Cepat Mahir Algoritma dalam C

I Putu Gede Darmawan
IPGD_BALI@yahoo.com

Rekursif

Pengenalan

Rekursif ialah salah satu teknik pemrograman dengan cara memanggil sebuah fungsi dari dirinya sendiri, baik itu secara langsung maupun tidak langsung. Pemanggilan fungsi rekursif secara langsung berarti dalam fungsi tersebut terdapat *statement* untuk memanggil dirinya sendiri sedangkan secara tidak langsung berarti fungsi rekursif tersebut memanggil 1 atau lebih fungsi lain sebelum memanggil dirinya sendiri.

```
[ret-val] coba([parameter])
{
 // statement
coba([parameter])
 // statement
}
```

Fungsi Rekursif Langsung

```
[ret-val] satu([parameter])
{
 // statement
 dua([parameter])
 // statement
}
```

```
[ret-val] dua([parameter])
{
 // statement
 satu([parameter])
 // statement
}
```

Fungsi Rekursif Tak Langsung

Rekursif tidak selalu lebih jelek daripada iteratif . Ada kalanya sebuah fungsi rekursif justru mempermudah penyelesaian masalah yang ditemui pada kasus iteratif (pengulangan) Kelemahan pada proses rekursif antar lain, memerlukan tempat penampungan stack yang cukup besar. Karena setiap kali pemanggilan fungsi , register – register seperti cs (untuk memory far) dan ip harus disimpan , belum lagi untuk penanganan local variable serta parameter fungsi yang tentunya membutuhkan ruang untuk stack lebih banyak lagi . Selain itu karena setiap pemanggilan fungsi , register dan memory harus di push ke stack maka setelah selesai pemanggilan perlu diadakannya pop stack . untuk mengembalikan memory dan register kembali ke keadaan awal , ini sering disebut sebagai *overhead* .

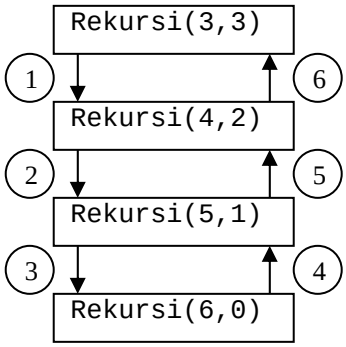
Proses Rekursif

Untuk dapat memahami proses yang terjadi dalam sebuah fungsi rekursif , marilah kita simak contoh fungsi rekursif berikut :

```
void rekursi(int a,int b)
{
 if (b == 0) return;
 a++;
 b--;
 printf("Masuk  -> a = %d || b = %d \n",a,b);
 rekursi(a,b);
 printf("Keluar -> a = %d || b = %d \n",a,b);
}
```

Misalkan Fungsi tersebut dipanggil dengan nilai a = 3 dan b = 3 maka pertama tama di cek apakah b = 0 (`if (b == 0) return`), jika sama maka keluar. Ternyata nilai b tidak sama dengan 0 maka tambahkan a dengan 1 dan kurangi b dengan 1 . Maka keadaan sekarang menjadi a = 4 dan b = 2 . Baris berikutnya menampilkan nilai a dan b ke layar (`printf("Masuk  -> a = %d || b = %d \n",a,b)`). Kemudian panggil fungsi rekursi dengan nilai a = 4 dan b = 2 . Langkah – langkah tersebut diulang terus sampai pemanggilan fungsi rekursi dengan nilai a = 6 dan b = 0. Pada saat ini kondisi if bernilai benar sehingga fungsi akan keluar (return) dan melanjutkan perintah setelah pemanggilan fungsi rekursi dengan a = 6 dan b = 0. Yaitu mencetak nilai a dan b (`printf("Keluar ->`

`a = %d || b = %d \n",a,b))`. Setelah mencetak nilai a dan b maka fungsi rekursif akan keluar lagi , dan melanjutkan perintah setelah pemanggilan fungsi rekursif sebelumnya dimana nilai a = 5 dan b = 1 . Demikian seterusnya sampai nilai a = 4 dan nilai b = 2. yang tidak lain pemanggilan fungsi rekurif yang pertama. Proses pemanggilan fungsi rekursif dapat diilustrasikan :



Langkah ke :
1. a = 4 ; b = 2 . Cetak : Masuk -> a = 4 || b = 2
2. a = 5 ; b = 1 . Cetak : Masuk -> a = 5 || b = 1
3. a = 6 ; b = 0 . Cetak : Masuk -> a = 6 || b = 0
4. a = 6 ; b = 0 , Cetak : Keluar -> a = 6 || b = 0
5. a = 5 ; b = 1 , Cetak : Keluar -> a = 5 || b = 1
6. a = 4 ; b = 2 , Cetak : Keluar -> a = 4 || b = 2

Untuk memperjelas lagi penggunaan fungsi rekursif dibawah ini akan di berikan contoh contoh program dengan menggunakan rekursif .

Menghitung Nilai Faktorial dan Fibonacci Dengan Rekursif

Untuk menghitung nilai faktorial bilangan bulat positif marilah kita daftarkan dulu nilai – nilai faktorial untuk mempermudah pengambilan algoritma .

0! = 1
1! = 1
2! = 2 x 1
3! = 3 x 2 x 1 = 3 x 2!
4! = 4 x 3 x 2 x 1 = 4 x 3!

} N! = N x (N – 1) !

Nah dari daftar diatas dapat dibuat fungsi rekursi untuk menghitung nilai faktorial ke n yaitu

$$Fakt(n) = \begin{cases} n > 1 \rightarrow n * Fakt(n-1) \\ n == 0 \,||\, n == 1 \rightarrow 1 \end{cases}$$

Jika di terjemahkan ke dalam bahasa C menjadi :

```
int Fakt(int n)
{
        if (n == 1 || n == 0) return 1;
        return n * Fakt(n-1);
}
```

Untuk Mencari Bilangan Fibonacci juga sama . Sebelumnya mari kita daftarkan dulu bilangan Fibonacci untuk mempermudah pencarian algoritma .

1 1 2 3 5 8 13 ...

Dapat dilihat bahwa
Fibo(0) = 1
Fibo(1) = 1
Fibo(2) = Fibo(1) + Fibo(0) = 1 + 1 = 2
Fibo(3) = Fibo(2) + Fibo(1) = 2 + 1 = 3

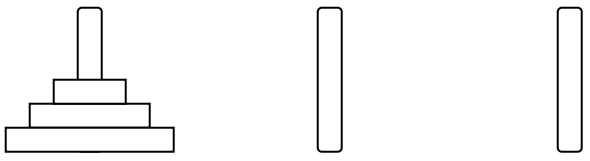
Maka akan didapat rumusan untuk bilangan Fibonacci berupa :

$$Fibo(n) = \begin{cases} n > 1 \rightarrow Fibo(n-1) + Fibo(n-2) \\ n == 1 \,||\, n == 0 \rightarrow 1 \end{cases}$$

Untuk Listing Program dalam bahasa C saya serahkan ke pembaca sebagai latihan.

Menara Hanoi

Menara Hanoi ialah salah satu permainan yang dulunya dimainkan oleh seorang pendeta di Hanoi . Tujuan permainan ini ialah memindahkan n buah pringan dari tonggak asal (A) melalui tonggak bantu (B) menuju tonggak tujuan (C) .Dengan aturan – aturan bahwa piringan yang lebih kecil tidak boleh berada di bawah piringan yang lebih besar .



Menara Hanoi dengan 3 buah piringan

Seperti biasa untuk memecahkan masalah kita daftarkan dulu langkah – langka yang diambil mulai n = 1. Dengan dfinisi piringan yang paling atas ialah piringan 1.
Untuk n = 1 :

- Pindahkan piringan 1 dari A ke C

Untuk n = 2 :

- Pindahkan piringan 1 dari A ke B
- Pindahkan piringan 2 dari A ke C
- Pindahkan piringan 3 dari B ke C

Dari contoh diatas dapat diambil kesimpulan , untuk memindahkan N piringan dari tonggak asal (A) ke tonggak tujuan (C) maka piringan ke N harus berada di tonggak tujuan (C) , sedangkan piringan ke 1 sampai (N - 1) harus berada di tonggak bantu (B). Setelah piringan ke 1 sampai (N-1) berada pada tonggak bantu (B) , kemudian pindahkan piringan ke 1 sampai (n-1) tersebut dari tonggak bantu (B) ke tonggak tujuan (C) **Nah** bagaimana caranya membawa piringan ke 1 sampai (N-1) dari tonggak asal ke tonggak bantu (B) , caranya sama saja yaitu dengan memindahkan piringan ke (n-1) dari tonggak asal (A) ke tonggak tujuan yang pada saat ini berada pada tonggak (B) sedangkan piringan dari 1 sampai ke (N-2) dipindahkan ke tonggak bantu yang saat ini berada di tonggak (C) , Setelah piringan ke 1 sampai (N-2) berada pada tonggak bantu (C) , kemudian pindahkan piringan ke 1 sampai (N-2) ke tonggak tujuan (B) dan seterusnya.
Metode penyelesaian permainan Hanoi diatas sering disebut sebagai metode back tracking , jadi solusi dicari dengan cara mundur ke belakang dahulu , baru kemudian solusi bisa di temukan .

Berikut Listing program Menara Hanoi dalam bahasa C

```
#include <stdio.h>

void Hanoi(int n,char asal,char bantu,char tujuan)    // pindahkan piringan ke n
{                                                     // dari asal menuju tujuan
                                                      // melalui bantu
        if (n == 0) return;
        Hanoi(n-1,asal,tujuan,bantu);                 //pindahkan piringan ke n-1
                                                      // dari asal ke bantu melalui
                                                      //tonggak tujuan
        printf("Pindahkan piringan ke %d ke dari %c ke %c\n",n,asal,tujuan);

        Hanoi(n-1,bantu,asal,tujuan);                 //pindahkan piringan ke n – 1
                                                      // dari bantu menuju tujuan
                                                      // melalu asal
}

int main(void)
{
        int n;
        printf("Jumlah piringan ? ");
        scanf("%d",&n);
        Hanoi(n,'a','b','c');
        return 0;
}
```

Berikut Hasil dari program diatas .

```
Jumlah piringan ? 3
Pindahkan piringan ke 1 ke dari a ke c
Pindahkan piringan ke 2 ke dari a ke b
Pindahkan piringan ke 1 ke dari c ke b
Pindahkan piringan ke 3 ke dari a ke c
Pindahkan piringan ke 1 ke dari b ke a
Pindahkan piringan ke 2 ke dari b ke c
Pindahkan piringan ke 1 ke dari a ke c
```