# **Desain dan Analisis Recursive Function (TOWER OF HANOI)**

## **Pseudo Code:**

Berikut merupakan pseudocode dari tower of hanoi:

FUNCTION tower\_of\_hanoi(n, t1, t2, t3)

IF n == 1 THEN

PRINT "Pindahkan DISK " + n + " DARI " t1 + " KE " t3

RETURN

END IF

CALL tower\_of\_hanoi(n-1, t1, t3, t2)

PRINT "Pindahkan DISK " + n + " DARI " t1 + " KE " t3

CALL tower\_of\_hanoi(n-1, t2, t1, t3)

END FUNCTION

BEGIN PROGRAM

DECLARE disc AS INTEGER

PRINT "Masukkan Jumlah Disc: "

READ input TO disc

CALL tower\_of\_hanoi(disc, "T1", "T2", "T3")

END PROGRAM

## 

## **Penjelasan Langkah:**

Secara garis besar, apabila program ini memiliki **base case** apabila **n** mencapai nilai 1, maka akan dilakukan pemberhentian pemanggilan fungsi rekursif. Fungsi **tower\_of\_hanoi** kita tidak melakukan perubahan pada diska yang tersedia sebagai parameter **n,** namun pada fungsi ini kita melakukan perubahan bagaimana tiang tersebut yang harus berpindah. Secara struktur angka akan tetap perulangan dari n sampai 1, tetapi yang berubah adalah bagaimana cara kita menempatkan tiang sehingga parameter **n** seolah berpindah. Berikut Merupakan rincian dari bagaimana program ini bekerja baris per baris.

* MAIN PROGRAM
  + DECLARE disc AS INTEGER
    - deklarasi disc sebagai integer agar kita bisa melakukan custom disc untuk membuat penyelesaian tower of hanoi
  + PRINT "Masukkan Jumlah Disc: "
    - menampilkan perintah untuk memasukkan jumlah disc
  + READ input TO disc
    - membaca hasil input dan memasukkannya kedalam variable disc
  + CALL tower\_of\_hanoi(disc, "T1", "T2", "T3")
    - Pemanggilan fungsi tower\_of\_hanoi
* FUNCTION
  + IF n == 1 THEN
  + PRINT "Pindahkan DISK " + n + " DARI " t1 + " KE " t3
  + RETURN
  + END IF
    - Base case pada fungsi rekursif agar program tidak berjalan selamanya
  + CALL tower\_of\_hanoi(n-1, t1, t3, t2)
    - pada baris ini kita memanggil kembali metode tower\_of\_hanoi dengan memasukkan n dikurang oleh satu angka dan juga kita tukar parameter t3 dan t2
  + PRINT "Pindahkan DISK " + n + " DARI " t1 + " KE " t3
    - pada baris ini, kita akan lakukan printing saat iterasi mencapai **n** dan pada proses pemindahan dari variable **t1** ke **t3** (fungsi rekursif ini memodifikasi cara pemanggilan tiang)
  + CALL tower\_of\_hanoi(n-1, t2, t1, t3)
    - Pada baris ini kita melakukan fungsi rekursif kembali namun dengan perubahan parameter antara **t1** dan **t2**

Kami juga mencantumkan kode C agar bisa dicoba, dari kode tersebut maka akan menghasilkan output seperti table dibawah

|  |  |
| --- | --- |
| 1 | Pindahkan disk 1 dari T1 ke T2 |
| 2 | Pindahkan disk 2 dari T1 ke T3 |
| 3 | Pindahkan disk 1 dari T2 ke T3 |
| 4 | Pindahkan disk 3 dari T1 to T2 |
| 5 | Pindahkan disk 1 dari T3 ke T1 |
| 6 | Pindahkan disk 2 dari T3 to T2 |
| 7 | Pindahkan disk 1 dari T1 ke T2 |
| 8 | Pindahkan disk 4 dari T1 ke T3 |
| 9 | Pindahkan disk 1 dari T2 ke T3 |
| 10 | Pindahkan disk 2 dari T2 ke T1 |
| 11 | Pindahkan disk 1 dari T3 ke T1 |
| 12 | Pindahkan disk 3 dari T2 ke T3 |
| 13 | Pindahkan disk 1 dari T1 ke T2 |
| 14 | Pindahkan disk 2 dari T1 to T3 |
| 15 | Pindahkan disk 1 dari T2 ke T3 |

## **Analisis Kompleksitas:**

Kompleksitas kode tersebut bekerja secara rekursif dengan pemanggilan fungsi rekursif sebanyak dua kali, perlu diperhatikan juga bahwa kita melakukan operasi pengurangan n dengan angka 1 sebagai iterasi diska yang ada (n-1). Dari hal Itu Kita bisa simpulkan bahwa Kompleksitas Program ini memiliki notasi yang mana untuk banyaknya diska sebanyak 4 akan dibutuhkan iterasi sebanyak 15 kali

# 

# **Analisis Kesalahan pada Implementasi Recursive Function**

Diberikan cuplikan Pseudocode seperti lampiran di bawah ini:

|  |
| --- |
| Function factorial(n):  if n == 0:  return 1  else:  return n \* factorial(n) |

Kami menemukan kesalahan dari pseudocode tersebut, kode tersebut memiliki tujuan untuk mendapatkan hasil nilai faktorial dari parameter **n** yang di input. Terdapat beberapa kesalahan yang kami lihat adalah pemanggilan fungsi rekursif yang secara gamblang langsung memasukkan isi dengan parameter awal. hal ini mengakibatkan fungsi **factorial** akan terus berjalan tanpa mencapai base case dikarenakan pemanggilan kembali suatu fungsi hanya menurunkan parameter yang sama, hal itu mengakibatkan pemanggilan tanpa henti. Kita Juga bisa merubah kondisi **n == 0** menjadi **n == 1,** dikarenakan perhitungan factorial hanya sampai 1; hal ini tidak akan berdampak apa apa karena hal yang di return merupakan value **1**

Untuk Memperbaiki Kode tersebut dapat dilakukan dengan cara merubah **factorial(n)** menjadi **factorial(n - 1).** Dengan begini kode akan berjalan dengan mulus sesuai dengan ekspektasi yang diharapkan:

|  |
| --- |
| Function factorial(n):  if n == 0:  return 1  else:  return n \* factorial(n - 1) |