA   :"; cin >> b;
clrscr();
cout<<"NILAI  SEBELUM PEMANGGILAN
      FUNGSI"<<endl;
cout<<"BILANGAN  PERTAMA = "<<a<<endl;
cout<<"BILANGAN  KEDUA   = "<<b<<endl;
tukar(a,b);
cout<<"\nNILAI  SETELAH  PEMANGGILAN
      FUNGSI"<<endl;
cout<<"BILANGAN  PERTAMA = "<<a<<endl;
cout<<"BILANGAN  KEDUA   = "<<b<<endl;
getch();
}
```

```
tukar(int x, int y)
{
  int z;
  z = x;  x = y;  y = z;
  cout<<"\n\nNILAI  SETELAH
      PERTUKARAN"<<endl;
  cout<<"BILANGAN  PERTAMA  = "<<x<<endl;
  cout<<"BILANGAN  KEDUA    = "<<y<<endl;
}
```

b. Pemanggilan dengan Referensi (*Call by Reference*)

Pemanggilan dengan reference merupakan upaya untuk melewatkan alamat dari suatu variabel kedalam fungsi. Cara ini dapat dipakai untuk mengubah isi suatu variabel diluar fungsi dengan melaksanakan pengubahan dilakukan didalam fungsi.

```
/* Penggunaan Call By Reference
   Program Pertukaran Nilai
    -------------------------- */
#include<conio.h>
#include<stdio.h>
#include<iostream.h>
tukar(int *x, int *y);
void main()
{
      int a, b;
      cout<<"MASUKAN BILANGAN PERTAMA :"; cin
      >> a;
      cout<<"MASUKAN BILANGAN KEDUA   :"; cin
      >> b;
      clrscr();
      cout<<"Nilai Sebelum Pemanggilan Fungsi";
      cout<<"\na = "<<a<<"          b = "<<b;
      tukar(&a,&b);
      cout<<endl;
      cout<<"\nNilai Setelah Pemanggilan Fungsi";
      cout<<"\na = "<<a<<"          b = "<<b;
      getch();
}
```

```
tukar(int *x, int *y)
{
 int z;
 z = *x;  *x = *y;  *y = z;
 cout<<endl;
 cout<<"\nNilai di Akhir Fungsi Tukar()";
 cout<<"\nx = "<<*x<<"                y = "<<*y;
}
```

4. Pengiriman Data Ke Fungsi

a. Pengiriman Data Konstanta Ke Fungsi

Mengirimkan suatu nilai data konstanta ke suatu fungsi yang lain dapat dilakukan dengan cara yang mudah, dapat dilihat dari program berikut :

```
/* ---------------------- */
/* Pengriman data Konstanta */
/* ---------------------- */
#include<conio.h>
#include<stdio.h>
#include<iostream.h>
luas(float sisi);
main()
{
   float luas_bs;
   clrscr();
   luas_bs = luas(4.25);
   cout<<"\nLuas Bujur Sangkar = "<<luas_bs;
   getch();
}
luas(float sisi)
{
  return(sisi*sisi);
}
```

b. Pengiriman Data Variabel Ke Fungsi

Bentuk pengiriman data Variabel, sama seperti halnya pengiriman suatu nilai data konstanta ke suatu fungsi, hanya saja nilai yang dikirimkan tersebut senantiasa dapat berubah-ubah. Bentuk pengiriman tersebut dapat dilihat dari program berikut :

```
/* ---------------------- */
/* Pengriman data Konstanta */
/* ---------------------- */
#include<conio.h>
#include<stdio.h>
#include<iostream.h>
luas(float sisi);
main()
{
  float luas_bs, sisi_bs;
  clrscr();
  cout<<"\nMenghitung Luas Bujur Sangkar"<<endl;
  cout<<"\nMasukan Nilai Sisi Bujur Sangkar : ";
  cin>>sisi_bs;
  luas_bs = luas(sisi_bs);
  cout<<"\nLuas Bujur Sangkar = "<<luas_bs<<" Cm";
  getch();
}
luas(float sisi)
{
 return(sisi*sisi);
}
```

5. Pernyataan *return()*.

Digunakan untuk mengirimkan nilai atau nilai dari suatu fungsi kepada fungsi yang lain yang memanggilnya. Pernyataan ***return()*** diikuti oleh argumen yang berupa nilai yang akan dikirimkan. Contoh pemakaian pernyataan ***return()*** dapat dilihat pada contoh berikut ;

```
/* ------------------------ */
/* Penggunaan Fungsi return() */
/* ------------------------ */
#include<conio.h>
#include<stdio.h>
#include<iostream.h>
tambah(int c);   //prototype fungsi tambah
main()
{
   int a, b = 5;
   clrscr();
   a = tambah(b);
   cout<<"Nilai Setelah Pemanggilan Fungsi :"<<a;
   getch();
}
tambah(int c)   //fungsi tambah
{
  return(c+=2);
}
```

6. inline Function

Fungsi inline (*inline function*) digunakan untuk mempercepat proses program, terutama program-program yang menggunakan sering menggunakan fungsi, terutama program-program yang menggunakan pernyataan perulangan proses seperti for, while dan do – while. Inline function dideklarasikan dengan menyisipkan kata kunci **inline** didepan tipe data.

```
/* ------------------------ */
/* Penggunaan inlide function */
/* ------------------------ */
#include<conio.h>
#include<stdio.h>
#include<iostream.h>
inline int hitung(int a, int b)
{
  return(a * b);
}
main()
{
  int k;
  clrscr();
  for(k = 1; k < 20; k++)
    cout<< k <<". "<<hitung(k, 2 * k) << endl;
  getch();
}
```

7. Function Overloading

Function Overloading adalah mendefinisikan beberapa fungsi, sehingga memiliki nama yang sama. Dapat diartikan bahwa fungsi yang overload berarti menyediakan versi lain dari fungsi tersebut. Salah satu kelebihan dari C++ adalah Overloading.

Sebagai contoh membentuk fungsi yang sama dengan tipe yang berbeda-beda dan dibuatkan pula nama fungsi yang berbeda-beda pula.

```
/* --------------------------- */
/* Penggunaan function overloading */
/* --------------------------- */
#include<conio.h>
#include<stdio.h>
#include<iostream.h>
int hitung(int b);
long hitung(long c);
float hitung(float d);
void main()
{
  clrscr();
  cout<<hitung(4)<<endl;
  cout<<hitung(2)<<endl;
  cout<<hitung(3)<<endl;
  cout<<hitung(5)<<endl;
  getch();
}
int hitung(int b)
{
 return(b*b);
}
long hitung(long c)
{
 return(c*c);
}
double hitung(double d)
{
 return(d*d);
}
```