Praktikum 6

Rekursi

POKOK BAHASAN:

- ✓ Konsep Rekursi
- ✓ Perbandingan Perulangan biasa dan Rekursi
- ✓ Implementasi Rekursi dalam Bahasa C

TUJUAN BELAJAR:

Setelah melakukan praktikum dalam bab ini, mahasiswa diharapkan mampu:

- ✓ Memahami rekursi sebagai konsep yang dapat digunakan untuk merumuskan solusi sederhana dalam sebuah permasalahan yang sulit untuk diselesaikan secara iteratif dengan menggunakan loop `for, while do`.
- ✓ Membantu mahasiswa bagaimana "berpikir secara rekursif"
- ✓ Mengidentifikasi permasalahan-permasalahan pemrograman yang bisa diselesaikan dengan menggunakan rekursi, sekaligus menyelesaikannya

DASAR TEORI:

Rekursi mempunyai arti suatu proses yang bisa 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. 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:

n! = (n)(n-1)(n-2) … (1)

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-1*dikalikan dengan *(n-2)!*, dan *(n-2)!* adalah *n-2* dikalikan dengan *(n-3)!* dan seterusnya sampai dengan *n = 1*, kita menghentikan penghitungan *n!* Cara rekursif untuk permasalahan ini, secara umum dapat kita detailkan sebagai berikut:

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

Dengan demikian ada tiga syarat yang harus dipenuhi agar suatu permasalahan dapat diselesaikan secara rekursif:

1. Permasalahan dapat dipecah menjadi lebih sederhana. Seperti dicontohkan untuk permasalahan faktorial di atas, 4! = 3! * 4. Dari permasalahan ini dapat dijelaskan bahwa 4! dapat dipecah dengan melibatkan faktorial yang lebih sederhana, yaitu 3!.
2. Karena dengan rekursi kita memanggil fungsi secara 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. Pada percobaan 1 merupakan contoh program rekursi yang tidak pernah berhenti karena tidak ada kondisi yang menghentikan prosesnya. Bandingkan dengan percobaan 2 yang mempunyai kondisi pengakhiran rekursi, yaitu jika nilai N sudah lebih kecil atau sama dengan nol. Sedangkan untuk rekursi pada fungsi faktorial, kondisi jika n=0 merupakan batas akhir proses rekursi dari fungsi tersebut.
3. Ada bagian program yang melakukan pemanggilan terhadap dirinya sendiri.

Berikut ini adalah suatu permasalahan yang dapat kita selesaikan dengan menggunakan proses rekursif. Yaitu deret fibonaci yang mempunyai nilai suku-suku bilangan sebagai berikut:

0, 1, 1, 2, 3, 5, 8, 13, 21, …

Ciri khusus deret ini adalah tiap-tiap nilai suku adalah hasil penjumlahan dari nilai dua suku sebelumnya. Nilai suku ke-3 adalah penjumlahan dari nilai suku ke-1 (bernilai 1)ditambah dengan nilai suku ke-2 (bernilai 1). Deret ini dimulai dari nilai suku ke nol yang bernilai nol. Suku ke-1 juga bernilai 1, sehingga suku ke-0 dan 1 bernilai sama dengan nilai urutannya. Cara rekursif untuk permasalahan ini, secara umum dapat kita detailkan sebagai berikut:

$$F(n) = \begin{cases} n & \text{jika n=0, n=1} \\ F(n-2) + F(n-1) & \text{jika n>1} \end{cases}$$

Rekursi ini jarang dipakai, diantaranya disebabkan:

- Membuat fungsi sulit dipahami. Algoritma ini hanya cocok untuk persoalan tertentu saja (misalnya pada binary tree atau pohon biner). Bandingkan program pada percobaan 3 dan percobaan 4. Kita akan lebih mudah membaca program pada percobaan 3.
- Memerlukan stack dengan ukuran yang lebih besar. Sebab setiap kali fungsi dipanggil, variabel lokal dan parameter formal akan ditempatkan ke stack dan adakalanya akan menyebabkan stack tak cukup lagi.
- Untuk kasus-kasus tertentu, algoritma ini mempunyai kelemahan. Kelemahan ini terleetak pada suatu proses yang sudah pernah dilakukan akan diproses kembali, sehingga membuat proses menjadi lebih lama dan tidak perlu. Contoh proses rekursi yang mengandung kelemahan misalnya adalah proses untuk mengitung nilai suku deret fibonaci seperti yang sudah dijelaskan di atas.

TUGAS PENDAHULUAN

Buatlah flowchart untuk permasalahan yang diberikan pada latihan soal no 5 di bawah ini sebagai Tugas Pendahuluan.

PERCOBAAN

1. Buatlah workspace untuk praktikum Struktur Data dengan menggunakan Visual C++.
2. Buatlah project untuk praktikum KEENAM.
3. Cobalah untuk masing-masing percobaan di bawah ini.
4. Selesaikan soal-soal yang ada dan mengimplementasikan flowchart yang anda buat pada Tugas Pendahuluan.

Percobaan 1 : Fungsi Rekursi tanpa Batas Akhir

```
#include <stdio.h>
void Tidak_Berhenti();
main()
{
     Tidak_Berhenti();
}
void Tidak_Berhenti()
{
     printf("Ctrl-Break untuk berhenti.\n");
     Tidak_Berhenti();
}
```

Percobaan 2 : Fungsi Rekursi dengan Batas Akhir

```
#include <stdio.h>

void Berhenti_N_Kali(int n);

main()
{
     int N=3;
     Berhenti_N_Kali(N);
}

void Berhenti_N_Kali(int n)
{
     static int i=0;
```

```
     if (n<=0) return;
     printf("%d kali\n",++i);
     Berhenti_N_Kali(n-1);
}
```

Percobaan 3 : Menghitung Nilai Faktorial dengan Menggunakan Perulangan

```
#include <stdio.h>

 int fact_it (int n)
 {
     int i,fak;

 /*******************************************************
 *  Menghitung sebuah faktorial dengan proses looping  *
 *******************************************************/
     temp = 1;
     for (i=1; i<=n; i++)
          fak = fak * i;
     return (fak);
 }

void main()
{
  int fac;
  printf("Masukkan berapa faktorial : ");
  scanf("%d",&fac);

  printf("Hasil faktorial dari adalah : %d ", fact_it(fac));
  printf('\n');
}
```

Percobaan 4 : Menghitung Faktorial dengan Menggunakan Rekursif

```
#include <stdio.h>

int fact_rec(int n)
{
/*********************************************************
 *    Menghitung sebuah faktorial secara rekursif        *
*********************************************************/
     if (n < 0)
          return 0;
     else if (n == 0)
          return 1;
     else if (n == 1)
          return 1;
```

```
      else
            return n * fact_rec(n-1);
}

void main()
{
      int fac;
      printf("Masukkan berapa faktorial : ");
      scanf("%d",&fac);


      printf("Hasil faktorial dari adalah : %d ", fact_rec(fac));
      printf('\n');
}
```

Percobaan 5 : Menampilkan Deret Fibonaci dengan Rekursif

```
  #include <stdio.h>

  int Fibonaci(int N);

  void main()
  {
     int i, N;

     printf("Masukkan batas akhir dari bilangan fibonaci : ");
     scanf("%d",&N);

     for (i=0; i<=N; i++)
           printf("%d, ",Fibonaci(i));
  }

  int Fibonaci(int N)
  {
     if (N<2) return (N);
     else
           return(Fibonaci(N-2)+Fibonaci(N-1));
  }
```

LATIHAN

1. Buatlah sebuah fungsi yang menulis angka dari n ke 0 dengan menggunakan proses rekursi.
2. Tuliskan sebuah fungsi untuk menulis anka dari 0 ke n dengan menggunakan proses rekursi.

3. Tuliskan sebuah fungsi rekursi yang melakukan pengecekan apakah sebuah elemen X merupakan anggota dari sebuah array a[n].
4. Tuliskan fungsi rekursi untuk membalik suatu kalimat. Sebagai contoh, kalimat 'Struktur Data' dibalik menjadi 'ataD rutkurtS'.

   Fungsi rekursi ini menerima parameter bertipe string dan mengembalikan string hasil pembalikan.
5. Tulis sebuah fungsi yang melakukan pengecekan apakah sebuah angka merupakan bilangan prima atau bukan (n bukan bilangan prima jika dapat dibagi dengan angka kurang dari n)