MODUL VI
Struktur Data

Judul	REKURSIVITY DAN PENGURUTAN LANJUT	
Penyusun	**Distribusi**	**Perkuliahan**
Nixon Erzed	Teknik Informatika Universitas Esa Unggul	Pertemuan – VI online

Tujuan :

Setelah mengikuti kuliah ini, mahasiswa dapat memahami cara bekerjanya fungsi rekursif, mengerti perbedaannya dengan fungsi iteratif dan dapat mengimplementasikannya dalam menyelesaikan masalah.

Materi :

1. Pengertian Umum
2. Model Fungsi Rekursif
3. Quick Sort

REKURSIVITAS

Dalam penerapan pemanggilan prosedur atau fungsi secara berulang, terdapat dua pendekatan yaitu :
- Teknik Iteratif
- Teknik Rekursif.

A. ITERATIF

Perulangan iteratif merupakan perulangan yang melakukan proses perulangan terhadap sekelompok instruksi di mana perulangan tersebut akan berhenti jika batasan syarat sudah tidak terpenuhi.

Algoritma iteratif
- Teknik Iteratif merupakan suatu teknik pembuatan algoritma dengan pemanggilan procedure beberapa kali atau hingga suatu kondisi tertentu terpenuhi.
- Tidak ada variabel lokal baru
- Program tidak sederhana
- Menggunakan perulangan for-next, while-do, atau repeat-until

Perulangan iteratif merupakan perulangan yang melakukan proses perulangan terhadap sekelompok intruksi. Perulangan dilakukan dalam batasan syarat tertentu. Ketika syarat tersebut tidak terpenuhi lagi maka perulangan akan terhenti.

Perhatikan contoh dibawah ini :

```
Procedure Konversi ( S : stack;  data : desimal;  var  b : biner)
Var    t, d  :  integer

Begin
        t := 0
        x :=  data
        While x  > 0  do

            PUSH(S, x MOD 2, t)
            x := x DIV 2

        End-while

        While  t > 0 do

            POP(S, d, t)
            Pack (d, b)

        End-while

        Write(" hasil konversi", data, "adalah ", b)

End;
```

Kelebihan perulangan iteratif:

- Mudah dipahami dan mudah melakukan debugging ketika ada perulangan yang salah.
- Dapat melakukan nested loop atau yang disebut dengan looping bersarang.
- Proses lebih singkat karena perulangan terjadi pada kondisi yang telah disesuaikan.
- Jarang terjadi overflow karena batasan dan syarat perulangan yang jelas.

Kelemahan perulangan iteratif:

- Tidak dapat menggunakan batasan berupa fungsi.
- Perulangan dengan batasan yang luas akan menyulitkan dalam pembuatan program perulangan itu sendiri.

Contoh lain program dengan perulangan Iteratif:

Program Contoh 1
Bentuk fungsi iteratif :

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

int jumlah(int n)
{
int hasil = 0;

for (int i=0; i<n; i=i+2)
        hasil = hasil + i;
return hasil;
}
void cetak(int n) {
for (int i=0; i<n; i=i+2)
cout << i << " ";
}
int main(int argc, char *argv[])
{
int n = 10;
cout << jumlah(n);
cetak(n);
system("PAUSE");
return EXIT_SUCCESS;
}
```

B.REKURSIF

Rekursif dapat diartikan bahwa suatu proses yang bisa memanggil dirinya sendiri. Sedikit menyimpang dari pengertian ada sedikit pendapat tentang Rekursif salah satunya adalah menurut definisi dalam Microsoft Bookshelf, Rekursif adalah kemampuan suatu rutin untuk memanggil dirinya sendiri.

Dalam Rekursif sebenarnya terkandung pengertian prosedur dan fungsi. Perbedaannya adalah bahwa rekursif bisa memanggil ke dirinya sendiri, tetapi prosedur dan fungsi harus dipanggil lewat pemanggil prosedur dan fungsi.

Rekursif merupakan teknik pemrograman yang penting dan beberapa bahasa pemrograman mendukung keberadaan proses rekursif ini. Dalam prosedur dan fungsi, pemanggilan ke dirinya sendiri bisa berarti proses berulang yang tidak bisa diketahui kapan akan berakhir.

Perhatikan ilustrasi fungsi factorial berikut ini :

$$1! = 1$$
$$2! = 1 \times 2 = 2$$
$$3! = 1 \times 2 \times 3 = 6$$
$$4! = 1 \times 2 \times 3 \times 4 = 24$$
$$5! = 1 \times 2 \times 3 \times 4 \times 5 = 120$$
$$6! = 1 \times 2 \times 3 \times 4 \times 5 \times 6 = 720$$
$$\ldots\ldots$$
$$n! = 1 \text{ x } 2 \text{ x } 3 \ldots \text{ x } (n-2) \text{ x } (n-1) \text{ x } (n)$$

jika disajikan secara terbalik n! dapat dituliskan sebagai berikut :

$$n! = (n) \text{ x } (n-1) \text{ x } (n-2) \text{ x } (n-3) \text{ x } \ldots 5 \text{ x } 4 \text{ x } 3 \text{ x } 2 \text{ x } 1$$

karena

$$2! = 2 \text{ x } 1$$

maka

$$3! = 3 \text{ x } 2 \text{ x } 1 \; = \; 3 \text{ x } 2!$$
$$4! = 4 \text{ x } 3 \text{ x } 2 \text{ x } 1 = 4 \text{ x } 3!$$

sehingga

$$n! = \; n \text{ x } (n-1) \text{ x } (n-2) \text{ x } (n-3) \text{ x } \ldots 5 \text{ x } 4 \text{ x } 3 \text{ x } 2 \text{ x } 1$$
$$= \; n \text{ x } (n-1)!$$

Pola factorial n!, jika disajikan sebagai sebuah fungsi yang dipanggil secara berulang dapat digambarkan sebagai berikut :

F = 5 * faktorial (4);

F = 5 * 4 * faktorial (3);

F = 5 * 4 * 3 * faktorial (2);

F = 5 * 4 * 3 * 2 * faktorial (1)

F = 5 * 4 * 3 * 2 * 1;

F = 120

Algoritma rekursif

- Teknik Rekursif merupakan salah satu cara pembuatan algoritma dengan pemanggilan procedure atau function yang sama
- Ada variabel lokal baru
- Program menjadi lebih sederhana
- Menggunakan perulangan if else

Kelebihan perulangan rekursif:

- Sangat mudah untuk melakukan perulangan dengan batasan yang luas dalam artian melakukan perulangan dalam skala yang besar.
- Dapat melakukan perulangan dengan batasan fungsi.

Perulangan rekursif merupakan salah satu metode didalam pemrograman dimana didalam sebuah fungsi terdapat intruksi yang memanggil fungsi itu sendri, atau lebih sering disebut memanggil dirinya sendiri.

Kekurangan perulangan rekursif:

- Tidak bisa melakukan nested loop atau looping bersarang.
- Biasanya membuat fungsi sulit untuk dipahami, hanya cocok untuk persoalan tertentu saja.
- Trace error sulit.
- Memerlukan stack yang lebih besar, sebab setiap kali fungsi dipanggil, variabel lokal dan parameter formal akan ditempatkan ke stack dan ada kalanya akan menyebabkan stack tak cukup lagi (Stack Overrun).
- Proses agak berbelit-belit karena terdapat pemangilan fungsi yang berulang-ulang dan pemanggilan data yang ditumpuk.

Berikut ini contoh-contoh Rekursivitas

1. Fungsi Factorial

```
Function Fact(n : integer) : integer
Begin
    If n = 1
       Then  Fact ⇐ 1
       Else
            Fact ⇐ n * Fact(n-1)
     End-if
End-Fact
```

Misalkan akan dihitung 3!, penelusuran prosesnya sebagai berikut

```
Fact (n = 3)
If n = 1  → salah
       Fact ⇐ 3 * Fact(n=2)
                  If n = 1  → salah
                         Fact ⇐ 2 * Fact(n=1)
                                    If n = 1  → benar
                                          Fact ⇐ 1
                         Return to Fact(n=2)
                         Fact ⇐ 2 * 1 = 2
       Return to Fact(n=3)
       Fact ⇐ 3 * 2 = 6
End
```

2. Contoh Tulisan di dalam sistem Peti

Terdapat sebuah sistem peti, terdapat peti didalam peti. Pada peti terdapat sebuah kata, kata pertama dimulai dari peti terdalam, misalnya :

Kita akan membahas materi pencarian
5 4 3 2 1

Kita → peti terdalam
Pengurutan → peti terluar

Jika "kata" tertulis didasar Peti
Maka urutan pembacaan agar terbaca sesuai urutan kata yang benar adalah sebagai berikut :

mulai dari i = 1 panggil BacaPesan(i = 1) hingga semua kata terbaca

```
Procedure BacaPesan ( i : integer)
Begin
      Bukapeti ( i )
      IF masih_ada_peti
        Then   BacaPesan( i+1)
      End-if
      BacaKata ( i )
      PindahkanPeti ( i )
End-Bacapesan
```

Dimulai i = 1

```
Bacapesan(1)
    Bukapeti(1)
    Masih_ada_peti = true
      Bacapesan(2)
        Bukapeti(2)
        Masih_ada_peti = true
         Bacapesan(3)
              Bukapeti(3)
              Masih_ada_peti = true
                 Bacapesan(4)
                    Bukapeti(4)
                    Masih_ada_peti = true
                       Bacapesan(5)
                          Bukapeti(5)
                          Masih_ada_peti = false
                          Bacakata(5)              → kita
                          PindahkanPeti(5)
                       End-bacapesan(5)            → Return(4)
                    Bacakata(4)                    → akan
                    Pindahkanpeti(4)
                 End_bacapesan(4)                  → Return(3)
              Bacakata(3)                          → membahas
              Pindahkanpeti(3)
          End-Bacapesan(3)                         → retrurn (2)
        Bacakata(2)                                → materi
        Pindahkanpeti (2)
      End-bacapesan(2)                             → return(1)
    Bacakata(1)                                    → pencarian
    Pindahkanpeti(1)
End-bacapesan(1)                                   → end prosedur
```

jika pada program contoh 1, diubah kedalam bentuk rekursif :

Dalam bentuk rekursif :

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

int jumlah(int n) {

if(n==0) return (0);
else return (n-2 + jumlah(n-2));
}

void cetak(int n) {
if(n!=0){
cetak(n-2);
cout << n-2 << ” “;
}
}
int main(int argc, char *argv[])
{
int n = 10;
cout << jumlah(n);
cetak(n);
system(“PAUSE”);
return EXIT_SUCCESS;
}
```

Catatan :
Fungsi recursive adalah suatu fungsi yang memanggil dirinya sendiri. Pada beberapa persoalan, fungsi rekursif sangat berguna karena mempermudah solusi. Namun demikian, fungsi rekursif juga memiliki kelemahan, yakni memungkinkan terjadinya overflow pada stack, yang berarti stack tidak lagi mampu menangani permintaan pemanggilan fungsi karena kehabisan memori stack adalah area memori yang dipakai untuk variable lokal untuk mengalokasikan memori ketika suatu fungsi dipanggil. Oleh karena itu, jika bisa diselesaikan dengan metode iteratif, gunakanlah metode iteratif.
Dan mesti diingat, tidak semua bahasa pemrograman mengadapatasi rekursivitas.

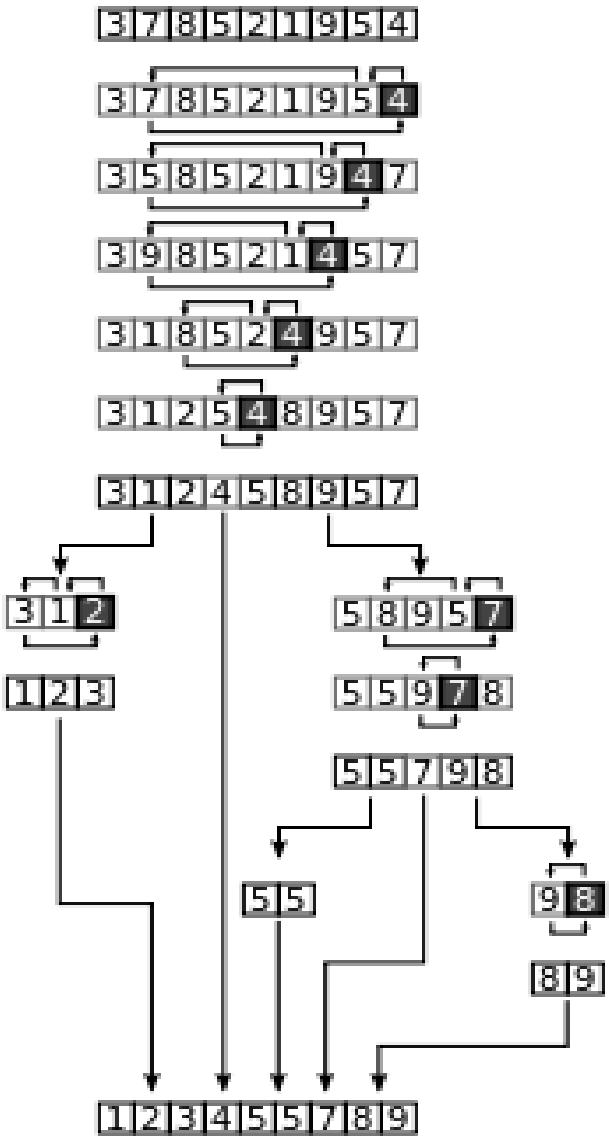
Prosedur pencarian dalam Pohon Pencarian Biner berikut ini termasuk algoritma rekursif.

```
Procedure  Search (var  Scurrent : Btree;   cari : typedata );
 begin
      if   cari  <  Scurrent^.Data
         then
           if  Scurrent^.kiri <>  nil
               then    Search ( Scurrent^.kiri , cari )
              else     write  ('data tidak ditemukan')
           { end-kiri }
         else
           if   cari  > Scurrent^.Data
             then
                 if  Scurrent^.kanan <>  nil
                    then    Search ( Scurrent^.kanan , cari )
                   else     write  ('data tidak ditemukan')
                { end-kanan }
               else
                  write  ('Data ditemukan')
          {end}
     {end}
 end;
```

Algoritma Quick Sort

Quick sort merupakan algoritma kedua tercepat setelah merge sort. Quick sort disebut juga *divide and conquer algorithm.* Quicksort dikembangkan oleh Tony Hoare. Quick sort memiliki average case *n.log(n)* untuk mengurutkan n item.

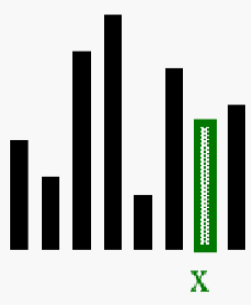
Algoritme ini juga dikenal sebagai Partition-Exchange Sort atau disebut sebagai Sorting Pergantian Pembagi. Pada kasus terburuknya, algoritme ini membuat perbandingan (n^2), malaupun kejadian seperti ini sangat langka. Quicksort sering lebih cepat dalam praktiknya daripada algoritma merge sort dan heapshort.
Pengurutan dan referensi lokalisasi memori quicksort bekerja lebih baik dengan menggunakan cache CPU, jadi keseluruhan sorting dapat dilakukan hanya dengan ruang tambahan (log n). Perhatikan ilustrasi pegurutan berikut :



Langkah-langkah quick sort :

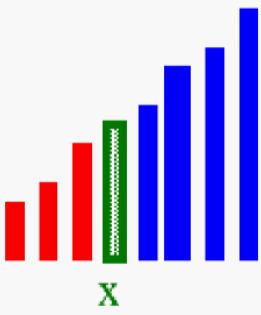
1. **SELECT**

   Pilih sebuah element , elemen ini kita sebut *pivot*



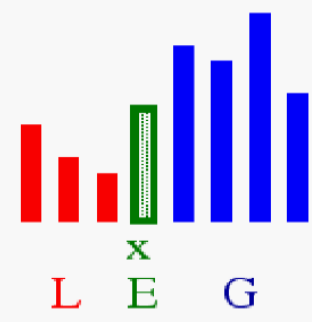
2. **DIVIDE**

   Kemudian pisahkan data menjadi 2 bagian, bagian yang **lebih kecil** dari pivot dan bagian yang **lebih besar** dari pivot



3. **RECUR and CONQUER**

   Kemudian 2 bagian tersebut diurutkan secara rekursif dengan memanggil fungsi itu sendiri, misal **quicksort()**



```
Deklarasi
    Var
          Data : array [1..20] of <type data>

Procedure  Quicksort (awal, akhir)
Var   kiri, kanan, pivot : integer
       Temp : <type data>
Begin
   IF   KIRI  < KANAN
    THEN
      Pivot := awal
      Kiri := awal + 1
      Kanan := akhir
      While  kiri ≤  kanan do
        Begin
            While data [kiri] < data [pivot]  and kiri <= kanan
              do
                 kiri := kiri + 1
            end-while
            While data [kanan] > data [pivot] and kanan >= kiri
              do
                  kanan := kanan -1
            end-while
            if  kiri < kanan
               then  Tukar (data[kiri] dengan data [kanan])
             end-if
       End-while
       Tukar ( data [pivot] dengan data [kanan] )
       Quicksort (awal, (kanan–1))
       Quicksort ((kanan+1), akhir)
   END-IF
End-quicksort
```

Perhatikan kode dibawah ini dan silakan dicoba:

```
#include <stdio.h>

void quicksort (int *a, int lo, int hi)
{
int m = lo, n = hi, h;
int x = a[(lo + hi) / 2];

do {
while (a[m] < x) {
++m;
}
while (a[n] > x) {
--n;
}
if (m <= n) {
h = a[m];
a[m] = a[n];
a[n] = h;
++m;
--n;
}
} while (m <= n);

if (lo < n) {
quicksort(a, lo, n);
}
if (m < hi) {
quicksort(a, m, hi);
}
}

#define MAX 5

int main (void)
{
int data[MAX] = { 3, 4, 1, 2, 8 };
int x;

printf("sebelum disortir\n");
for (x = 0; x < MAX; ++x) {
printf("%d ", data[x]);
printf("\n");
quicksort(data, 0, MAX - 1);
printf("setelah disortir\n");
for (x = 0; x < MAX; ++x) {
printf("%d ", data[x]);
}
printf("");

return(0);
}
```