Rekursi

BAB 5

Materi

5.1 Rekursi Dasar

- Rekursi Komputasi 4!
- Implementasi Komputasi Faktorial Rekursi
- Implementasi Komputasi Faktorial Iteratif

5.2 Rekursi Tail

- Rekursi Tail Komputasi 4!
- Implementasi Komputasi Faktorial Rekursi Tail

5.3 Kesimpulan

Pengantar

Rekursi adalah konsep pengulangan yang penting dalam ilmu komputer.

Konsep ini dapat digunakan untuk merumuskan solusi sederhana dalam sebuah permasalahan yang sulit untuk diselesaikan secara iteratif dengan menggunakan *loop for, while do*.

Pada saat tertentu konsep ini dapat digunakan untuk mendefinisikan permasalahan dengan konsisten dan sederhana.

Pada saat yang lain, rekursi dapat membantu untuk mengekspresikan algoritma dalam sebuah rumusan yang menjadikan tampilan algoritma tersebut mudah untuk dianalisa.

5.1 Rekursi Dasar

Rekursi mempunyai arti suatu proses yang bias memanggil dirinya sendiri.

Dalam sebuah rekursi sebenarnya tekandung pengertian sebuah prosedur atau fungsi.

Perbedaannya adalah bahwa rekursi bisa memanggil dirinya sendiri, kalau prosedur atau fungsi harus diipanggil melalui pemanggil prosedur atau fungsi.

Untuk memulai bahasan rekursi, kita membahas sebuah masalah sederhana yang kemungkinan kita tidak berpikir untuk menyelesaikan dengan cara rekursif.

Yaitu permasalahan faktorial, yang mana kita menghitung hasil faktorial dari sebuah bilangan, yaitu n.

Faktorial dari n (ditulis n!), adalah hasil kali dari bilangan tersebut dengan bilangan di bawahnya, di bawahnya hingga bilangan 1.

Contoh : Rekursi Komputasi 4!

Sebagai contoh, 4! = (4)(3)(2)(1).

Salah satu cara untuk menghitung adalah dengan menggunakan loop, yang mengalikan masing-masing bilangan dengan hasil sebelumnya. Penyelesaian dengan cara ini dinamakan iteratif, yang mana secara umum dapat didefinisikan sebagai berikut:

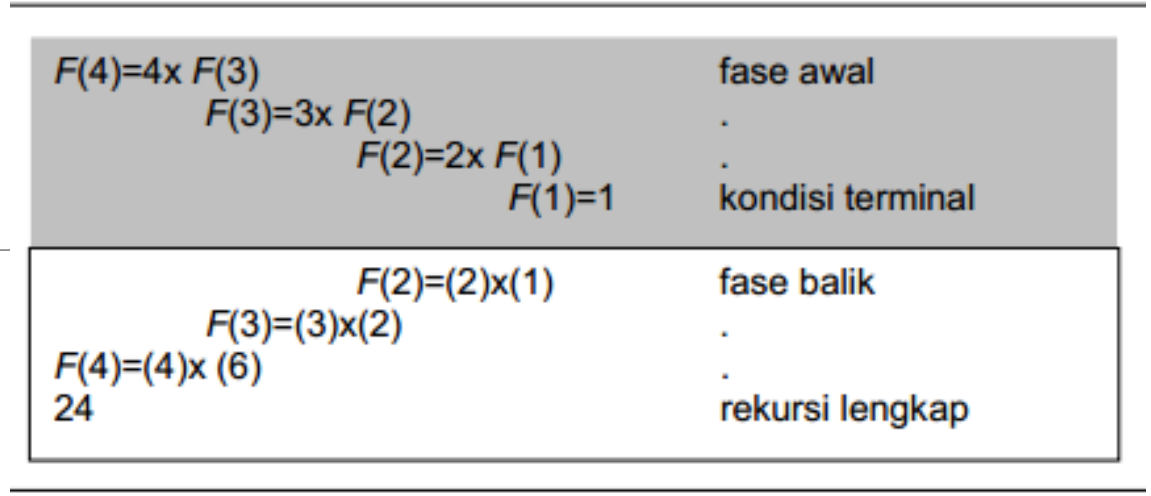
$$F(n) = \begin{cases} 1 & \text{jika } n=0,\ n=1 \\ nF(n-1) & \text{jika } n>1 \end{cases}$$

n! = (n)(n-1)(n-2) …

Cara lain untuk menyelesaikan permasalahan di atas adalah dengan cara rekursi, dimana n! adalah hasil kali dari n dengan (n-1)!.

Untuk menyelesaikan (n-1)! adalah sama dengan n!, sehingga (n-1)! adalah n-1dikalikan dengan (n-2)!, dan (n-2)! adalah n-2 dikalikan dengan (n-3)! dan seterusnya sampai dengan n = 1, kita menghentikan penghitungan n!

Contoh (Lanjutan)



Dalam gambar tersebut juga digambarkan dua fase dasar dari sebuah proses rekursi:

fase awal dan fase balik. Dalam fase awal, masing-masing proses memanggil dirinya sendiri. Fase awal ini berhenti ketika pemanggilan telah mencapai kondisi terminal.

Sebuah kondisi teminate adalah kondisi dimana sebuah fungsi rekursi kembali dari pemanggilan, artinya fungsi tersebut sudah tidak memanggil dirinya sendiri dan kembali pada sebuah nilai.

Sebagai contoh, dalam penghitungan faktorial dari n, kondisi terminal adalah n = 1, n = 0. Untuk setiap fungsi rekursi, minimal harus ada satu kondisi terminal.

Setelah fase awal selesai, kemudian proses melanjutkan pada fase balik, dimana fungsi sebelumnya akan dikunjungi lagi dalam fase balik ini. Fase ini berlanjut sampai pemanggilan awal, hingga secara lengkap proses telah berjalan.

Implementasi Komputasi Faktorial Rekursi

```
int fact_rec(int n)
{ if (n < 0)
return 0;
else if (n == 0)
return 1;
else if (n == 1)
return 1;
else
return n * fact(n-1); }
```

dalam C, fact_rec, dengan parameter sebuah bilangan n dan menghitung faktorial secara rekursi.

Fungsi tersebut bekerja sebagai berikut. Jika n kurang dari 0, maka fungsi kembali ke 0, menunjukkan kesalahan. Jika n = 0 atau 1, fungsi kembali ke 1 karena 0! dan 1! hasilnya 1.

Dan ini adalah kondisi terminal. Selainnya itu, fungsi kembali dengan hasil n kali factorial dari n-1.

Faktorial dari n-1 adalah penghitungan kembali secara rekursif dengan memanggil fact lagi.

Adanya kondisi terminal yang kita cantumkan dalam program tersebut, yaitu jika n = 0 atau n = 1.

Hal ini harus kita lakukan dalam setiap fungsi rekursif. Jika tidak, maka eksekusi tidak akan pernah berhenti. Akibatnya memori tumpukan akan habis dan Komputer akan hang.

Implementasi Komputasi Faktorial Iteratif (Menggunakan Looping Sederhana)

```
int fact_it (int n) {
int temp;
temp = 1;
if (n < 0)
        return 0;
else if (n == 0)
        return 1;
else if (n == 1)
        return 1;
else
        for (i=2; i<=n; ++i)
        temp = temp * i;
return (temp); }
```

5.2 Rekursi Tail

Sebuah proses rekursi dikatakan rekursi tail jika pernyataan terakhir yang akan dieksekusi berada dalam tubuh fungsi dan hasil yang kembali pada fungsi tersebut bukanlah bagian dari fungsi tersebut.

Ciri fungsi rekursi tail adalah fungsi tersebut tidak memiliki aktivitas selama fase balik.

Ciri ini penting, karena sebagian besar compiler modern secara otomatis membangun kode untuk menuju pada fase tersebut.

Ketika kompiler mendeteksi sebuah pemanggilan yang mana adalah rekursi tail, maka kompiler menulis aktivitas yang ada sebagai sebuah record yang dimasukkan ke dalam stack.

Kompiler dapat melakukan hal tersebut karena pemanggilan rekursi adalah pernyataan terakhir yang dieksekusi dalam aktivitas yang sedang berlangsung, sehingga tidak ada aktivitas yang harus dikerjakan pada saat pemanggilan kembali.

5.2 Rekursi Tail (Lanjutan)

Untuk memahami bagaimana sebuah rekursi tail bekerja, kita akan kembali membahas tentang penghitungan komputasi dari faktorial secara rekursi.

Sebagai langkah awal, akan sangat membantu jika kita memahami alasan dalam definisi awal yang bukan rekursi tail.

Dimana dalam definisi tersebut, penghitungan n! adalah dengan mengalikan n dengan (n-1)! dalam setiap aktivitas, yang mana hal tersebut terus diulang dari n = n –1 sampai n = 1.

Definisi ini bukanlah rekursi tail karena nilai yang dikembalikan dalam setiap aktivitas bergantung pada perkalian n dengan nilai yang dikembalikan oleh aktivitas sesudahnya.

Oleh karena itu, pencatatan aktivitas dari masing-masing pemanggilan harus diingat dalam stack sampai nilai-nilai yang dikembalikan dalam aktivitas-aktivitas sesudahnya telah terdefinisi.

Rekursi Tail Komputasi 4!

$$F(n,a) = \begin{cases} a & \text{jika } n=0,\ n=1 \\ F(n-1,na) & \text{jika } n>1 \end{cases}$$

$F(4,1)=F(3,4)$		fase awal
$F(3,4)=F(2,12)$		.
$F(2,12)=F(1,24)$		.
$F(1,24)=24$		kondisi terminal
	24	fase balik rekursi lengkap

Pada definisi ini digunakan parameter kedua, yaitu a, yang mana merupakan nilai utama dari penghitungan faktorial secara rekursif.

Hal ini mencegah kita untuk mengalikan nilai yang dikembalikan dalam setiap aktivitas dengan n. Dalam masingmasing pemanggilan rekursi kita mendefinisikan **a = na** dan **n = n – 1**. Kita melanjutkan sampai **n = 1**, sebagai kondisi terminal.

Dalam gambar diatas menggambarkan proses komputasi **4!** dengan menggunakan rekursi tail.

Perhatikan bahwa di sana tidak ada aktivitas setelah fase balik.

Implementasi Komputasi Faktorial Rekursi Tail

Program disamping mempresentasikan sebuah fungsi dalam bahasa C, yang menerima sebuah bilangan **n** dan menghitung faktorialnya dengan menggunakan **rekursi tail**.

Fungsi ini juga menerima parameter tambahan yaitu a, untuk initial pertama diset 1.

Fungsi ini hampir sama dengan fact pada Program Rekursi Awal, kecuali dia menggunakan a untuk mengelola lebih jauh nilai dari komputasi faktorial dalam proses rekursi.

```
int facttail(int n, int a)
{
if (n < 0)
        return 0;
else if (n == 0)
        return 1;
else if (n == 1)
        return a;
else
        return facttail(n-1,n*a);
}
```

Contoh Program Faktorial

```cpp
#include <iostream>
using namespace std;

long rekursiffaktorial(int f)
{
    if (f == 0)

        return 1;

    else

        return f * rekursiffaktorial(f - 1);
}
int main()
{
    int x;
        cout<<"Masukan Angka yang akan difaktorialkan : ";
    cin>>x;
    cout << x <<"! = " << rekursiffaktorial(x) <<endl;
    return 0;
}
```

```
Masukan Angka yang akan difaktorialkan : 9
9! = 362880

--------------------------------
Process exited after 3.854 seconds with return value 0
Press any key to continue . . .
```

5.3 Kesimpulan

1. Rekursi adalah kemampuan suatu rutin untuk memanggil dirinya sendiri.

2. Penggunaan sebuah rekursi harus mendefinisikan kondisi terminal, jika tidak eksekusi program tidak pernah berhenti, sehingga penggunaan memori tumpukan habis dan komputer akan hang.

3. Dalam beberapa situasi, pemecahan secara rekursif mempunyai keuntungan dan kekurangan yang dapat diperbandingkan dengan cara iteratif.

Terima Kasih

Pertanyaan?