BAB II

MATERI PENUNJANG

2.1. Algoritma

Suatu program komputer bekerja berdasarkan aturan-aturan atau instruksi-instruksi sesuai dengan masalah yang akan diselesaikannya. Gambaran proses yang berupa langkah-langkah yang akan dilaksanakan tersebut disebut **algoritma**.

Sebagai proses yang ditujukan untuk memperoleh hasil tertentu, maka suatu algoritma mempunyai karakteristik antara lain :

a. terbatas, artinya jelas bilamana proses akan berhenti. Hal ini penting agar program saat dieksekusi tidak mengalami *hanging* (proses menggantung atau berkepanjangan tanpa henti)
b. setiap langkahnya didefinisikan dengan jelas dan tidak menimbulkan makna ganda
c. setiap langkahnya diupayakan sesederhana mungkin sehingga dalam pemrosesan dapat mengefisienkan waktu eksekusi
d. menghasilkan output, sedangkan input mungkin diperlukan mungkin juga bisa tidak diperlukan.

Untuk mengilustrasikan algoritma biasanya dibuat dalam bentuk diagram alir (*flow chart*), yakni urutan langkah-langkah secara skematis yang menggambarkan

aliran eksekusi program dari awal sampai akhir.

2.2. Struktur Data

Untuk mengimplementasikan algoritma penyelesaian masalah ke dalam program komputer terlebih dahulu dilakukan penyusunan serta pendefinisian dari struktur tempat penyimpanan sehingga data dapat digunakan oleh program.

Suatu struktur data secara umum terdiri dari tiga bagian utama, yaitu :

a. Himpunan struktur dari tempat penyimpanan atau *storage*, yakni merupakan koleksi dari variabel-variabel dan hubungan antara satu variabel yang lain.

b. Himpunan dari fungsi-fungsi dasar.

   Dapat digunakan pada struktur tempat penyimpanan yang ada dan dapat digunakan pada setiap bagian program.

c. Himpunan dari algoritma.

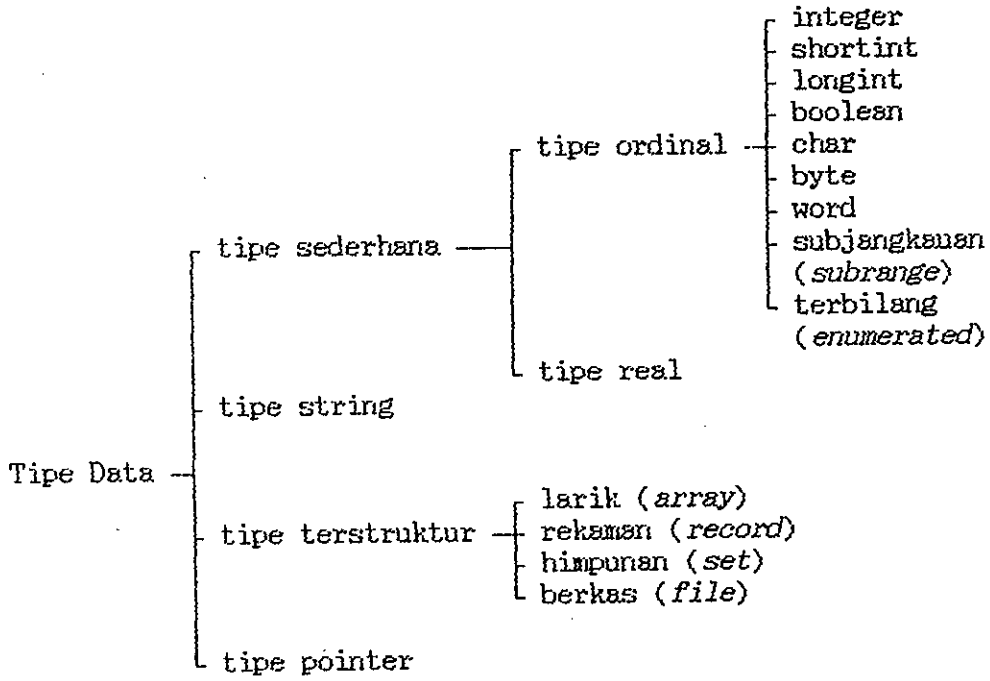
   Digunakan untuk pengubahan dari struktur tempat penyimpanan.

2.3. Tipe Data

Salah satu sub pembahasan dalam struktur data adalah mengenai tipe data. Definisi tipe data dari suatu variabel adalah kumpulan nilai-nilai yang dapat

dimuat oleh variabel yang bersangkutan. Dalam penyusunan suatu program komputer penggunaan tipe data sangat berkaitan dengan operasi-operasi yang dapat dilakukan terhadap variabel yang bersangkutan.

Dalam turbo pascal tipe data harus dideklarasikan di awal program, tepatnya pada bagian deklarasi **type**. Bagan berikut memperlihatkan pembagian tipe data yang dikenal dalam turbo pascal.



gambar 2.1. tipe data dalam turbo pascal

Berikut ini hanya akan dijelaskan tiga tipe yang akan banyak digunakan dalam pembahasan nanti, yaitu tipe rekaman, tipe larik dan tipe pointer.

2.3.1. Tipe rekaman

Tipe rekaman (**record**) adalah tipe data yang dapat digunakan untuk menyimpan lebih dari satu nilai untuk variabelnya (terstruktur), di mana setiap komponennya tidak harus mempunyai tipe sama. Komponen-komponen pada variabel yang bertipe **record** ini biasa disebut sebagai *field*.

Contoh dan deklarasi tipe record :

```
type  peserta = record
                  Nama   : string[20];
                  Nomor  : string[10];
                  Usia   : byte
                end;
```

Keterangan :

peserta : pengenal yang menunjukkan tipe data yang akan dideklarasikan

nomor, nama, usia : nama field yang akan digunakan

string[20],

string[10], byte : tipe data untuk masing-masing field

Apabila diinginkan untuk mengeksekusi unsur fieldnya maka dapat dinyatakan dengan menyebutkan nama variabel yang bertipe record diikuti tanda '.' dan nama field yang bersangkutan. Misal ingin membaca field nomor dari variabel peserta, dinyatakan dengan :

```
read(peserta.nomor)
```

2.3.2. Tipe larik

Larik (**array**) adalah tipe data terstruktur yang mempunyai komponen dalam jumlah tetap dan setiap komponennya mempunyai tipe data sama. Posisi masing-masing komponen dalam larik disebut nomor index.

Contoh dan deklarasi tipe larik

```
type  vektor = array[1..100] of real;
```

Keterangan :

vektor : nama pengenal yang dideklarasikan bertipe larik

1..100 : tipe data untuk nomor index

real : tipe data dari komponen

2.3.3. Tipe pointer

Tipe data pointer adalah data yang nilainya digunakan untuk menunjuk ke lokasi lain di mana berisi data yang akan diproses. Dalam pointer nilai yang ditunjuk oleh suatu pointer biasa disebut sebagai **simpul/node**. Sifat yang khas dari tipe data pointer adalah pengalokasiannya yang relatif fleksibel apabila diperlukan saja. Karena itulah variabel yang bertipe seperti ini biasa disebut sebagai variabel dinamis (*dinamic variable*).

Pengalokasian pada memori komputer untuk varibel dinamis dilakukan hanya pada saat diperlukan, yakni pada saat program dieksekusi, lokasi untuk varibel

belum ditentukan pada saat program dikompilasi, melainkan baru dicatat bahwa suatu variabel diperlukan sebagai variabel dinamis.

Bentuk umum deklarasi pointer adalah :

```
type  pengenal = ^simpul;
      simpul   = tipe;
```

dimana :

pengenal : nama pengenal yang menyatakan tipe data pointer

simpul : nama dari simpul

tipe : tipe dari simpul

Tanda '^' menunjukkan bahwa pengenal adalah tipe pointer. Tipe data simpul yang dinyatakan dalam tipe bisa berupa sembarang tipe data. Dalam banyak program aplikasi biasa terdapat sekumpulan data yang dikumpulkan dalam sebuah rekaman (record), maka dalam kasus seperti ini simpul dari pointer dapat berupa record. Misal :

```
type  data_nil = ^simpul;
      simpul   = record
                   Nama   : string[20];
                   NIM    : string[10];
                   Nilai1 : 1..100;
                   Nilai2 : 1..100
                 end;
```

Untuk pengalokasian simpul dalam memori digunakan statemen **new**, yang mempunyai bentuk umum :

```
var  P1 : data_nil;
new(P1);
```

dimana P1 adalah nama variabel yang bertipe pointer (data_nil).

Sedangkan untuk menunjukkan keaktifan simpul-simpulnya maka diperlukan sejumlah pointer yang sesuai. Pointer tersebut dimasukkan sebagai **field** pada **record** dari simpul yang bersangkutan. Dengan demikian deklarasi dapat dimisalkan seperti berikut :

```
type  data_info = ^simpul;
      simpul    = record
                    info    = tipe;
                    berikut = data_info
                  end;
```

di mana :

data_info : pengenal yang menyatakan tipe data pointer

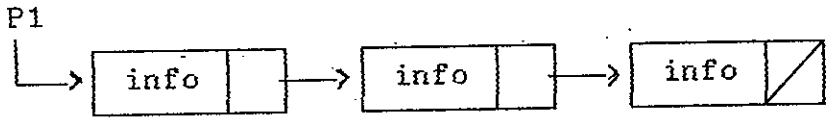
simpul : nama simpul

info : nama field dari data yang bisa bertipe sembarang

tipe : tipe data dari field info

berikut : nama field yang bertipe pointer

Dengan adanya field **berikut** yang juga bertipe pointer ini, maka antara simpul-simpul dapat digandengkan dengan cara pointer pada simpul yang satu menunjuk ke simpul lain. Apabila pointer tidak menunjuk ke simpul lain, maka nilainya adalah **nil**. Susunan simpul-simpul yang digandengkan melalui pointer seperti itu dinamakan senarai berantai.



gambar 2.2 *senarai berantai*

Dari pembahasan tipe-tipe data di atas, dapat dilihat perbandingan antara tipe data larik dan tipe data pointer sebagai berikut :

Tipe data larik	Tipe data pointer
Pengalokasian pada saat deklarasi type / var	Pengalokasian bila diperlukan/akan dieksekusi
Jumlah komponen yang bisa dimasup sesuai yang dideklarasikan atau kurang	Jumlah bisa bertambah sesuai keperluan
Alokasi yang tidak diisi/ digunakan akan kosong dan tidak dapat digunakan untuk yang lain	Apabila tidak diperlukan dapat dihapus, memori dapat digunakan untuk keperluan lain

2.4. Operasi Pada Pointer

Misal dideklarasikan suatu pointer dan variabel bertipe pointer sebagai berikut :

```
type  daf_alm = ^simpul;
      simpul  = record
                  Nama     : string[20];
                  Alamat   : string[30];
                  sambung  : daf_alm
                end;

var   T1, T2   : daf_alm;
```

maka beberapa operasi yang dapat dilakukan a.l. :

a. Menkopi pointer

Pengkopian pointer menghasilkan suatu simpul yang ditunjuk oleh lebih dari satu ~~simpul~~ pointer.

Contoh dua variabel dialokasikan bertipe daf_alm.

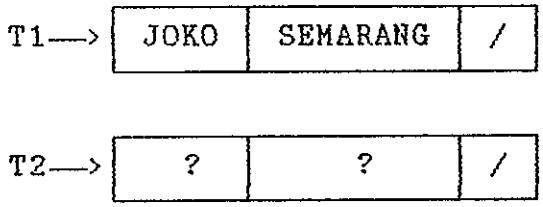
```
new(T1); new(T2);
```

maka ada dua simpul baru yang siap dimasup

Statemen

```
T1^.Nama   := 'JOKO';
T1^.Alamat := 'SEMARANG';
```

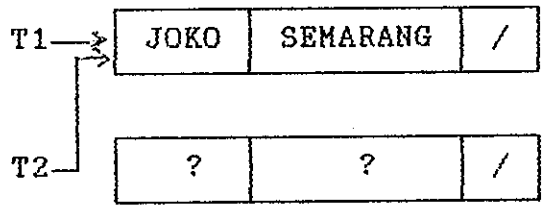
menyebabkan perubahan pada isi simpul T1



Statemen untuk menkopi pointer dinyatakan dengan :

```
T2 := T1;
```

menghasilkan



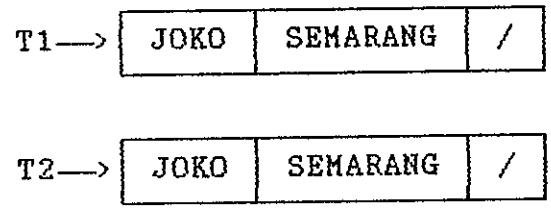
b. Menkopi isi simpul

Penkopian terhadap isi simpul menghasilkan beberapa simpul yang memiliki isi sama

Statemen penkopian isi simpul :

```
T1^ := T2^;
```

kondisi setelah penkopian adalah :



c. Operasi dengan menggunakan operator relasi

Karena untuk pointer hanya berurusan dengan alamat tertentu dalam memori, maka hanya bisa dioperasikan dengan dua operator yaitu = dan <>.

Contoh :

if T1 = T2 then

menyatakan apakah kedua pointer menunjuk ke lokasi yang sama

if T1^ = T2^ then

menyatakan apakah isi kedua simpul sama

2.5. Pengurutan Data

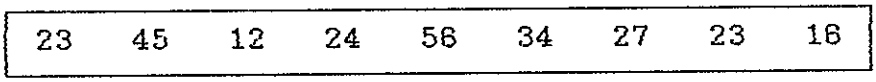
Pengurutan data (sorting) secara umum didefinisikan sebagai suatu proses untuk menyusun kembali himpunan objek menggunakan aturan tertentu, yaitu urut naik (*ascending*) atau urut turun (*descending*).

Data yang terurut biasa diperlukan untuk menentukan prioritas, urutan pengolahan data atau kepentingan pencarian data.

Ada bermacam metode untuk pengurutan data ini, namun di sini hanya akan diambil metode **QuickSort** sebagai metode yang akan digunakan dalam pemrograman penganggaran modal pada pembahasan selanjutnya. Metode ini diperkenalkan oleh **C.A.R. Hoare**. Secara garis besar metode ini dilakukan dengan membagi vektor yang berisi himpunan data yang akan diurutkan menjadi dua subvektor, keduanya dipisahkan oleh suatu elemen yang

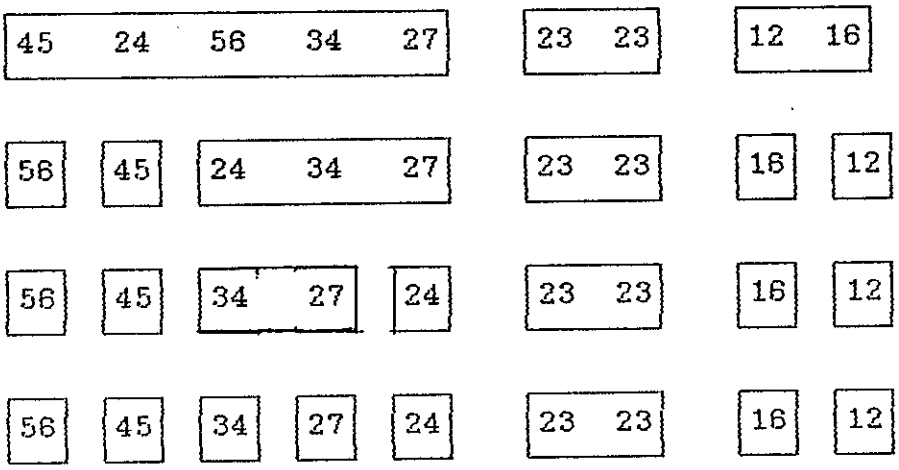
diambil sembarang dari himpunan data tersebut, biasanya diambil elemen pertama. Dua subvektor itu adalah himpunan elemen-elemen yang nilainya lebih kecil dari elemen pemisah dan himpunan elemen-elemen yang nilainya lebih besar dari elemen pemisah. Masing-masing subvektor kemudian dipecah lagi menjadi dua bagian dengan cara yang sama. Demikian seterusnya sampai seluruh subvektor hanya mempunyai satu elemen. Susunan subvektor-subvektor yang terakhir tersebut kalau digabungkan telah menjadi sebuah vektor yang terurutkan.

Ilustrasi dari metode ini, dimisalkan ada vektor



akan diurutkan secara *descending*, maka

diambil elemen pertama 23



gambar 2.3 *ilustrasi metode* QuickSort

Pengurutan senarai dengan metode QuickSort secara prinsipil sama, yakni membagi senarai menjadi tiga bagian, dengan asumsi pengurutan secara descending yakni :

a. bagian tengah

adalah bagian yang elemennya merupakan simpul yang pertama dipilih, dan simpul yang nilai objek kunci pembandingnya sama dengan simpul tersebut

b. bagian depan

berisi simpul-simpul yang objek kunci pembandingnya lebih besar dari objek kunci pembanding pada bagian tengah

b. bagian belakang

berisi simpul-simpul yang objek kunci pembandingnya lebih kecil dari objek kunci pembanding pada bagian tengah

Tiap-tiap bagian tersebut simpul pertama dan simpul terakhirnya ditandai dengan suatu pointer, misalkan di sini :

DpF untuk menandai simpul pertama pada bagian depan

DpL untuk menandai simpul terakhir pada bagian depan

TengF untuk menandai simpul pertama pada bagian tengah

TengL untuk menandai simpul terakhir pada bagian tengah

BlkF untuk menandai simpul pertama pada bagian belakang

BlkL untuk menandai simpul terakhir pada bagian

belakang

Fungsi pointer-pointer tersebut adalah sebagai pointer penghubung antar bagaian yang secara terurut.

Misalkan dideklarasikan suatu pointer sebagai berikut :

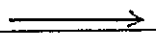
```
type   list   = ^simpul;
       simpul = record
                  info  : byte;
                  kanan : list
                end;
```

maka prosedur untuk pengurutan data menggunakan metode QuickSort adalah :

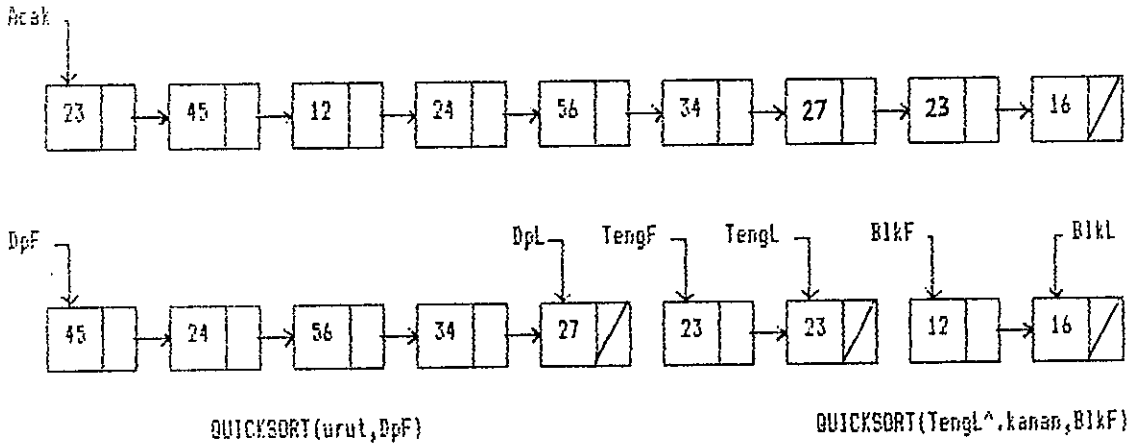
```
procedure SAMBUNG(B : list; var S,S1 : list);
begin
  if S = nil then S := B
  else S1^.kanan := B;

  S1 := B;
end;


procedure QUICKSORT(var urut : list; acak : list);
var  DpF,DpL,TengF,TengL,BlkF,BlkL : list;
begin
  if acak = nil then
     begin akhir := nil;
           urut  := acak
     end;
  else
     begin DpF := nil; TengF := nil; BlkF := nil;
           SAMBUNG(acak,TengF,TengL);
           acak := acak^.kanan;
```



```
            while acak <> nil then
                  begin if acak^.info > TengF^.info then
                              SAMBUNG(acak,DpF,DpL)
                        else if acak^.info = TengF^.info then
                              SAMBUNG(acak,TengF,TengL)
                        else SAMBUNG(acak,BlkF,BlkL);
                        acak := acak^.kanan
                  end;
            if DpF <> nil then
               begin DpL^.kanan := nil;
                     QUICKSORT(urut,DpF);
                     akhir^.kanan := TengF
               end
            else urut := TengF;
            if BlkF <> nil then BlkL^.kanan := nil;
            QUICKSORT(TengL^.kanan,BlkF);
            if Akhir = nil then Akhir := TengL
      end
end;
```



gambar 2.4. *pemecahan senarai menjadi tiga bagian dengan metode* **QuickSort**

Dari tiga bagian tersebut, bagian depan dan bagian

belakang akan akan diurutkan dengan metode QuickSort, artinya kedua bagian ini akan dipecah-pecah lagi menjadi tiga bagian juga. Pengurutan pada bagian depan dalam pemanggilan prosedur QUICKSORT menggunakan parameter nilai **DpF** sebagai penunjuk senarai yang akan diurutkan, dan parameter acuan **urut** untuk menunjuk senarai yang telah diurutkan. Sedangkan bagian belakang menggunakan parameter nilai **BlkF** sebagai penunjuk senarai yang akan diurutkan, dan parameter acuan **TengL^.kanan** menujukkan bahwa senarai yang telah diurutkan akan digandengkan setelah simpul terakhir dari bagian tengah.