



# MODUL PRAKTIKUM DASAR-DASAR PEMROGRAMAN



```
index =0;
7
8 while (index < 10)
9 {
10     printf("The index is %d \n", index);
11     index++;
12 }
13
14
15
16
17
18
19
20 /* for (index = 0; index <10; index++)
21 {
22     printf("The index is %d" index);
23 }
```

The index is 0  
The index is 1  
The index is 2  
The index is 3  
The index is 4  
The index is 5  
The index is 6  
The index is 7  
The index is 8  
The index is 9

Process returned 0 (0x0) execution time : 0.105 s  
Press any key to continue.

----- Run: Release in Loops (compiler: GNU GCC Compiler)-----  
Checking for existence: C:\Users\Brandon\Documents\Brandon\CS\_270\code\_blocks\Loops\bin\Release\Loops.exe  
Executing: "C:\Program Files (x86)\CodeBlocks\cb\_console\_runner.exe" "C:\Users\Brandon\Documents\Brandon\CS\_270\code\_blocks\Loops\bin\Release\Loops.exe" (in C:\Users\Brandon\Documents\Brandon\CS\_270\code\_blocks\Loops\bin\Release\Loops.exe)

Disusun Oleh:  
Prio Handoko, S.Kom., M.T.I.

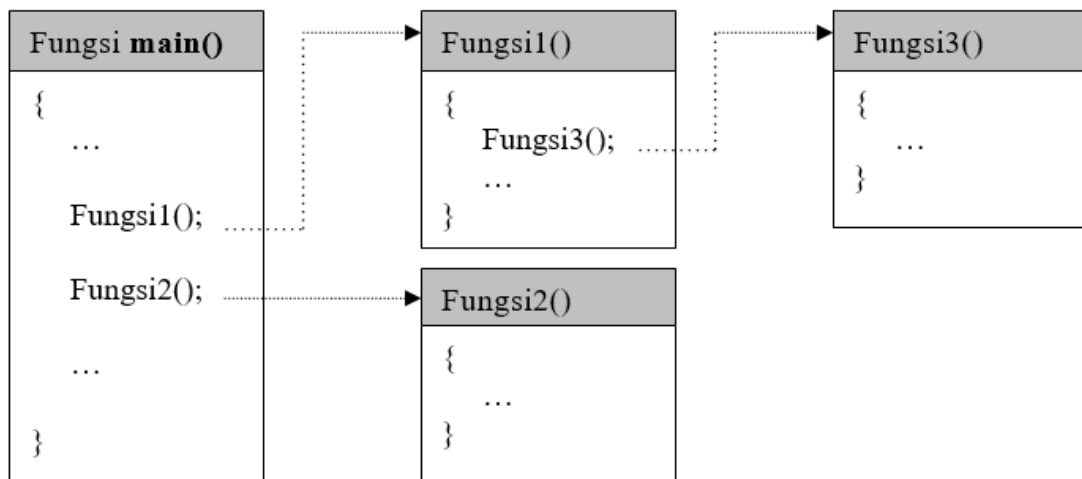
FAKULTAS DESAIN & TEKNOLOGI  
PROGAM STUDI INFORMATIKA  
UNIVERSITAS PEMBANGUNAN JAYA  
TANGERANG SELATAN 2019

## MODUL PRAKTIKUM 5

### FUNGSI

Menurut definisinya, fungsi adalah suatu blok program yang digunakan untuk melakukan proses-proses tertentu. Sebuah fungsi dibutuhkan untuk menjadikan program yang akan kita buat menjadi lebih modular dan mudah untuk dipahami alurnya. Dengan adanya fungsi, maka kita dapat mengurangi duplikasi kode program sehingga performa dari program yang kita buat pun akan meningkat. Dalam bahasa C, sebuah program terdiri atas fungsi-fungsi, baik yang didefinisikan secara langsung di dalam program maupun yang disimpan di dalam file lain (misalnya file *header*). Satu fungsi yang pasti terdapat dalam program yang ditulis menggunakan bahasa C adalah fungsi **main()**. Fungsi tersebut merupakan fungsi utama dan merupakan fungsi yang akan dieksekusi pertama kali.

Dalam bahasa C, fungsi terbagi menjadi dua macam, yaitu fungsi yang mengembalikan nilai (*return value*) dan fungsi yang tidak mengembalikan nilai. Fungsi yang tidak mengembalikan nilai tersebut dinamakan dengan *void function*. Bagi Anda yang sebelumnya pernah belajar bahasa Pascal, *void function* ini serupa dengan *procedure* yang terdapat di dalam bahasa Pascal. Gambar di bawah ini menerangkan bagaimana kompilator C membaca fungsi-fungsi yang didefinisikan di dalam program secara berurutan sesuai dengan waktu pemanggilannya.



Gambar 1. Pemanggilan fungsi dalam Bahasa C

Mula-mula fungsi **main()** akan dieksekusi oleh kompilator. Oleh karena fungsi **main()** tersebut memanggil **Fungsi1()**, maka kompilator akan mengeksekusi **Fungsi1()**. Dalam **Fungsi1()** juga terdapat pemanggilan **Fungsi3()**, maka kompilator akan mengeksekusi **Fungsi3()** dan kembali lagi mengeksekusi baris

selanjutnya yang terdapat pada **Fungsi1()** (jika ada). Setelah **Fungsi1()** selesai dieksekusi, kompilator akan kembali mengeksekusi baris berikutnya pada fungsi **main()**, yaitu dengan mengeksekusi kode-kode yang terdapat pada **Fungsi2()**. Setelah selesai, maka kompilator akan melanjutkan pengeksekusian kode pada baris-baris selanjutnya dalam fungsi **main()**. Apabila ternyata dalam fungsi **main()** tersebut kembali terdapat pemanggilan fungsi lain, maka kompilator akan meloncat ke fungsi lain tersebut, begitu seterusnya sampai semua baris kode dalam fungsi **main()** selesai dieksekusi.

### Apa Nilai yang dikembalikan Oleh Fungsi **main()** ?

Pada bab-bab sebelumnya kita telah banyak menggunakan fungsi **main()** di dalam program yang kita buat. Mungkin sekarang Anda akan bertanya apa sebenarnya nilai yang dikembalikan oleh fungsi **main()** tersebut? Jawabnya adalah nilai 0 dan 1. Apabila fungsi **main()** mengembalikan nilai 0 ke sistem operasi, maka sistem operasi akan mengetahui bahwa program yang kita buat tersebut telah dieksekusi dengan benar tanpa adanya kesalahan. Sedangkan apabila nilai yang dikembalikan ke sistem operasi adalah nilai 1, maka sistem operasi akan mengetahui bahwa program telah dihentikan secara tidak normal (terdapat kesalahan). Dengan demikian, seharusnya fungsi **main()** tidaklah mengembalikan tipe **void**, melainkan tipe data **int**, seperti yang terlihat di bawah ini.

```
int main(void) {
    ...
    return 0;          /* Mengembalikan nilai 0 */
}
```

Namun, apabila Anda ingin mendefinisikan nilai kembalian tersebut dengan tipe **void**, maka seharusnya Anda menggunakan fungsi **exit()** yang dapat berguna untuk mengembalikan nilai ke sistem operasi (sama seperti halnya **return**). Adapun parameter yang dilewatkan ke fungsi **exit()** ini ada dua, yaitu:

1. Nilai 0 (**EXIT\_SUCCESS**), yaitu menghentikan program secara normal
2. Nilai 1 (**EXIT\_FAILURE**), yaitu menghentikan program secara tidak normal

Berikut ini contoh penggunaannya di dalam fungsi `main()`.

```
void main(void) {
    ...
    exit(0);      /* dapat ditulis exit(EXIT_SUCCESS) */
}
```

### Fungsi Tanpa Nilai Balik (Void Function)

Pada umumnya fungsi tanpa nilai balik (*return value*) ini digunakan untuk melakukan proses-proses yang tidak menghasilkan nilai, seperti melakukan pengulangan, proses pengesetan nilai ataupun yang lainnya. Dalam bahasa C, fungsi semacam ini tipe kembaliannya akan diisi dengan nilai `void`. Adapun bentuk umum dari pendefinisian fungsi tanpa nilai balik adalah sebagai berikut:

```
void nama_fungsi(parameter1, parameter2,...) {
    Statemen_yang_akan_dieksekusi;
    ...
}
```

Berikut ini contoh dari pembuatan fungsi tanpa nilai balik.

```
void Tulis10Kali(void) {
    int j;
    for (j=0; j<10; j++) {
        printf("Saya sedang belajar bahasa C");
    }
}
```

Adapun contoh program lengkap yang akan menggunakan fungsi tersebut adalah seperti yang tertulis di bawah ini.

```
#include <stdio.h>

/* Mendefinisikan sebuah fungsi dengan nama Tulis10Kali */
void Tulis10Kali(void) {
    int j;

    for (j=0; j<10; j++) {
        printf("Saya sedang belajar bahasa C");
    }
}
```

```
int main(void) {
    Tulis10Kali();      /* Memanggil fungsi Tulis10Kali() */
    return 0;
}
```

### Fungsi dengan Nilai Balik

Berbeda dengan fungsi di atas yang hanya mengandung proses tanpa adanya nilai kembalian, di sini kita akan membahas mengenai fungsi yang digunakan untuk melakukan proses-proses yang berhubungan dengan nilai. Adapun cara pendefinisian adalah dengan menuliskan tipe data dari nilai yang akan dikembalikan di depan nama fungsi, berikut ini bentuk umum dari pendefinisian fungsi dengan nilai balik di dalam bahasa C.

```
tipe_data nama_fungsi(parameter1, parameter2,...) {
    Statemen_yang_akan_dieksekusi;
    ...
    return nilai_balik;
}
```

Sebagai contoh, di sini kita akan membuat fungsi sederhana yang berguna untuk menghitung nilai luas bujursangkar. Adapun sintak untuk pendefinisian adalah sebagai berikut.

```
int HitungLuasBujurSangkar(int sisi)
{
    int L;  /* mendeklarasikan variabel L untuk menampung nilai
            luas */
    L = sisi * sisi;    /* memasukkan nilai sesuai dengan rumus
                        yang berlaku */
    return L;          /* mengembalikan nilai yang didapat dari
                        hasil proses */
}
```

Sedangkan untuk menggunakan fungsi tersebut, Anda harus menuliskan program lengkap seperti di bawah ini.

```

int main(void)
{ int S, Luas;

    /* Mengeset nilai variabel S dengan nilai 10*/
    S = 10;

    /* Memanggil fungsi HitungLuasBujurSangkar
       dan menampung nilainya ke variabel Luas
    */
    Luas = HitungLuasBujurSangkar(S);

    /* Mencetak hasil perhitungan ke layar monitor */
    printf("Luas bujur sangkar dengan sisi %d adalah %d", S,
           Luas);

    return 0;
}

```

### Fungsi dengan Parameter

Parameter adalah suatu variabel yang berfungsi untuk menampung nilai yang akan dikirimkan ke dalam fungsi. Dengan adanya parameter, sebuah fungsi dapat bersifat dinamis. Parameter itu sendiri terbagi menjadi dua macam, yaitu parameter formal dan parameter aktual. Parameter formal adalah parameter yang terdapat pada pendefinisian fungsi, sedangkan parameter aktual adalah parameter yang terdapat pada saat pemanggilan fungsi. Untuk lebih memahaminya, perhatikan contoh pendefinisian fungsi di bawah ini.

```

int  TambahSatu(int  x) {
    return ++x;
}

```

Pada sintak di atas, variabel **x** dinamakan sebagai *parameter formal*. Sekarang perhatikan sintak berikut.

```

int main(void) { int
    a = 10, hasil;
    hasil =  TambahSatu(a);
    return 0;
}

```

Pada saat pemanggilan fungsi **TambahSatu()** di atas, variabel **a** dinamakan dengan *parameter aktual*.

## Jenis Parameter

Dalam dunia pemrograman dikenal tiga jenis parameter, yaitu parameter masukan, keluaran dan masukan/keluaran. Untuk memahami perbedaan dari setiap jenis parameter, di sini kita akan membahasnya satu per satu.

### Parameter Masukan

Parameter masukan adalah parameter yang digunakan untuk menampung nilai data yang akan dijadikan sebagai masukan (*input*) ke dalam fungsi. Artinya, sebuah fungsi dapat menghasilkan nilai yang berbeda tergantung dari nilai parameter yang dimasukkan pada saat pemanggilan fungsi tersebut. Berikut ini contoh program yang akan menunjukkan kegunaan dari parameter masukan.

```
#include <stdio.h>

#define pi 3.14159

/* Mendefinisikan suatu fungsi dengan parameter berjenis
   masukan */
double HitungKelilingLingkaran(int radius) {
    double k;
    k = 2 * pi * radius;
    return k;
}

/* Fungsi Utama */
int main(void) {
    int r;
    printf("Masukkan nilai jari-jari lingkaran : ");
    scanf("%d", &r);
    double keliling = HitungKelilingLingkaran(r);
    printf("Keliling lingkaran dengan jari-jari %d : %f", r,
           keliling);
    return 0;
}
```

Pada sintak program di atas, variabel *r* merupakan parameter aktual yang berfungsi sebagai parameter masukan karena variabel tersebut digunakan untuk menampung nilai yang akan menjadi masukan (*input*) untuk proses perhitungan di dalam fungsi **HitungKelilingLingkaran()**.

### Parameter Keluaran

Kebalikan dari parameter masukan, parameter keluaran adalah parameter yang digunakan untuk menampung nilai kembalian / nilai keluaran (*output*) dari suatu proses. Umumnya parameter jenis ini digunakan di dalam fungsi yang tidak mempunyai nilai balik. Untuk lebih memahaminya, perhatikan contoh program di bawah ini yang merupakan modifikasi dari program sebelumnya.

```

#include    <stdio.h>
#define PI 3.14159

/* Mendefinisikan fungsi yang mengandung parameter keluaran */
void HitungKelilingLingkaran(int radius, double *K) {
    *K = 2 * PI * radius;
}

/* Fungsi Utama */
int main(void) {
    int R;
    double Keliling;

    printf("Masukkan nilai jari-jari lingkaran : ");
    scanf("%d", &R);
    HitungKelilingLingkaran(R, Keliling);
    printf("Keliling lingkaran dengan jari-jari %d : %f", R,
        Keliling);

    return 0;
}

```

Pada sintak program di atas, variabel `Keliling` berfungsi sebagai parameter keluaran karena variabel tersebut digunakan untuk menampung nilai hasil dari proses yang terdapat di dalam fungsi. Sedangkan variabel `R` adalah variabel yang bersifat sebagai parameter masukan dimana nilainya digunakan untuk menampung nilai yang akan dilewatkan ke dalam fungsi. Adapun contoh hasil yang akan diberikan dari program di atas adalah seperti yang tertera di bawah ini.

#### *Parameter Masukan/Keluaran*

Selain parameter masukan dan keluaran, terdapat parameter jenis lain, yaitu parameter masukan/keluaran dimana parameter tersebut mempunyai dua buah kegunaan, yaitu sebagai berikut:

- ❑ Pertama parameter ini akan bertindak sebagai parameter yang menampung nilai masukan.
- ❑ Setelah itu, parameter ini akan bertindak sebagai parameter yang menampung nilai keluaran.

Berikut ini diberikan contoh program dimana di dalamnya terdapat sebuah parameter yang berperan sebagai parameter masukan/keluaran.



```
#include <stdio.h>
#define PI 3.14159

/* Mendefinisikan fungsi dengan parameter masukan/keluaran */
void HitungKelilingLingkaran(double *X) {
    *X = 2 * PI * (*X);
}

/* Fungsi Utama */
int main(void) {
    int R;
    double param;
    printf("Masukkan nilai jari-jari lingkaran : ");
    scanf("%d", &int);

    /* Melakukan typecast dari tipe int ke tipe double */
    param = (double) R;

    HitungKelilingLingkaran(&param);
    printf("Keliling lingkaran dengan jari-jari %d : %f", R,
           param);
    return 0;
}
```

### Melewatkan Parameter Berdasarkan Nilai (*Pass By Value*)

Terdapat dua buah cara untuk melewati parameter ke dalam sebuah fungsi, yaitu dengan cara melewati berdasarkan nilainya (*pass by value*) dan berdasarkan alamatnya (*pass by reference*). Namun pada bagian ini hanya akan dibahas mengenai pelewatan parameter berdasarkan nilai saja, sedangkan untuk pelewatan parameter berdasarkan alamat akan kita bahas pada sub bab berikutnya. Pada pelewatan parameter berdasarkan nilai, terjadi proses penyalinan (*copy*) nilai dari parameter formal ke parameter aktual. Hal ini akan menyebabkan nilai variabel yang dihasilkan oleh proses di dalam fungsi tidak akan berpengaruh terhadap nilai variabel yang terdapat di luar fungsi. Untuk lebih memahaminya, perhatikan contoh program berikut dimana di dalamnya terdapat sebuah parameter yang dilewatkan dengan cara *pass by value*.

```
#include <stdio.h>
/* Mendefinisikan fungsi dengan melewati parameter berdasarkan
nilai */
void TambahSatu(int X) {
    X++;
    /* Menampilkan nilai yang terdapat di dalam fungsi */
    printf("Nilai di dalam fungsi : %d\n", X);
}
```

```

/* Fungsi Utama */
int main(void) {
    int Bilangan;
    printf("Masukkan sebuah bilangan bulat : ");
    scanf("%d", &Bilangan);

    /* Menampilkan nilai awal */
    printf("\nNilai awal : %d\n", Bilangan);

    /* Memanggil fungsi TambahSatu */
    TambahSatu(Bilangan);

    /* Menampilkan nilai akhir */
    printf("Nilai akhir : %d\n", Bilangan);

    return 0;
}

```

Seperti yang kita lihat pada hasil program di atas bahwa nilai dari variabel **Bilangan** tetap bernilai 10 walaupun kita telah memanggil fungsi **TambahSatu()**. Hal ini disebabkan karena variabel **Bilangan** dan variabel **x** merupakan dua variabel yang tidak saling berhubungan dan menempati alamat memori yang berbeda sehingga yang terjadi hanyalah proses penyalinan (peng-copy-an) nilai dari variabel **Bilangan** ke variabel **x**. Dengan demikian, perubahan nilai variabel **x** tentu tidak akan mempengaruhi nilai variabel **Bilangan**.

### Melewatkan Parameter Berdasarkan Alamat (*Pass By Reference*)

Di sini, parameter yang dilewatkan ke dalam fungsi bukanlah berupa nilai, melainkan suatu alamat memori. Pada saat kita melewati parameter berdasarkan alamat, terjadi proses referensial antara variabel yang terdapat pada parameter formal dengan variabel yang terdapat parameter aktual. Hal tersebut menyebabkan kedua variabel tersebut akan berada pada satu alamat di memori yang sama sehingga apabila terdapat perubahan nilai terhadap salah satu dari variabel tersebut, maka nilai variabel satunya juga akan ikut berubah. Untuk melakukan hal itu, parameter fungsi tersebut harus kita jadikan sebagai *pointer* (akan dibahas pada bab terpisah).

Agar dapat lebih memahami materi ini, di sini kita akan menuliskan kembali kasus di atas ke dalam sebuah program. Namun, sekarang kita akan melakukannya dengan menggunakan cara *pass by reference*.

```

#include <stdio.h>

/* Mendefinisikan fungsi dengan melewati parameter berdasarkan
alamat */
void TambahSatu(int *X) {
    (*X)++;
    /* Menampilkan nilai yang terdapat di dalam fungsi */
    printf("Nilai di dalam fungsi : %d\n", *X);
}

/* Fungsi Utama */
int main(void) {
    int Bilangan;
    printf("Masukkan sebuah bilangan bulat : ");
    scanf("%d", &Bilangan);

    /* Menampilkan nilai awal */
    printf("\nNilai awal : %d\n", Bilangan);

    /* Memanggil fungsi TambahSatu dengan mengirimkan
alamat variabel Bilangan */
    TambahSatu(&Bilangan);

    /* Menampilkan nilai akhir */
    printf("Nilai akhir : %d\n", Bilangan);

    return 0;
}

```

## Rekursi

Rekursi adalah proses pemanggilan fungsi oleh dirinya sendiri secara berulang. Istilah ‘rekursi’ sebenarnya berasal dari bahasa Latin ‘recursus’, yang berarti ‘menjalankan ke belakang’. Rekursi digunakan untuk penyederhanaan algoritma dari suatu proses sehingga program yang dihasilkan menjadi lebih efisien. Pada bagian ini kita akan mempelajarinya langsung melalui contoh-contoh program.

### Menentukan Nilai Faktorial

Pada bagian ini kita akan membuat sebuah fungsi rekursif untuk menentukan nilai faktorial dengan memasukkan nilai yang akan dihitung sebagai parameter fungsi ini. Sebagai contoh apabila parameter yang kita masukkan adalah 5, maka hasilnya adalah

$$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$$

Proses tersebut dapat kita sederhanakan melalui fungsi matematis sebagai berikut.  $F! (N) = N * F! (N-1)$

Namun yang harus kita perhatikan di sini adalah  $F! (0) = 1$ , ini adalah suatu tetapan

numerik yang tidak dapat diubah. Berikut ini contoh implementasi kasus tersebut ke dalam sebuah program.

```
#include <stdio.h>

/* Mendefinisikan fungsi untuk menghitung nilai faktorial */
int Faktorial(int N) {
    if (N == 0) {
        return 1;
    } else {
        return N * Faktorial(N-1);
    }
}

int main(void) {
    int bilangan;
    printf("Masukkan bilangan yang akan dihitung : ");
    scanf("%d", &bilangan);
    printf("%d! = %d", bilangan, Faktorial(bilangan));
    return 0;
}
```

Konsep dari proses di atas sebenarnya sederhana, yaitu dengan melakukan pemanggilan fungsi **Faktorial()** secara berulang. Untuk kasus ini, proses yang dilakukan adalah sebagai berikut.

```
Faktorial(5)  = 5 * Faktorial(4)
Faktorial(4)  = 4 * Faktorial(3)
Faktorial(3)  = 3 * Faktorial(2)
Faktorial(2)  = 2 * Faktorial(1)
Faktorial(1)  = 1 * Faktorial(0)
Faktorial(0)  = 1
Faktorial(1)  = 1 * 1
Faktorial(2)  = 2 * 1
Faktorial(3)  = 3 * 2
Faktorial(4)  = 4 * 6
Faktorial(5)  = 5 * 24
               = 120
```

#### *Menentukan Nilai Perpangkatan*

Sekarang kita akan melakukan rekursi untuk menghitung nilai  $B^N$ , dimana B adalah bilangan basis dan N adalah nilai eksponen. Kita dapat merumuskan fungsi tersebut seperti di bawah ini.

$$B^N = B * B^{N-1}$$

Dengan demikian apabila kita implementasikan ke dalam program, maka sintaknya kurang lebih sebagai berikut.

```
#include <stdio.h>

/* Mendefinisikan fungsi untuk menghitung nilai eksponensial */
int Pangkat(int basis, int e) {
    if (e == 0) {
        return 1;
    }else {
        return basis * Pangkat(basis, e-1);
    }
}

int main(void) {
    int B, N;
    printf("Masukkan bilangan basis : "); scanf("%d", &B);
    printf("Masukkan bilangan eksponen : "); scanf("%d", &N);
    printf("%d^%d = %d", B, N, Pangkat(B, N));
    return 0;
}
```

Proses yang terdapat di atas adalah sebagai berikut.

$Pangkat(2, 5) = 2 * Pangkat(2, 4)$   
 $Pangkat(2, 4) = 2 * Pangkat(2, 3)$   
 $Pangkat(2, 3) = 2 * Pangkat(2, 2)$   
 $Pangkat(2, 2) = 2 * Pangkat(2, 1)$   
 $Pangkat(2, 1) = 2 * Pangkat(2, 0)$   
 $Pangkat(2, 0) = 1$   
 $Pangkat(2, 1) = 2 * 1$   
 $Pangkat(2, 2) = 2 * 2$   
 $Pangkat(2, 3) = 2 * 4$   
 $Pangkat(2, 4) = 2 * 8$   
 $Pangkat(2, 5) = 2 * 16$   
 $\quad = 32$

#### *Konversi Bilangan Desimal ke Bilangan Biner*

Kali ini kita akan membuat sebuah fungsi rekursif tanpa nilai balik yang digunakan untuk melakukan konversi bilangan desimal ke bilangan biner. Untuk informasi lebih detil mengenai bilangan biner dan heksadesimal, Anda dapat melihat lampiran B – Bit dan Byte di bagian akhir buku ini. Artinya, di sini kita tidak akan membahas bagaimana proses pengkonversian tersebut, melainkan kita lebih berkonsentrasi ke pembahasan mengenai rekursi.

Adapun sintak program untuk melakukan hal tersebut adalah sebagai berikut.

```
#include <stdio.h>

void DesimalKeBiner(int n) {
    if (n>1) {
        DesimalKeBiner(n/2);
    }
    printf("%d", n%2);
}

int main(void)
{
    int a;
    printf("Masukkan bilangan desimal yang akan dikonversi : ");
    scanf("%d",&a);
    printf("%d dalam biner : ", a)
    DesimalKeBiner(a);

    return 0;
}
```

### LATIHAN

1. Buatlah program untuk menyelesaikan perhitungan aritmatika di bawah ini dimana setiap proses yang dikerjakan diwakilkan oleh sebuah fungsi dengan nilai balik yang memiliki 2 variabel.

$$X = (A + B) * ((C - D) / E)$$

2. Buatlah sebuah program rekursif untuk melakukan konversi dari bilangan desimal ke *oktadecimal*!