Fungsi dan Prosedur

Modular Programming

- Program pendek dan simple = mudah dihandle.
- Program besar, banyak dan kompleks = tidak mudah dihandle.
- Kesulitan:
  - sulit mencari dan mengingat variabel-variabel yang sudah dideklarasikan
  - sulit melakukan dokumentasi
  - sulit mencari kesalahan program
  - sulit melihat efisiensi algoritma
  - code program kadang ditulis berulang-ulang padahal mengerjakan suatu hal yang sama

Modular Programming (2)

- Merupakan paradigma pemrograman yang pertama kali diperkenalkan oleh Information & Systems Institute, Inc. pada the National Symposium on Modular Programming pada 1968.
- Salah satu tokoh modular programming adalah **Larry Constantine**
- Pemrograman Modular adalah suatu teknik pemrograman di mana program yang biasanya cukup besar dibagi-bagi menjadi beberapa bagian program yang lebih kecil
- Keuntungan:
  - Program lebih pendek
  - Mudah dibaca dan dimengerti
  - Mudah didokumentasi
  - Mengurangi kesalahan dan mudah mencari kesalahan
    - Kesalahan yang terjadi bersifat “lokal”

Modular programming pada C

- Bahasa C sangat mendukung modular programming
- Sejak awal bahasa C sudah membagi program-programnya menjadi modul-modul (bagian-bagian)
- Modul pada bahasa C dikenal dengan nama fungsi (function)
- Bahasa C terdiri dari fungsi-fungsi, baik yang langsung dideklarasikan dalam program ataupun dipisah di dalam header file.
- Fungsi yang selalu ada pada program C adalah fungsi **main**

Function

- **Fungsi / function** adalah suatu kumpulan instruksi/perintah/program yang dikelompokkan menjadi satu, letaknya terpisah dari program yang menggunakan fungsi tersebut, memiliki nama tertentu yang unik, dan digunakan untuk mengerjakan suatu tujuan tertentu.
- Dalam bahasa pemrograman lain fungsi dapat disebut sebagai subrutin (basic, VB) atau procedure (pascal, Delphi)

Keuntungan Fungsi

- Dapat melakukan pendekatan *top-down* dan *divide-and-conquer*:
  - Top-down: penelusuran program mudah
  - Divide-and-conquer: program besar dapat dipisah menjadi program-program kecil.
- Kode program menjadi lebih pendek, mudah dibaca, dan mudah dipahami
- Program dapat dikerjakan oleh beberapa orang sehingga program cepat selesai dengan koordinasi yang mudah.
- Mudah dalam mencari kesalahan-kesalahan karena alur logika jelas dan sederhana
- Kesalahan dapat dilokalisasi dalam suatu modul tertentu saja.
- Modifikasi program dapat dilakukan pada suatu modul tertentu saja tanpa mengganggu program keseluruhan

Keuntungan Fungsi (2)

- Fungsi – fungsi menjadikan program mempunyai struktur yang jelas.
  - Dengan memisahkan langkah – langkah detail ke satu atau lebih fungsi – fungsi, maka fungsi utama (**main**) akan menjadi lebih pendek, jelas dan mudah dimengerti.
- Fungsi –fungsi digunakan untuk menghindari penulisan program yang sama yang ditulis secara berulang – ulang. Langkah – langkah tersebut dapat dituliskan sekali saja secara terpisah dalam bentuk fungsi. Selanjutnya bagian program yang membutuhkan langkah – langkah ini tidak perlu selalu menuliskannya, tidak cukup memanggil fungsi tersebut.
- Mempermudah dokumentasi.
- *Reusability*: Suatu fungsi dapat digunakan kembali oleh program atau fungsi lain

Sifat-sifat fungsi

- Nilai **fan-in** tinggi, artinya semakin sering suatu modul dipanggil oleh pengguna semakin tinggi nilai fan-in
- Nilai **Fan-out** rendah, artinya semakin spesifik fungsi suatu modul akan semakin rendah nilai fan-out
- Memiliki **Self-contained** tinggi: artinya kemampuan untuk memenuhi kebutuhannnya sendiri

Kategori fungsi dalam C

- **Standard Library Function**
  - Yaitu fungsi-fungsi yang telah disediakan oleh C dalam file-file header atau librarynya.
  - Misalnya: clrscr(), printf(), getch()
  - Untuk function ini kita harus mendeklarasikan terlebih dahulu library yang akan digunakan, yaitu dengan menggunakan preprosesor direktif.
    - Misalnya: #include <conio.h>
- **Programmer-Defined Function**
  - Adalah function yang dibuat oleh programmer sendiri.
  - Function ini memiliki nama tertentu yang unik dalam program, letaknya terpisah dari program utama, dan bisa dijadikan satu ke dalam suatu library buatan programmer itu sendiri yang kemudian juga di-include-kan jika ingin menggunakannya.

Perancangan Fungsi

Dalam membuat fungsi, perlu diperhatikan:

- Data yang diperlukan sebagai inputan
- Informasi apa yang harus diberikan oleh fungsi yang dibuat ke pemanggilnya
- Algoritma apa yang harus digunakan untuk mengolah data menjadi informasi

Contoh fungsi

Contoh:

```
int GetMax(int nFirst, int nLast)
{
   int nReturn;
   if (nFirst>nLast)
      nReturn=nFirst;
   else
      nReturn=nLast;
   return nReturn;
}
```

Contoh fungsi

```
#include<stdio.h>
int HITUNG(int A, int B);

void main()
{
 int A,B,T;
 A=5; B=2;T=0;
 T = HITUNG(A,B);
 printf("\n %d", T);
}

int HITUNG(int A, int B)
{  int T;
   A = A * 2;
   B = B * 2;
   T= A+B;
   return(T);
}
```

Bagian ini yang disebut : **main** program atau program induk atau disebut juga Fungsi Induk atau Function main Oleh Bahasa C diberi nama **main( )**

Dalam program induk ada instruksi yang memanggil atau menCALL Function lain, baik fungsi yang kita buat sendiri, maupun fungsi pustaka yang disediakan oleh C/C++

Bagian ini memuat fungsi. Fungsi ini fungsi yang kita buat sendiri
Fungsi ini mempunyai :
Nama : HITUNG
Tipe : int

Dalam contoh ini Fungsi HITUNG ditulis dibawah atau sesudah fungsi main Sebuah Fungsi dapat juga ditulis diatas atau sebelum Fungsi main()

Contoh fungsi

Contoh-02.

```
#include<stdio.h>
void main()
{
 printf("Jakarta");
}
```

Program ini **tidak** menggunakan Funtion lain selain main function

Tercetak: Jakarta

Struktur Fungsi

- Deklarasi function (function prototype/ declaration)

  Terdiri dari:

  - Judul fungsi
  - Tipe data yang akan dikembalikan/void
  - Tidak ada kode implementasi function tersebut

  Bentuk umum:

  tipe_data|void nama_fungsi([arguman 1, argument 2,….]);

Struktur Fungsi

- Tubuh Function/Definisi Function (Function Definition)

  Terdiri dari:

  - function prototype yang disertai dengan kode implementasi dari function tersebut,
  - yang berisikan statemen/instruksi yang akan melakukan tugas sesuai dengan tujuan dibuatnya fungsi tersebut.

Deklarasi fungsi (2)

- Deklarasi fungsi diakhiri dengan titik koma
- Tipe_data dapat berupa segala tipe data yang dikenal C ataupun tipe data buatan, namun tipe data dapat juga tidak ada dan digantikan dengan void yang berarti fungsi tersebut tidak mengembalikan nilai apapun
- Nama fungsi adalah nama yang unik
- Argumen dapat ada atau tidak (opsional) yang digunakan untuk menerima argumen/parameter. Antar argumen-argumen dipisahkan dengan menggunakan tanda koma.

Deklarasi Fungsi (3)

- Suatu fungsi **perlu** (tapi tidak harus) dideklarasikan sebelum digunakan.
- Untuk alasan dokumentasi program yang baik, sebaiknya semua fungsi yang digunakan dideklarasikan terlebih dahulu
- Deklarasi fungsi ditulis sebelum fungsi tersebut digunakan

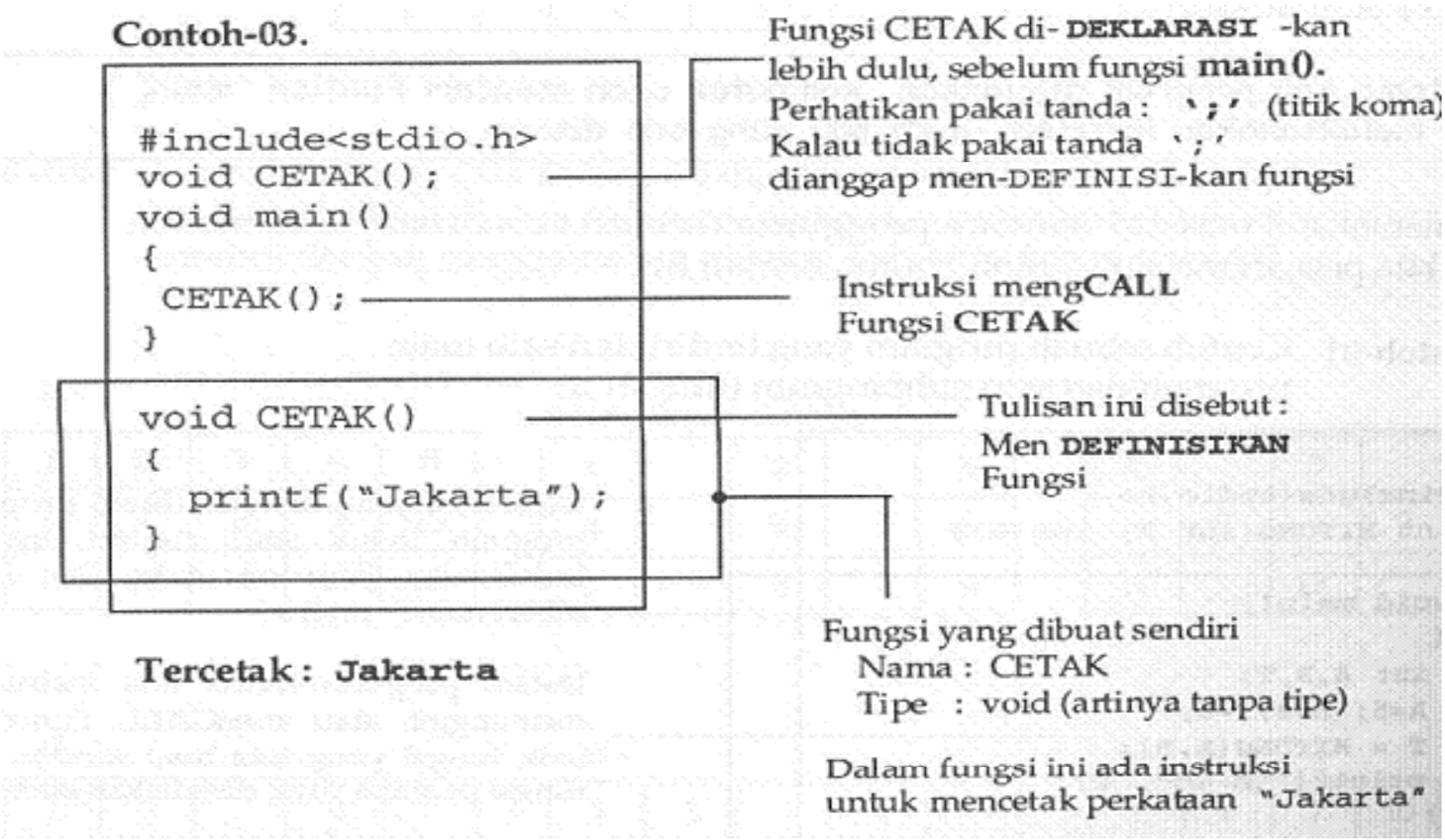
Bentuk Umum Definisi Fungsi

```
tipe_data/void nama_fungsi([arguman 1, argument 2,….]) //funciton prototype
{
      //bagian ini merupakan tubuh fungsi.
      [variabel_lokal;]

      [Statement_1;]
      [Statement_2;]
      ...
      [Statement_3;]
      [return (variabel)];
}
```

Definisi Fungsi (2)

- Tubuh fungsi dapat berisi segala perintah yang dikenal oleh C, pada dasarnya tubuh fungsi sama dengan membuat program seperti biasa.
- Return bersifat opsional, adalah **keyword** pengembalian nilai dari fungsi ke luar fungsi,
  - return wajib jika fungsi tersebut mengembalikan nilai berupa tipe data tertentu,
  - sedangkan return tidak wajib jika fungsi tersebut bersifat void.

Contoh Deklarasi dan Definisi Fungsi



Contoh Definisi Fungsi

```
#include <stdio.h>
/*----- Fungsi untuk memutlakan nilai negatip -----*/
double Absolut(double X)    /* definisi fungsi  */
{
       if(X<0) X= -X;
       return(X);
}

void main( )
{
       float Nilai;

       Nilai = -123,45;
      printf("%7,2f nilai mutlaknya adalah %7,2f\n",Absolut(Nilai));
}
```

Contoh Deklarasi dan Definisi Fungsi (2)

```
#include <stdio.h>
double Absolut(double X);    /*deklarsi fungsi Absolut */

void main()
{
      float Nilai;
      Nilai = -123,45;
      printf("%7,2f nilai mutlaknya adalah %7,2f\n",Nilai,
      Absolut(Nilai));
}

/*----- Fungsi untuk memutlakkan nilai negatip- -----*/

double Absolut(double X)    /* definisi fungsi */
{
     if(X<0) X = -X;
     return(X);
}
```

Jika program ini dijalankan, akan didapatkan hasil :

```
-123,45 nilai mutlaknya adalah 123,45
```

Kapan menggunakan Deklarasi dan Definisi Fungsi?

- Karena prinsip kerja program C sekuensial, maka
  - Jika bagian dari program yang menggunakan fungsi diletakkan sebelum definisi dari fungsi, maka **deklarasi dari fungsi diperlukan**.
  - Akan tetapi jika bagian dari program yang menggunakan fungsi terletak setelah definisi dari fungsi, maka **deklarasi dari fungsi boleh tidak dituliskan**.

Jenis fungsi dalam C

- Fungsi yang tidak mengembalikan nilai (void)
- Fungsi yang mengembalikan nilai (non-void)

Fungsi Void

- Fungsi yang void sering disebut juga **prosedur**
- Disebut void karena fungsi tersebut tidak mengembalikan suatu nilai keluaran yang didapat dari hasil proses fungsi tersebut.
- Ciri: tidak adanya keyword return.
- Ciri: tidak adanya tipe data di dalam deklarasi fungsi.
- Ciri: menggunakan keyword void.
- Tidak dapat langsung ditampilkan hasilnya
- Tidak memiliki nilai kembalian fungsi
- Contoh?

Fungsi non-void

- Fungsi non-void disebut juga **function**
- Disebut non-void karena mengembalikan nilai kembalian yang berasal dari keluaran hasil proses function tersebut
- Ciri: ada keyword return
- Ciri: ada tipe data yang mengawali deklarasi fungsi
- Ciri: tidak ada keyword void
- Memiliki nilai kembalian
- Dapat dianalogikan sebagai suatu variabel yang memiliki tipe data tertentu sehingga dapat langsung ditampilkan hasilnya.
- Contoh?

Contoh fungsi void dan non-void

- Void:

```
void tampilkan_jml(int a,int b){
    int jml;
    jml = a + b;
    printf("%d",jml);
}
```

- Non-void:

```
int jumlah(int a,int b){
    int jml;
    jml = a + b;
    return jml;
}
```

Keyword void

- Keyword void juga digunakan jika suatu function tidak mengandung suatu parameter apapun.

```
void print_error(void){
    printf("Error : unexpected error occurred!");
}
```

Contoh fungsi Faktorial

```
#include <stdio.h>
#include <conio.h>

int faktorial (int N);    /*prototype fungsi factorial*/

void main()
{
     int N;
     int fak;
     printf ("berapa factorial ? ");scanf("%d",&N);
     fak = faktorial(N);
     printf("%d factorial = %d\n",N,fak);
     getch();
}

/*-----fungsi untuk menghitung nilai N factorial -----*/
int faktorial(int N)   /*definisi fungsi*/
{
     int I;
     int F=1;
     if(N<=0)
     return(0);
     for(I=2;I<=N;I++) F *= I;
     return(F);
}
```

The main Function

- function **main()** dibutuhkan agar program C dapat dieksekusi!
- Tanpa function main, program C dapat dicompile tapi tidak dapat dieksekusi (harus dengan flag parameter –c, jika di UNIX)
- Pada saat program C dijalankan, maka compiler C pertama kali akan mencari function main() dan melaksanakan instruksi-instruksi yang ada di sana.
- Function main, sering dideklarasikan dalam 2 bentuk:
  - **int main()**
  - **void main()**

int main()

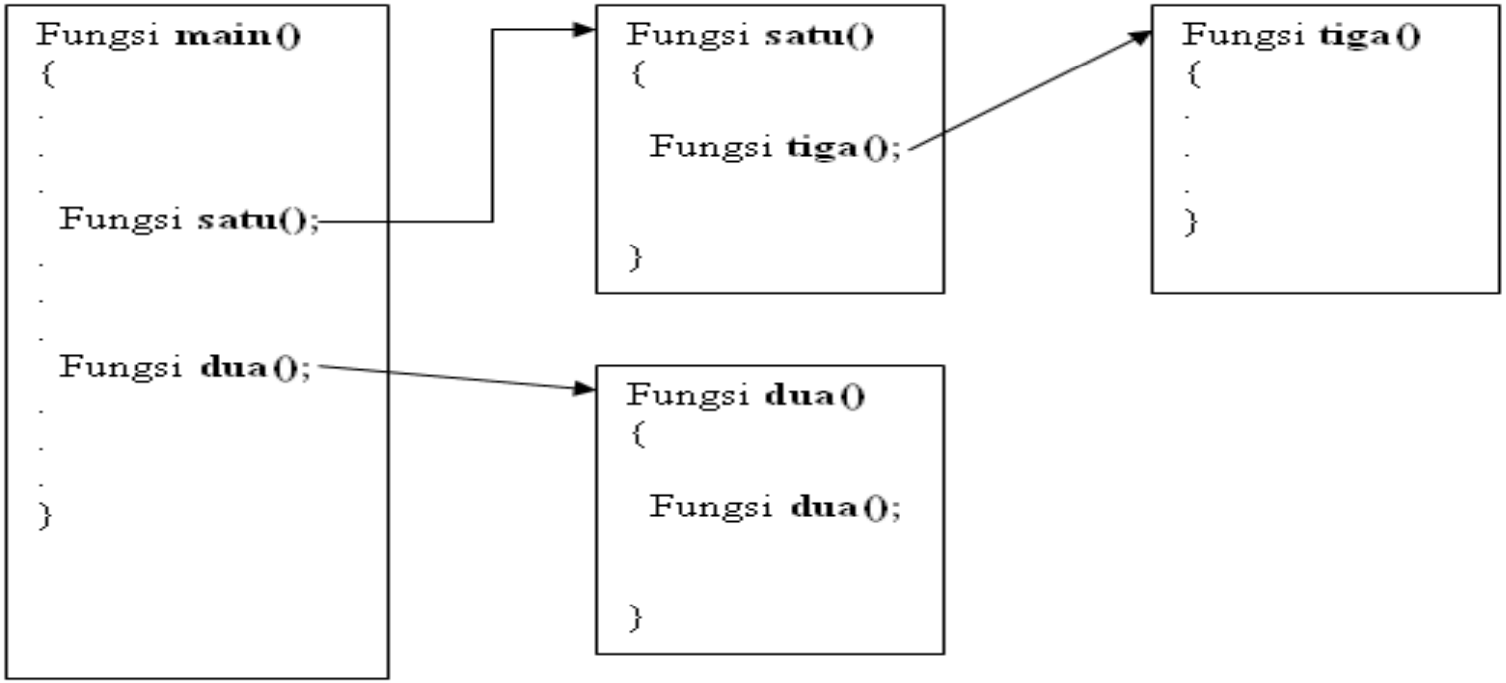
- Berarti di dalam function main tersebut harus terdapat keyword return di bagian akhir fungsi dan mengembalikan nilai bertipe data int,
- Mengapa hasil return harus bertipe int juga? karena tipe data yang mendahului fungsi main() diatas dideklarasikan int
- Tujuan nilai kembalian berupa integer adalah untuk mengetahui status eksekusi program.
  - jika “terminated successfully” (EXIT_SUCCESS) maka, akan dikembalikan status 0,
  - sedangkan jika “terminated unsuccessfully” (EXIT_FAILURE) akan dikembalikan nilai status tidak 0, biasanya bernilai 1
- Biasanya dipakai di lingkungan UNIX

void main()

- Berarti berupa function yang void sehingga tidak mengembalikan nilai status program sehingga nilai status program tidak bisa diketahui
- Biasanya dipakai pada program C di lingkungan Windows

Bentuk pemanggilan fungsi di C

- Pada dasarnya fungsi dapat memanggil fungsi lain, bahkan fungsi dapat memanggil dirinya sendiri (rekursif)



Scope Variable

- Sebuah variabel di dalam sebuah fungsi memiliki jangkauan tertentu.
- Skop variabel terdiri dari:
  - Variabel lokal
  - Variabel global
  - Variabel statis

Variabel lokal

- Variabel yang hanya dikenal di daerah yang lokal saja, misalnya di dalam sebuah fungsi/prosedur tertentu saja dan tidak dikenal di daerah lainnya.
- Harus dideklarasikan di dalam blok yang bersangkutan
- Variabel lokal dibentuk ketika fungsi dipanggil dan akan dihapus dari memori bila eksekusi terhadap fungsi selesai.
- Tidak ada inisialisasi otomatis

Variabel lokal (2)

Contoh-05a.

```
#include<stdio.h>
void CETAK();
void main()
{
 CETAK();
}

void CETAK()
{
   int A,B,T;
   A=5; B=2;
   T = A+B;
   printf("%d", T);
}
```

Tercetak : 7

DEKLARASI fungsi
tipe : void
karena tak ada nilai
yang dikirim ke fungsi
utama main()

Fungsi CETAK ini, merupakan suatu subprogram tersendiri yang dapat membuat variabel sendiri Semua variabel yang dibuat sendiri disini, variabel tersebut disebut bersifat LOKAL, yang artinya hanya berlaku dalam fungsi ini saja. Tidak berlaku di fungsi utama main(), atau dalam fungsi yang lainnya.

Variabel lokal (3)

```
#include <stdio.h>
void CETAK();

void main(){
     int A,B,T;
     A=5; B=2;
     T=A+B;
     CETAK();
}

void CETAK(){
     printf("%d",T);     //terjadi error, T tidak dikenal
}
```

Variabel Lokal (4)

```
#include <stdio.h>
#include <conio.h>

int TAMBAH(int A,int B);

int main(){
      int hasil;
      hasil = TAMBAH(2,3);
      printf("Hasil = %d",hasil);
      getch();
}

int TAMBAH(int A,int B){
      int C;
      C = A + B;
      {
            float C;
            C = 100;
      }
      return(C);
}
```

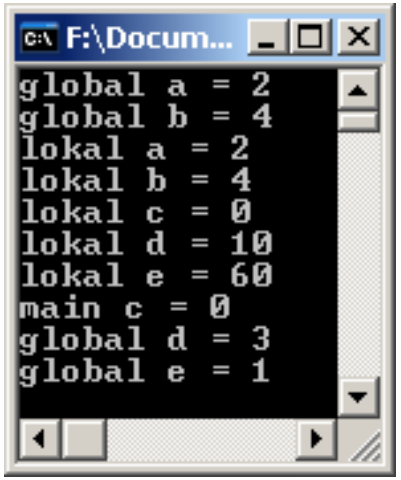
HasilNya: **5**

Mengapa tidak bernilai 100? Hal ini karena variabel C di deklarasikan di dalam blok sendiri sehingga dianggap berbeda dengan variabel C yang berisi nilai 5

Variabel Global

- Variabel yang dikenal diseluruh daerah di dalam program, di dalam dan luar fungsi.
- Dideklarasikan di luar suatu blok statemen atau di luar fungsi-fungsi yang menggunakannya.
- Variabel global dapat dideklarasikan kembali di dalam fungsi. (redeclare)
- Kerugian penggunaan variabel global:
  - Memboroskan memori computer karena computer masih menyimpan nilainya walaupun sudah tidak diperlukan lagi.
  - Mudah terjadi kesalahan program karena satu perubahan dapat menyebabkan perubahan menyeluruh pada program.
  - Pembuatan fungsi lebih sulit, karena harus diketahui variable global apa saja yang digunakan.
  - Pendeteksian kesalahan program lebih sulit dilakukan.

```
#include <stdio.h>
#include <conio.h>

int d=3,e=1;

void coba_lokal(int a,int b){
        int c = 0;
        int d = 10;
        int e;
        e = (a+b) * (c+d);
        printf("lokal a = %d\n",a);
        printf("lokal b = %d\n",b);
        printf("lokal c = %d\n",c);
        printf("lokal d = %d\n",d);
        printf("lokal e = %d\n",e);
}

void main(){
        int a=2;
        int b;
        b = 4;
        int c=0;

        printf("main a = %d\n",a);
        printf("main b = %d\n",b);
        coba_lokal(a,b);
        printf("main c = %d\n",c);
        printf("global d = %d\n",d);
        printf("global e = %d\n",e);
        getch();
}
```

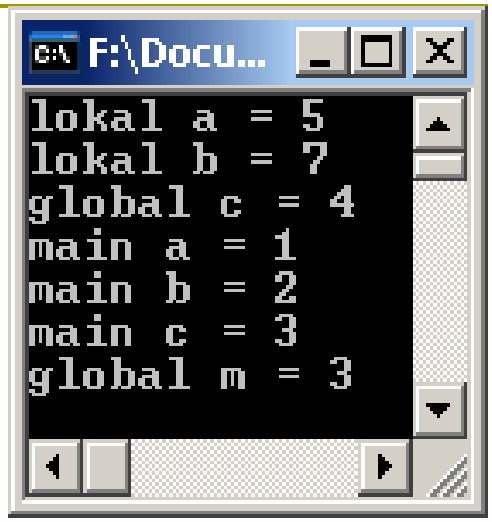


Pengenalan variabel

- Jika dalam sebuah fungsi terdapat variabel a dan di dalam program utama juga terdapat variabel a juga (nama sama), maka variabel yang dipakai tergantung dari siapa yang mengaksesnya.
- Jika yang mengakses adalah fungsi, maka variabel yang dipakai adalah variabel lokal, jika yang mengakses adalah program utama, maka yang dipakai adalah variabel dalam program utama.

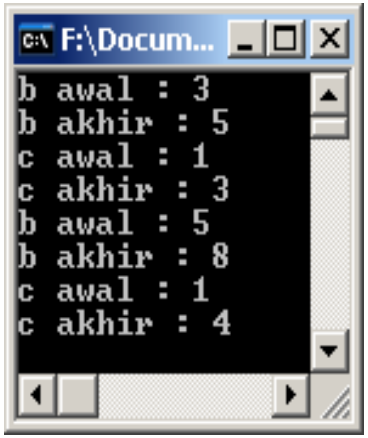
Contoh

```
#include <stdio.h>
#include <conio.h>

int c = 4;
int m = 3;

void lokal(){
      int a = 5;
      int b = a + 2;
      printf("lokal a = %d\n",a);
      printf("lokal b = %d\n",b);
      //karena tidak ada c, maka ambil global
      printf("global c = %d\n",c);
}

void main(){
      int a = 1;
      int b = 2;
      int c = 3;
      lokal();
      printf("main a = %d\n",a);
      printf("main b = %d\n",b);
      //walaupun global c ada tapi c yang digunakan yang di main
      printf("main c = %d\n",c);
      //karena tidak ada m, maka ambil global
      printf("global m = %d\n",m);
      getch();
}
```



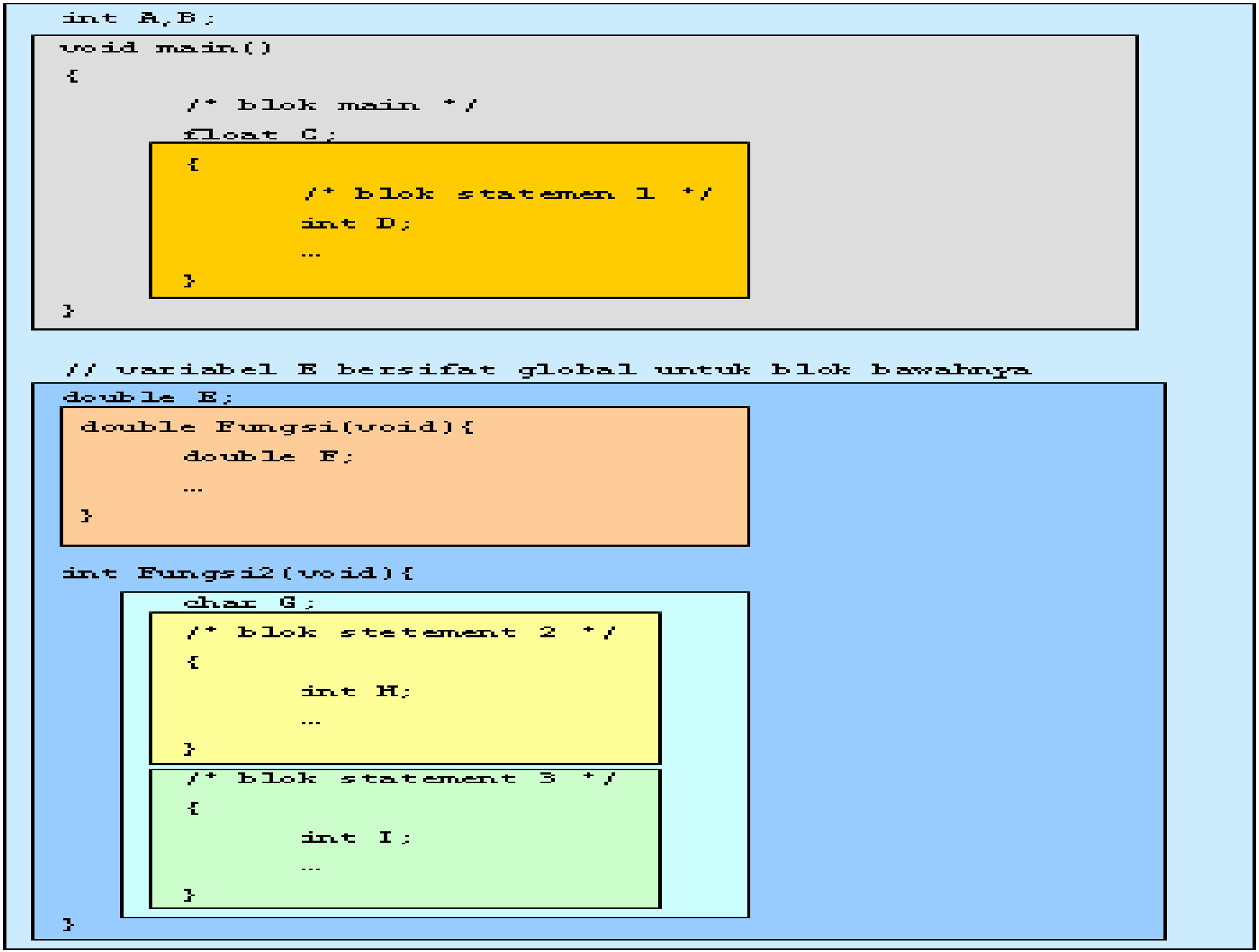
Variabel statis

```
#include <stdio.h>
#include <conio.h>

void coba_static(int a){
      static int b=3;
      int c=1;

      printf("b awal : %d\n",b);
      b += a;
      printf("b akhir : %d\n",b);

      printf("c awal : %d\n",c);
      c += a;
      printf("c akhir : %d\n",c);
}

void main(){
     int a=2;
     coba_static(a);
     a=3;
     coba_static(a);
     getch();
}
```

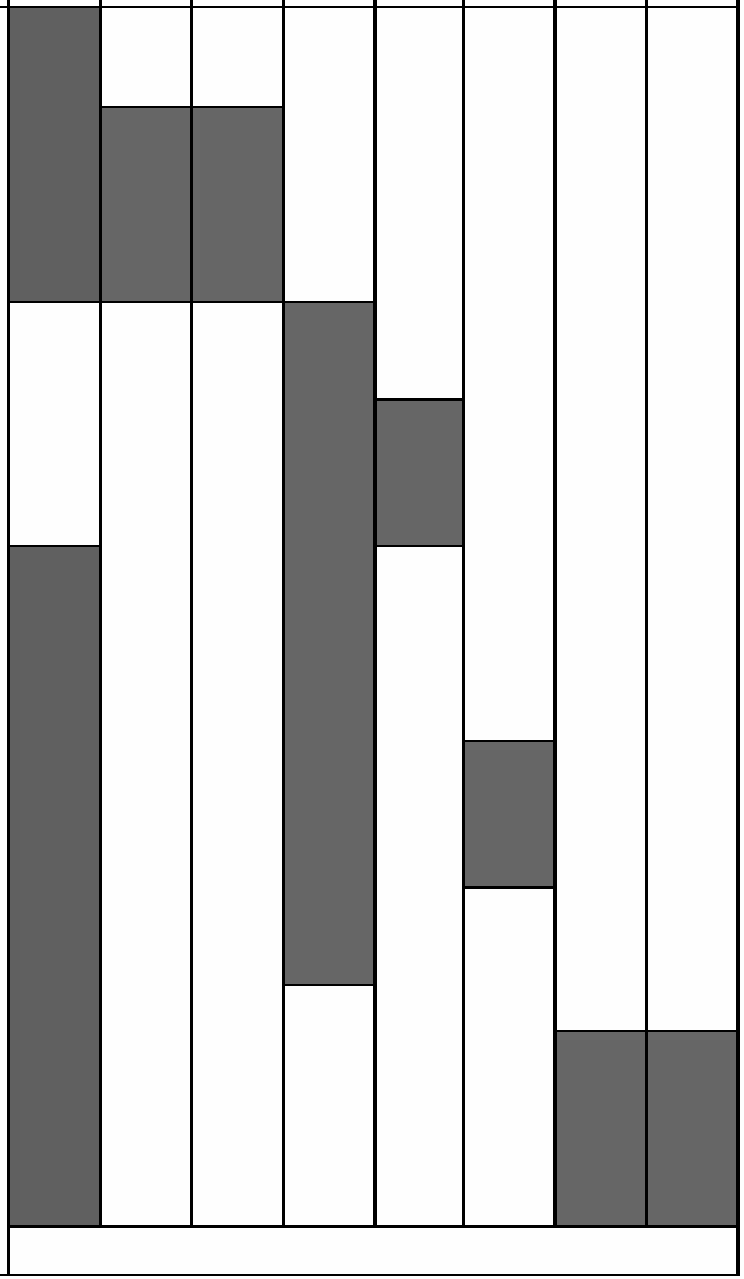


Variabel statis (2)

- Jika variabel statis bersifat lokal, maka hanya dikenal dalam fungsi tersebut saja.
- Jika variabel statis bersifat global, maka dikenal di seluruh program
- Inisialasisai hanya dilakukan sekali, yaitu pada saat fungsi dipanggil pertama kali.
- Adalah variabel yang memiliki nilai tetap, artinya nilai dari variabel tersebut akan tetap diingat oleh program, sehingga dapat digunakan untuk menyimpan state nilai pada saat pemanggilan fungsi berikutnya.
- Nilai variabel statis akan bernilai sama dengan nilai terakhirnya.


int A,B;
void main()
{
/* blok main */
float C;
{
/* blok statemen 1 */
int D;
...
}
}
// variabel E bersifat global untuk blok bawahnya
double E;
double Fungsi(void){
double F;
...
}
int Fungsi2(void){
char G;
/* blok stetement 2 */
{
int H;
...
}
/* blok statement 3 */
{
int I;
...
}
}

```
#include <stdio.h>
int x;
float func_1(int f1){
    int y1;
    ...
}

char y;
int func_2(){
    int x;
    ...
}

int func_3(){
    ...
    {
        int b;
        ...
    }
}

void main(){
    char y;
    int z;
    ...
}
```



Argumen Fungsi

- Sebuah fungsi bisa memiliki argumen-argumen yang bersifat opsional.
- Argumen-argumen tersebut berfungsi sebagai parameter inputan yang berupa variabel-variabel bagi fungsi tersebut (bersifat lokal).
- Argumen harus bertipe data tertentu.
- Terdapat 2 jenis parameter:
  - Parameter formal: parameter yang ditulis pada deklarasi fungsi.
  - Parameter aktual: parameter yang diinputkan dalam program pemanggil fungsi tersebut. Dapat berupa variabel atau langsung berupa nilai tertentu sesuai dengan tipe data yang dideklarasikan untuk masing-masing parameter fungsi

Parameter formal dan aktual

```
#include <stdio.h>
int JUMLAH(int X, int Y);

void main(){
      int A,B,T;
      A=5; B=2;
      T = JUMLAH(A,B);
      printf("%d",T);
}

int JUMLAH(int X, int Y){
      int H;
      H = X + Y;
      return(H);
}
```

X, Y disebut parameter formal

Variabel A,B,C lokal dalam main

A dan B disebut parameter aktual

X, Y disebut parameter formal

Variabel X,Y lokal dalam JUMLAH

Pengiriman parameter

- Komunikasi antar fungsi dilakukan dengan saling bertukar data
- Hasil dari suatu fungsi dapat diperoleh dari hasil baliknya (return),
- atau dengan variabel Global,
  - hasil proses dari suatu fungsi dapat diperoleh, karena variabel yang dipakai dalam fungsi bersifat global.
- Selain dengan cara tersebut di atas, hasil dapat juga diperoleh dari parameter aktual yang dikirimkan ke parameter formal, karena parameter formal seolah-olah akan mengirimkan kembali nilai hasil proses dalam fungsi.

Pengiriman Parameter

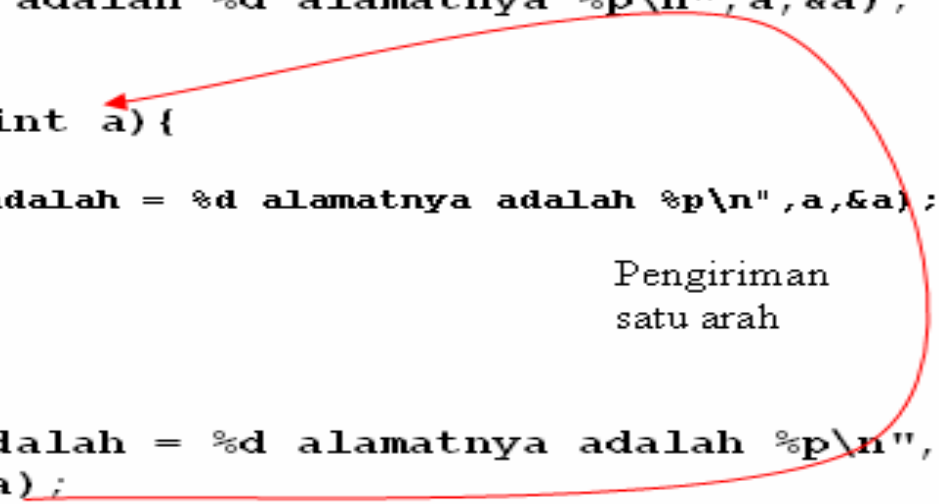
- Pengiriman secara nilai (*by value*)
  - Secara default pengiriman parameter di dalam C adalah by value
  - Pengubahan nilai di dalam fungsi tidak bisa mengubah nilai di luar fungsi
- Pengiriman secara acuan (*by reference*)
  - Tunggu di struktur data!

By Value

- Yang dikirimkan ke fungsi adalah **nilainya**, *bukan* **alamat memori** letak dari datanya
- Fungsi yang menerima kiriman nilai ini akan menyimpannya di **alamat terpisah** dari nilai aslinya yang digunakan oleh program yang memanggil fungsi tersebut
- Karena itulah pengubahan nilai di dalam fungsi **tidak akan berpengaruh** pada nilai asli di program yang memanggil fungsi walaupun keduanya menggunakan nama variabel yang sama
- Pengiriman by value adalah pengiriman **searah**, dari program pemanggil fungsi ke fungsi yang dipanggilnya
- Pengiriman by value dapat dilakukan untuk suatu **statement,** *tidak hanya* untuk suatu variabel, value, array atau konstanta saja.

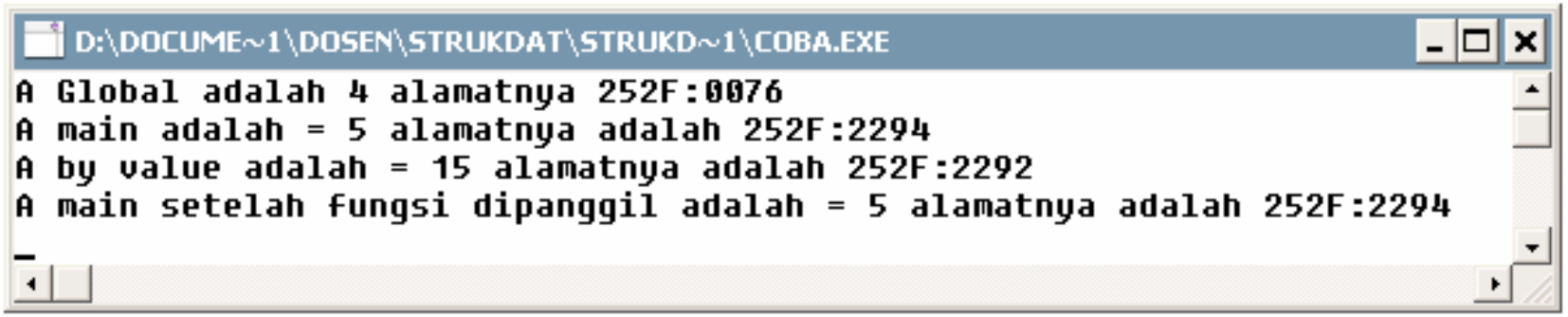
By Value (2)

```
#include <stdio.h>
#include <conio.h>

int a=4;

void getAGlobal(){
     printf("A Global adalah %d alamatnya %p\n",a,&a);
}

void fungsi_by_value(int a){
     a = a * 3;
     printf("A by value adalah = %d alamatnya adalah %p\n",a,&a);
}

void main(){
     int a = 5;
     getAGlobal();
     printf("A main adalah = %d alamatnya adalah %p\n",a,&a);
     fungsi_by_value(a);
     printf("A main setelah fungsi dipanggil adalah = %d
alamatnya adalah %p\n",a,&a);
     getch();
}
```



By Value (3)

Hasil:



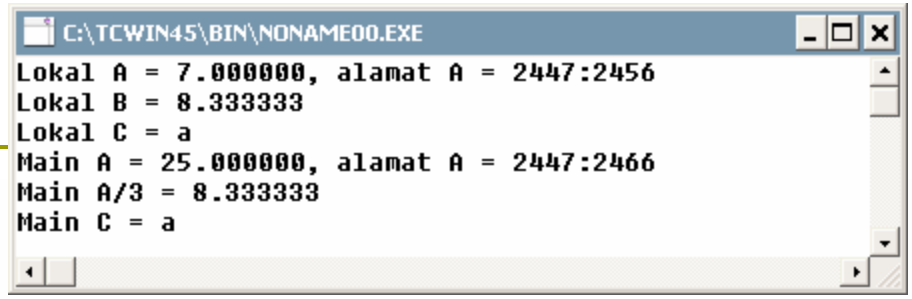
Di dalam Memori:

a di *global* nilai **4** alamat **252F:0076**	a di *main* nilai **5** alamat **252F:2294**	a di *fungsi_by_value* nilai **15** alamat **252F:2292**	a di *main after* *function* nilai **5** alamat **252F:2294**

By Value (4)

```
...
void fungsi_by_value(int a){
     a = a * 3;
     printf("A by value adalah = %d alamatnya adalah %p\n",a,&a);
}

void main(){
     int a = 5;
     getAGlobal();
     printf("A main adalah = %d alamatnya adalah %p\n",a);

     fungsi_by_value(5*a+1);
     getch();
}
...
```

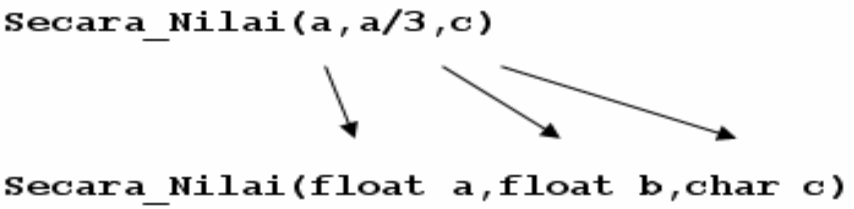
Contoh by value




```
#include <stdio.h>
#include <conio.h>

void Secara_Nilai(float a,float b,char c){
      float *Alamat_A;
      Alamat_A = &a;
      a = 7;
      printf("Lokal A = %f, alamat A = %p\n",a,Alamat_A);
      printf("Lokal B = %f\n",b);
      printf("Lokal C = %c\n",c);
}

void main(){
      float a=25,*Alamat_A;
      char c = 'a';
      Alamat_A = &a;
      Secara_Nilai(a,a/3,c);
      printf("Main A = %f, alamat A = %p\n",a,Alamat_A);
      printf("Main A/3 = %f\n",(a/3));
      printf("Main C = %c\n",c);
      getch();
}
```

Penjelasan

- Parameter aktual yang dikirimkan adalah datanya, yaitu **Secara_Nilai(a,a/3,c)**
- Alamat nilai a pada main dan a pada fungsi Secara_Nilai berbeda, yaitu **2447:2456** dan **2447:2466**
- Perubahan nilai a dalam fungsi Secara_Nilai menjadi **7** tidak mengubah nilai a pada main yaitu tetap **25**
- Pengirimannya satu arah
-



- Pengiriman parameter dapat berupa ungkapan (statement) yaitu **a/3**