

Nama : Ahamad Ruslandia Papua

NIM : 13020200002

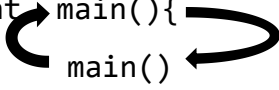
Teori Dasar Rekursif

Rekursif/ Recursion adalah sebuah fungsi dimana fungsi tersebut adalah pengulangan yang dapat mengulang atau dapat memanggil dirinya sendiri. Rekursif sendiri akan mengulangi fungsi dirinya sendiri dan tidak akan di ketahui kapan akan berakhir dari fungsi rekursif tersebut.

Contoh :

Fungsi Rekursif

```
int main(){  
    main()  
    return 0  
}
```



Proses fungsi Rekursif dari (main) tersebut akan mengulang fungsinya sendiri dan tidak ketahu pasti kapan akan berakhir, Sangat disarankan agar menggunakan Finite Recursion/ Rekursif Terbatas agar proses Rekursif tersebut bisa dibatasi dan berakhir.

Program Finite Recursion/ Rekursif Terbatas Menggunakan Fungsi Looping

```
#include <iostream>  
using namespace std;  
  
int bilangan_berpangkat(int a, int b){  
    int hasil a;  
  
    //Fungsi Looping  
    for(int i = 1; i < b; i++){  
        hasil = hasil * a;  
    }  
}
```

```

        return hasil;
    }
    int main(){
        int a;
        int b;

        cout << "Angka : ";
        cin >> a;
        cout << "Pangkat : ";
        cin >> b;

        cout << "Hasil : " << bilangan_berpangkat(a,b) << endl;

        cin.get();

        return 0;
    }

```

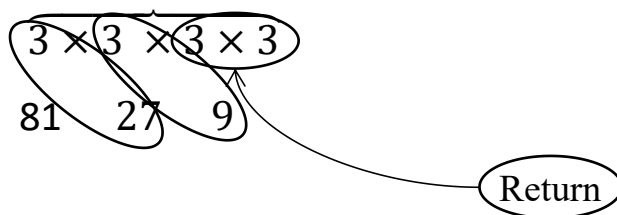
Hasil Output Program :

```

Angka : 3
Pangkat : 4
Hasil : 81

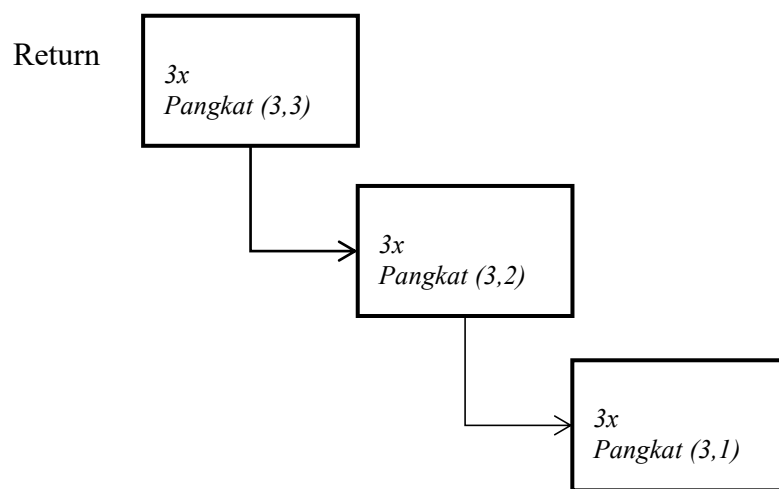
```

Proses Finite Recursion :



Proses Finite Recursion :

Pangkat (3, 4) = (a, b)



Penjelasan :

Simbol perkalian (\times) tersebut adalah return dalam sebuah program, Apabila kita menginput sebuah angka dalam suatu program yang telah di buat sebagai contoh kita kita akan menginput angka 3 maka angka tersebut akan tersimpan ke variabel int a, Selanjutnya jika kita menginput pangkat 4 maka angka tersebut akan tesimpan ke variabel int b. int a akan menglang dirinya sendiri hingga batas akhir pangkat yang kita masukkan pada int b.

Menghitung Kompleksitas Waktu Dari Algoritma Rekursif

Bentuk rekursif :

- Suatu subrutin/fungsi/ prosedur yang memanggil dirinya sendiri.
- Bentuk dimana pemanggilan subrutin terdapat dalam body subrutin
- Dengan rekursi, program akan lebih mudah dilihat

Bentuk rekursi bertujuan untuk :

- Menyederhanakan penulisan program
- Menggantikan bentuk iterasi

Syarat bentuk rekursif:

- Mempunyai kondisi terminal (basis)
- *Subroutine call* yang melibatkan parameter yang nilainya menuju kondisi terminal (*recurrence*)

Menghitung kompleksitas bentuk rekursif

Untuk bentuk rekursif, digunakan teknik perhitungan kompleksitas dengan relasi rekurens.

Contoh :

Menghitung Faktorial

```
Function Faktorial (input n : integer) → integer
{menghasilkan nilai n!, n tidak negatif}
Algoritma
  If n=0 then
    Return
  Else
    Return n*faktorial (n-1)
  Endif
```

Kompleksitas waktu :

- untuk kasus basis, tidak ada operasi perkalian $\rightarrow (0)$
- untuk kasus rekurens, kompleksitas waktu diukur dari jumlah perkalian (1)
ditambah kompleksitas waktu untuk faktorial (n-1)

Relasi rekurens :

$$T(n) = \begin{cases} 0 & , n = 1 \\ T(n-1) + 1 & , n > 0 \end{cases}$$

Kompleksitas waktu :

$$\begin{aligned} T(n) &= 1 + T(n-1) \\ &= 1 + 1 + T(n-2) = 2 + T(n-2) \\ &= 2 + 1 + T(n-3) = 3 + T(n-3) \\ &= \dots\dots\dots \\ &= \dots\dots\dots \\ &= n + T(0) \\ &= n + 0 \end{aligned}$$

Jadi $T(n) = n$

$$T(n) \in O(n)$$

Rekursif Dengan Memoisasi

Memoisasi adalah teknik untuk mengimplementasikan pemrograman dinamis untuk membuat algoritma rekursif efisien. Banyak manfaat yang sama seperti pemrograman dinamis tanpa harus memerlukan perubahan besar pada algoritma rekursif yang lebih alami. Fungsi rekursif sendiri yaitu melakukan perhitungan yang sama dengan input yang sama berkali-kali, ini berarti proses ini bisa memakan waktu lebih lama dari pada alternatif yang berulang.

Dasar Memoisasi :

- Hal pertama adalah merancang algoritma rekursif
- Apabila rekursif mengulang dirinya berulang kali, maka algoritma rekursif yang tidak efisien dapat menyimpan subproblem ini dalam tabel sehingga mereka tidak harus dikompilasi ulang.

Penerapan :

Untuk menerapkan memoisasi ke algoritma rekursif, Tabel tersebut akan dipertahankan dengan solusi subproblem, tetapi struktur kontrol untuk mengisi tabel terjadi selama eksekusi normal pada algoritma rekursif. Hal ini dapat diterapkan dalam langkah-langkah berikut:

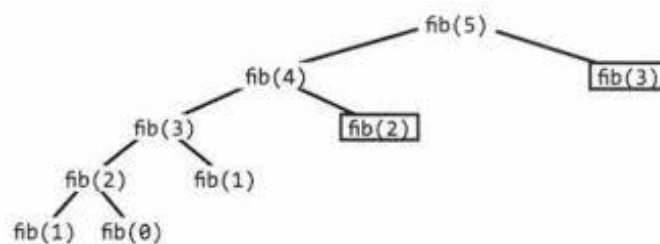
1. Algoritma rekursif memoisasi mempertahankan entri dalam tabel untuk solusi dari masing-masing subproblem.
2. Setiap entri tabel awalnya berisi nilai khusus untuk menunjukkan bahwa entri belum diisi.
3. Ketika subproblem pertama kali ditemui, solusinya dihitung dan akan disimpan dalam tabel.
4. Selanjutnya, nilai tersebut didongak daripada dihitung.

Untuk mengilustrasikan langkah-langkah berikut, mari kita ambil contoh untuk menghitung angka Fibonacci ke-n dengan algoritma rekursif sebagai berikut :

```
// Tanpa Memoisasi
static int fib(int n) {
    if (n == 0 || n == 1) return n;

    return fib(n - 1) + fib(n - 2);
}
```

Waktu Run untuk algoritma di atas adalah sekitar $O(2^n)$. Hal ini dapat divisualisasikan untuk fib (5) seperti gambar di bawah ini :



Fibonacci 5

Kita dapat melihat nilai yang sama pada gambar pohon di atas (misalnya fib (1), fib (0)..) sedang dihitung berulang kali. Bahkan, setiap kali kita menghitung fib (i), kita hanya bisa cache hasil ini dan menggunakannya nanti. Jadi, ketika kita memanggil fib (n), kita tidak perlu melakukan lebih dari $O(n)$ panggilan, karena hanya ada nilai-nilai yang mungkin dapat kita lemparkan pada fib (n).

```
// Menggunakan Memoisasi
static int fibonacciMemo(int n) {
    // tabel entry ke cache menghitung nilai
    int[] fibs = new int[n + 1];
    // memualai tabel entry menggunakan -1 untuk mengatakan
    nilai harus di hitung
    Arrays.fill(fibs, -1);
}
```



```
        return fib(n, fibs);
    }

    static int fib(int n, int[] fibs) {
        if (n == 0 || n == 1) return n;

        if (fibs[n] == -1) {
            fibs[n] = fib(n - 1, fibs) + fib(n - 2, fibs);
        }

        return fibs[n];
    }
}
```

Keuntungan memoisasi :

- Algoritma tidak harus diubah menjadi algoritma berulang.
- Menawarkan efisiensi yang sama (atau lebih baik) seperti pendekatan pemrograman dinamis biasa.