FUNGSI REKURSIF 11

Tujuan Instruksi Umum (TIU):

Mampu memahami konsep pemrograman modular dan mengetahui fungsi-fungsi standar yang disediakan oleh bahasa C sehingga mampu mengimplementasikannya dalam pembuatan program dengan menggunakan bahasa C untuk permasalahan yang cukup kompleks.

Tujuan Instruksi Khusus (TIK):

- Dapat memahami apa yang dimaksud dengan fungsi rekursif
- Memahami konsep rekursi dan dapat mengimplementasikannya dalam pemrograman, khususnya dalam bahasa C

Pembahasan Materi:

- Pengertian Rekursif
- Perbandingan antara rekursi dan iteratif

11.1 Pengertian Rekursi

Rekursi adalah suatu proses dari fungsi yang memanggil dirinya sendiri. Fungsi yang seperti ini disebut fungsi rekursif (*recursive function*). Dalam sebuah fungsi rekursif pemanggilan dapat terjadi berulang kali. Karena ada proses yang berulang-ulang maka harus ada suatu kondisi yang mengakhiri prosesnya. Jika tidak, maka proses tidak akan pernah berhenti sampai memori yang digunakan tidak dapat menampung lagi.

Pemecahan masalah dengan pendekatan rekursif dapat dilakukan jika masalah tersebut dapat didefinisikan secara rekursif, yaitu masalah dapat diuraikan menjadi masalah sejenis yang lebih sederhana.

Listing Program 11.1 Contoh Fungsi Rekursi

```
#include<stdio.h>

void rekursi(int n);

main()
{     int x=3;
      rekursi(x);   /* Pemanggilan Fungsi */
}

void rekursi(int n)
{     static int j=0;
      if(n<=0) return;                    /* Kondisi Perhentian pemanggilan fungsi kembali */
      printf("rekursi ke-%d\n",++j);
      rekursi(n-1);                       /* Pemanggilan fungsi rekursi di dalam fungsi rekursi */
}
```

Hasil Running:

```
Rekursi ke-1
Rekursi ke-2
Rekursi ke-3
```

Dalam membuat fungsi rekursi harus ditentukan kondisi perhentian. Pada contoh listing program 11.1 di atas kondisi perhentiannya adalah jika nilai **n** sudah lebih kecil atau sama dengan 0. Setiap kali fungsi memanggil dirinya sendiri, nilai dari **n** dikurangi dengan nilai 1, sehingga nilai **n** akhirnya akan menjadi nol dan proses rekursi akan diakhiri, sehingga fungsi ini akan memanggil dirinya sendiri sebanyak **n** kali.

Contoh untuk menggambarkan fungsi rekursif yang sering digunakan adalah fungsi untuk mendapatkan nilai faktorial dari suatu bilangan bulat.

Listing Program 11.2 Program Faktorial

```
#include <iostream.h>

long factorial(long a);

int main ()
{
   int n;
   long hasil;
   printf("Menghitung N Faktorial (N!)\n");
   printf("Masukkan N : ");scanf(“%d”, &n);

   hasil= factorial(n);  /* Pemanggilan fungsi factorial */

   printf(“%d ! = %d”, n, hasil);
   return 0;
}

long factorial (long a)
{
  if (a > 1)
      return (a * factorial (a-1));
  else
      return (1);
}
```

Hasil Running:

```
Menghitung N Faktorial (N!)
Masukkan N : 5
5 ! = 120
```

Program 11.2 di atas dapat dijelaskan sebagai berikut:

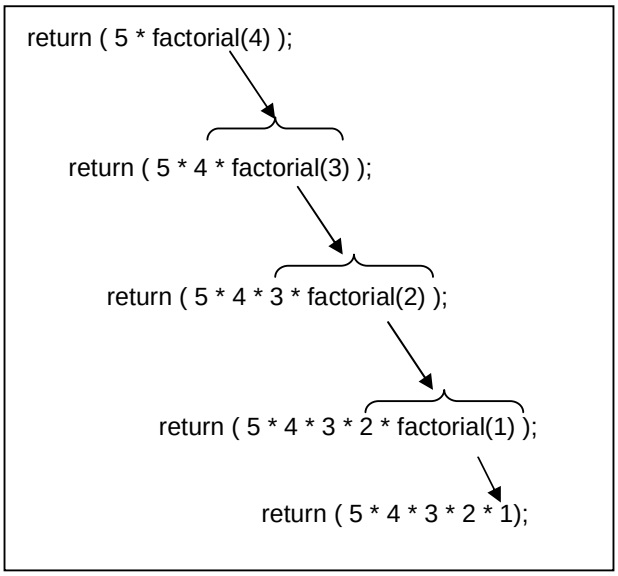
1. Fungsi utama memanggil fungsi **factorial()** dengan mengirimkan nilai n = 5 untuk parameter formal **a**, yang maksudnya akan dilakukan perhitungan sebanyak 5 faktorial.

2. Jika nilai dari **a** pertama kali yang diberikan oleh fungsi utama bernilai kurang dari atau sama dengan satu, maka hasil faktorial yang akan diberikan adalah bernilai 1.
3. Jika nilai **a** pertama kali yang diberikan oleh fungsi utama lebih besar dari 1, maka proses rekursi akan dilakukan misalnya nilai **a** = 5 , maka proses rekursi yang pertama adalah:

```
return ( 5 * factorial(4) );
```

Proses ini akan memanggil kembali fungsi dirinya sendiri dengan mengirimkan nilai 4 sebagai nilai **a** yang baru. Karena nilai **a** masih lebih besar dari 1 maka proses rekursi kedua akan dilakukan dengan hasilnya adalah 4 * factorial(3). Untuk **a** adalah 3, hasil yang diperoleh oleh rekursi adalah 3 * factorial(2) dan seterusnya sampai nilai **a** adalah 1. Untuk nilai **a** sama dengan 1 ini, perintah **return(1)** akan mengembalikan proses ke bagian yang memanggilnya.

Hasil akhir dari nilai yang akan dikembalikan oleh fungsi faktorial dari penjelasan di atas untuk 5 faktorial dapat dituliskan sebagai berikut:



11.2 Perbandingan Rekursif dengan Iteratif

Kekurangan fungsi rekursif:

1. Memerlukan *memory* yang lebih banyak untuk menyimpan *activation record* dan variabel lokal. *Activation record* diperlukan waktu proses kembali kepada pemanggil.
2. Memerlukan waktu yang lebih banyak untuk menangani *activation record.*

Secara umum gunakan penyelesaian rekursif hanya jika:

1. Penyelesaian sulit dilaksanakan secara iteratif.
2. Efisiensi dengan cara rekursif sudah memadai.
3. Efisiensi bukan masalah dibandingkan dengan kejelasan logika program.

Jika sebuah masalah dapat diselesaikan dengan cara iteratif maka gunakan iteratif. Jika penyelesaian dengan menggunakan iteratif memerlukan algoritma yang relatif rumit maka dapat dicoba dengan pendekatan rekursif karena rekursif memerlukan sumber daya yang lebih banyak.

Contoh program untuk membandingkan penggunaan itertif dan rekursif dapat dilihat dalam program menampilkan deret fibonacci pada listing program 11.3 dan listing program 11.4 bawah ini.

Listing Program 11.3 Program Fibonacci dengan Rekursif

```
#include <stdio.h>

int fibo(int n);

int main ()
{
   int x;

   printf("Menampilkan deret Fibonacci\n");
   printf("Batas suku bilangan ke : ");
   scanf("%d", &x);
   printf("Deret ke %d = %d", x, fibo(x));
   printf("\n\nDeret fibonacci : \n");

   for(int i=1; i<=x; i++)
      printf("%d ",fibo(i)); /* Pemanggilan deret Fibonacci sebanyak x kali */

   return 0;
}

int fibo(int n)
{
  if (n==1)
      return 1;
  else if(n==2) return 1;
      else
            return fibo(n-2)+ fibo(n-1);
}
```

Hasil Running:

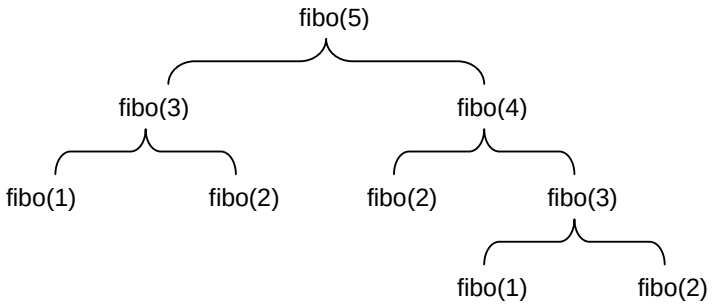
```
Menampilkan deret Fibonacci
Batas suku bilangan ke : 5
Deret ke 5 = 5

Deret fibonacci :
1 1 2 3 5
```

Penjelasan program:

Deret fibonacci mempunyai nilai suku-suku bilangan berikut:

0, 1, 1, 2, 3, 5, 8, 13, 21, ...

Ciri khusus deret ini adalah tiap-tiap suku adalah hasil penjumlahan dari nilai dua suku sebelumnya. Misalnya adalah nilai suku ke dua adalah penjumlahan nilai suku ke 0 (bernilai 0) dengan suku ke 1 (bernilai 1) jadi nilai suku ke 2 adalah sama dengan 1 (0 + 1). Nilai suku ke tiga adalah nilai suku ke dua ditambah nilai suku ke satu. Misalnya untuk mencari bilangan fibonacci ke- 5, maka urutan pengerjaannya adalah sebagai berikut:



Dari diagram terlihat bahwa untuk mendapatkan deret ke 5 dari deret fibonacci maka fungsi fibo(4) dihitung satu kali, fungsi fibo(3) dihitung dua kali, fungsi fibo(2) dihitung tiga kali dan fungsi fibo(1) dihitung dua kali. Hal ini menyebabkan proses lebih lama dan juga sumber daya yang dibutuhkan untuk menangani proses ini lebih banyak.

Listing Program 11.4 Program Fibonacci dengan Iteratif

```
#include <stdio.h>

int fibo(int n);

int main ()
{
   int x;

   printf("Menampilkan deret Fibonacci\n");
   printf("Batas suku bilangan ke : ");
   scanf("%d", &x);
   printf("Deret ke %d = %d", x, fibo(x));
   printf("\n\nDeret fibonacci : \n");

   for(int i=1; i<=x; i++)
      printf("%d ",fibo(i)); /* Pemanggilan deret Fibonacci sebanyak x kali */

   return 0;
}

int fibo(int n)
{
   int fb1, fb2, fn, i;
   if (n==1) return 1;
   if(n==2) return 1;
   fb1=1; fb2=1;
   for (i=3; i<=n; i++)
   {  fn = fb1 + fb2;
      fb2= fb1;
      fb1= fn;
   }
   return fn;
}
```

Hasil Running:

```
Menampilkan deret Fibonacci
Batas suku bilangan ke : 5
Deret ke 5 = 5

Deret fibonacci :
1 1 2 3 5
```

Penjelasan Program:

Dari listing program 11.4, terlihat bahwa setiap kali akan menghitung suku ke-n, maka nilai-nilai suku sebelumnya akan digunakan kembali, yaitu nilai dari n-1(variabel fb1) dan nilai dari n-2(variabel fb2). Pertama dihitung nilai dari deret ke-n, kemudian nilai dari deret n-1 diberikan kepada deret n-2, dan nilai dari deret n diberikan kepada deret n-1.

Latihan:

1. Apakah yang dimaksud dengan fungsi yang rekursif?
2. Apa persyaratan masalah untuk bisa diselesaikan dengan cara rekursif
3. Apa kekurangan fungsi rekursif?
4. Jelaskan mengenai *activation recors*.
5. Apakah lebih baik menulis program dengan menggunakan pendekatan iteratif ataukah rekursif?

Tugas Mandiri:

1. Buatlah program untuk menghitung kombinasi dan permutasi.
2. Buatlah program untuk menghitung jumlah nilai deret fibonacci dari masukan berupa batas deret.

==========

Referensi:

Deitel & Deitel, C How to Program 3rd Edition, Prentice Hall, New Jersey, 2001

Jogiyanto, Konsep Dasar Pemrograman Bahasa C, Andi Offset, Yogyakarta, 1993

Thompson Susabda Ngoen, Pengantar Algoritma dengan Bahasa C, Salemba Teknika, Jakarta, 2004

☺☺☺