BAB II

MATERI PENUNJANG

II.1 Algoritma

Definsi 2.1

Algoritma adalah suatu proses atau urutan aturan untuk menyelesaikan sebuah permasalahan dengan langkah terbatas dan dalam waktu yang terbatas.

Suatu algoritma mempunyai beberapa kriteria atau sifat-sifat secara umum, yaitu :

1. *Input*

   Suatu algoritma harus mempunyai nol atau banyak input, yaitu suatu besaran yang dimasukkan/diberikan sebelum dimulai.

2. *Output*

   Dari setiap nilai input algoritma menghasilkan nilai output / keluaran / hasil dari himpunan yang dispesifikasikan. Nilai output merupakan penyelesaian dari permasalahan. Suatu algoritma dapat memiliki satu atau lebih output.

3. *Defineteness*

   Langkah-langkah dalam suatu algoritma harus didefinisikan dengan tepat, tindakan penyelesaian harus ditetapkan dengan tepat dan jelas untuk setiap kasus.

4. *Finiteness*

   Sebuah algoritma menghasilkan output yang dinginkan setelah akhir dari proses untuk sembarang input dilakukan.

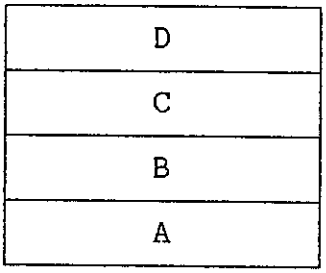
5. *Effectiveness*

   Setiap langkah/instruksi yang dilakukan sedapat mungkin dibuat sederhana, sehingga bisa dilakukan dengan mudah dan cepat.

II.2 Tumpukan (stack)

Salah satu bentuk struktur data dalam Ilmu Komputer adalah tumpukan (*stack*). Secara sederhana tumpukan bisa diartikan sebagai suatu kumpulan data yang seolah-olah ada data yang diletakkan di atas data yang lain. Bentuk struktur data seperti ini memungkinkan adanya penambahan data dan pengambilan (penghapusan) data melalui ujung yang sama yang disebut ujung atas tumpukan (*top of stack*).

Untuk menjelaskan pengertian di atas, diberikan contoh sebagai berikut. Ada dua buah kotak yang ditumpuk, sehingga kotak yang satu di atas kotak yang lain. Jika kemudian kedua kotak tersebut ditambah dengan kotak ketiga, keempat dan seterusnya, maka akan diperoleh tumpukan kotak yang terdiri dari N kotak. Contoh tersebut bisa diilustrasikan sebagai berikut.



Gambar 2.1

Dari gambar tersebut dapat dikatakan bahwa kotak B berada di atas kotak A dan berada di bawah kotak C. Gambar tersebut juga memperlihatkan bahwa dalam tumpukan hanya bisa menambah atau mengambil lewat satu ujung, yaitu ujung bagian atas. Dapat dilihat pula bahwa tumpukan merupakan kumpulan data yang sifatnya dinamis, artinya kita bisa menambah dan mengambil data darinya. Jika ada elemen baru yang akan ditambahkan maka akan diletakkan di atas kotak D, namun jika ada kotak yang akan diambil maka kotak D adalah kotak pertama yang terambil

Aturan semacam itu memperlihatkan bahwa tumpukan merupakan senarai (*list*) yang mempunyai sifat *'masuk terakhir keluar pertama'* (*last in first out- LIFO*). Operasi yang dapat dilakukan pada tumpukan adalah penambahan (PUSH) dan penghapusan (pengambilan - POP).

II.3. Perubah Dinamis

Di dalam Pascal terdapat dua jenis perubah (variabel) masing-masing dikenal sebagai perubah statis (*static variable*) dan perubah dinamis (*dynamic variable*).

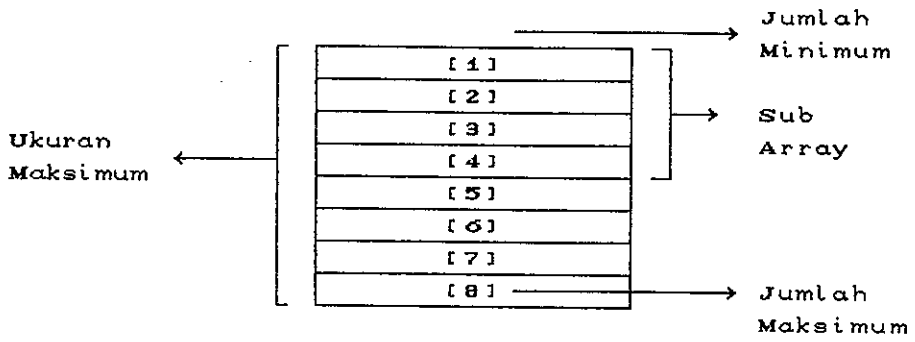
Nama perubah yang digunakan untuk mewakili suatu data sebenarnya menunjuk ke lokasi tertentu dalam pengingat utama (memori) komputer. Pada saat program dikompilasi, kompiler akan melihat pada bagian deklarasi perubah (var) untuk mengetahui nama-nama perubah apa saja yang akan dipergunakan, sekaligus mengalokasikan atau menyediakan tempat dalam pengingat untuk menyimpan nilai data tersebut.

Jadi sebelum dieksekusi, lokasi-lokasi data dalam pengingat sudah ditentukan dan tidak bisa diubah. Perubah yang demikian disebut dengan perubah statis (*static variable*). Artinya perubah yang dimaksud dalam program tersebut akan tetap menempati lokasi yang telah ditentukan. Dengan demikian banyaknya data yang bisa diolah adalah terbatas.

Struktur yang digunakan untuk menerapkannya adalah array (larik). Ukuran dari struktur statis dibuat pada waktu kompilasi, dan struktur ini akan selalu ada sepanjang eksekusi program. Jika suatu ketika pemakai (user) tidak memiliki bayangan jumlah komponen yang akan dimiliki. Kadang-kadang pendekatan dalam kondisi seperti ini adalah dengan mendeklarasikan suatu array berukuran besar untuk menampung jumlah maksimum data yang diperkirakan.

Ketika dimiliki jumlah data yang lebih kecil dari maksimumnya, maka panjang dari sub array tempat data terekam itu saja yang diakses. Sub array ini dapat berubah

panjangnya selama eksekusi, tetapi array itu sendiri tidak dapat berubah ukurannya. Perhatikan gambar berikut ini.



Gambar 2.2 Struktur Statis

Pada gambar tersebut jumlah maksimum data yang dapat ditampung oleh array tersebut hanya 8 komponen. Jika dipaksakan menambahkan data pada array tersebut, maka program akan terhenti karena deklarasi lariknya kurang. Jika pemakai dihadapkan pada permasalahan bahwa banyaknya data yang diolah tidak bisa dipastikan sebelumnya, teknik tersebut sangat tidak menguntungkan.

Terdapat teknik lain untuk mengantisipasi masalah tersebut yaitu menyajikan suatu senarai dengan komponen dari senarai adalah perubah dinamis (suatu perubah yang dialokasikan hanya pada saat diperlukan, yaitu setelah program dieksekusi). Pada saat kompilasi, lokasi perubah tersebut belum ditentukan. Kompiler hanya akan mencatat bahwa suatu perubah akan diperlakukan sebagai perubah dinamis. Artinya bahwa dapat didefinisikan suatu tipe data pada waktu kompilasi, tetapi tidak membuat suatu perubah tipe tersebut hingga eksekusi program. Variabel dinamis

ini dapat dibuat dari suatu tipe yang sederhana atau terstruktur yang dapat dibuat dan dihapus setiap waktu selama eksekusi program dengan menggunakan prosedur *New* dan *Dispose*.

Pada perubah statis, isi pengingat pada lokasi tertentu (nilai perubah) adalah nilai data yang sesungguhnya. Suatu perubah dinamis tidak ditunjukkan oleh nama perubahnya tetapi melalui suatu pointer yaitu suatu tipe data sederhana yang berisi alamat (lokasi) penyimpanan data sesungguhnya. Setiap perubah dinamis mempunyai suatu kumpulan pointer yang digunakan untuk mengakses.

Dengan demikian data yang sesungguhnya tidak bisa diakses secara langsung. Perubah dinamis diilustrasikan sebagai berikut.

Gambar 2.3

Pada gambar tersebut A adalah perubah dinamis. Nilai perubahnya misalnya 5. Nilai ini bukan nilai sesungguhnya, tetapi menunjukkan alamat nilai data yang sesungguhnya. Jadi data yang sesungguhnya tersimpan pada lokasi 5, yaitu 1000. Nilai-nilai perubah dinamis akan digunakan untuk menunjuk ke lokasi lain yang berisi data sesungguhnya yang akan diproses. Karena alasan tersebut maka perubah dinamis

lebih dikenal dengan sebutan *pointer*, yang artinya menunjuk ke sesuatu. Dalam pointer nilai data yang ditunjuk biasanya disebut *simpul* atau *node*.

II.3.1 Deklarasi Pointer dan Alokasi Tempat

Bentuk umum deklarasi pointer adalah :

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

Keterangan :

pengenal = nama pengenal yang akan menyatakan data bertipe pointer.

simpul = nama simpul.

tipe = tipe data dari simpul.

Tanda ^ di depan nama simpul harus ditulis, yang menyatakan dan menunjukkan bahwa pengenal adalah suatu tipe data pointer. Tipe data simpul yang dinyatakan dalam tipe bisa berupa string, char, integer, real dan lain sebagainya. Namun dalam kebanyakan program, biasanya terdapat sekumpulan data yang dikumpulkan dalam sebuah rekaman (*record*), sehingga akan banyak dijumpai tipe data pointer yang elemennya (data yang ditunjuk) adalah sebuah rekaman. Jadi tipe data simpul bisa dinyatakan dengan :

```
simpul  = record
            Info  : tipe ;
            Next  : pengenal ;
          end;
```

Keterangan

Info = Nama medan dari data yang bisa bertipe sembarang.

tipe = tipe data dari masing-masing medan.

Next = nama medan yang bertipe pointer.

Pada saat program dikompilasi, perubah yang didefinisikan akan menempati lokasi tertentu dalam pengingat, namun belum menunjuk ke suatu simpul. Pointer yang belum menunjuk ke suatu simpul nilainya dinyatakan sebagai NIL.

Untuk mengalokasikan simpul dalam pengingat, statemen yang digunakan adalah NEW, yang mempunyai bentuk umum :

```
new (perubah) ;
```

dengan perubah adalah perubah yang bertipe pointer.

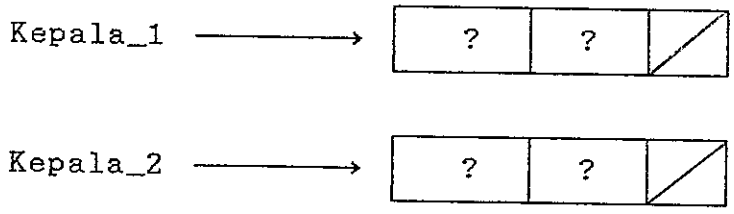
Contoh :

```
Pointer   = ^Data ;
Data      = record
              Kandidat : integer ;
              Jumlah   : integer ;
              Next     : pointer;
            end;
var   Kepala_1, Kepala_2   : Pointer ;
```

maka setelah diberikan statemen :

```
new (Kepala_1) ;
new (Kepala_2) ;
```

dieksekusi, dihasilkan dua simpul yang dilustrasikan sebagai berikut :



Gambar 2.4

Medan Kandidat dan jumlah belum mempunyai nilai, maka nilainya ditunjukkan dengan '?'. Sedangkan medan Next (bertipe pointer) karena tidak menunjuk ke simpul lain nilainya adalah NIL disimbolkan seperti gambar di atas.

II.3.2 Operasi pada Pointer

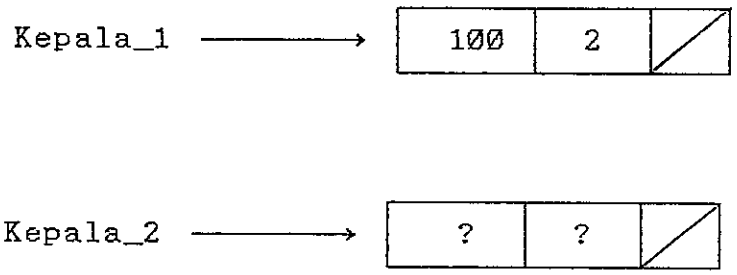
Secara umum ada dua macam operasi dasar yang dilakukan pada data yang bertipe pointer, yaitu :

1. Mengkopi pointer, sehingga simpul akan ditunjuk oleh lebih dari satu pointer.
2. Mengkopi isi simpul, sehingga dua atau lebih simpul yang ditunjuk oleh pointer yang berbeda mempunyai isi yang sama.

Syarat yang harus dipenuhi pada kedua operasi ini adalah poiter-pointer yang dioperasikan harus mempunyai deklarasi sama. Dengan menggunakan deklarasi pada contoh di atas, akan diperlihatkan operasi yang dimaksud. Setelah terdapat dua simpul yang masih kosong, jika diberikan statemen :

```
Kepala_1^.Kandidat := 100 ;
Kepala_1^.Jumlah := 2 ;
```

maka keadaan simpul berubah menjadi



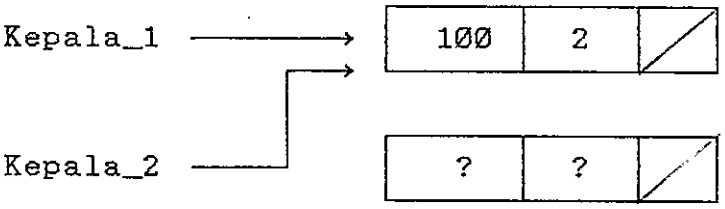
Gambar 2.5

(i) Operasi mengkopi pointer

Diberikan statemen

```
Kepala_2 := Kepala_1 ;
```

maka gambar 2.5 berubah menjadi



Gambar 2.6

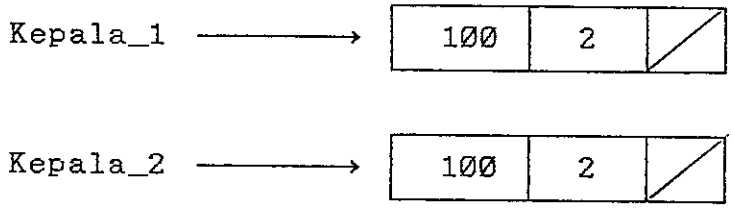
Gambar tersebut menunjukkan bahwa pointer Kepala_2 menunjuk ke simpul yang ditunjuk oleh pointer Kepala_1, simpul yang semula ditunjuk oleh Kepala_2 menjadi terlepas. Dalam keadaan seperti ini karena tidak ditunjuk oleh pointer lain, maka simpul tersebut tidak bisa diakses lagi karena lokasi simpul tersebut terhapus dari pengingat utama.

(ii) Mengkopi simpul

Statemen yang diberikan :

```
Kepala_2^ := Kepala_1^ ;
```

maka gambar 2.5 akan berubah menjadi



Gambar 2.7

Statemen pemberian hanya bisa dilaksanakan untuk perubah-perubah yang bertipe sama. Dengan demikian statemen pemberian berikut ini adalah tidak benar.

```
Kepala_1 := Kepala_2^ ;
Kepala_2^ := Kepala_1 ;
```

Operasi lain yang dapat dilakukan terhadap pointer adalah operasi relasi, namun karena pointer hanya mengidentifikasikan alamat tertentu dalam pengingat, maka operasi yang bisa dilakukan adalah = dan <>. Contoh

```
if Kepala_1 = Kepala_2 then
```

Statemen ini mengecek apakah kedua pointer menunjuk ke lokasi yang sama.

```
if Kepala_1 <> nil do
```

Statemen ini mengecek apakah pointer Kepala_1 tidak menunjuk ke suatu lokasi.

II.3.3 Menghapus Pointer

Pointer yang sudah dialokasikan (dibuat) bisa didelokasikan (dihapus) kembali pada saat program dieksekusi. Setelah suatu pointer didelokasi, maka lokasi yang semula ditempati oleh simpul yang ditunjuk oleh pointer tersebut akan bebas sehingga bisa digunakan oleh perubah lain.

Statemen yang digunakan untuk menghapus pointer adalah **dispose** yang mempunyai bentuk

dispose (perubah)

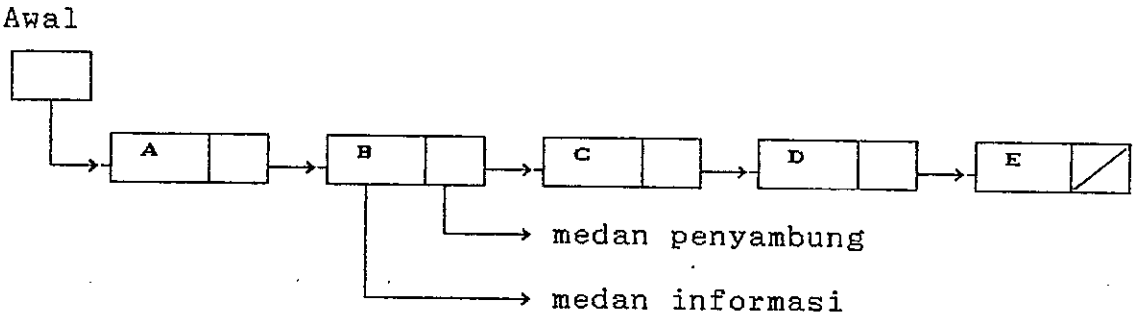
dengan perubah adalah sembarang perubah yang bertipe pointer.

II.4. Senarai Berantai

Salah satu bentuk struktur data dinamis yang paling sederhana adalah senarai berantai (*linked list*), atau senarai satu arah (*one-way list*). Yaitu jika suatu simpul memuat pointer yang menghubungkannya ke simpul berikutnya.

Dengan demikian senarai berantai adalah kumpulan komponen yang disusun secara berurutan dengan bantuan pointer. Masing-masing komponen disebut dengan simpul (*node*). Dengan demikian setiap simpul dalam suatu senarai berantai terbagi menjadi dua bagian. Bagian pertama, disebut medan informasi, berisi informasi yang akan disimpan dan diolah. Bagian kedua disebut medan penyambung (*linked field*), berisi alamat simpul berikutnya.

Pengaksesan suatu senarai berantai seperti permainan anak-anak yaitu mencari atau memburu harta karun. Masing-masing anak diberikan suatu petunjuk ke tempat persembunyian petunjuk berikutnya dan rentetan atau rangkaian petunjuk tersebut akhirnya membawanya ke harta karun. Gambar berikut menunjukkan diagram skematis dari senarai berantai 5 simpul.



Gambar 2.8

Setiap simpul dibagi dalam dua bagian, bagian kiri adalah medan informasi, bagian kanan adalah medan penyambung, sehingga dalam diagram digambarkan sebagai anak panah. Pada dasarnya medan penyambung sebenarnya adalah suatu pointer yang menunjuk ke simpul berikutnya, sehingga nilai dari medan ini adalah alamat suatu lokasi tertentu dalam pengingat. Keberadaan pointer Awal pada gambar 2.8 bukan merupakan bagian dari senarai, tetapi menunjuk ke simpul pertama dari senarai tersebut. Medan penyambung atau pointer yang tidak menunjuk ke simpul lain disebut pointer kosong, yang nilainya dinyatakan dengan *nil* (kata baku Pascal yang berarti bahwa pointer tidak

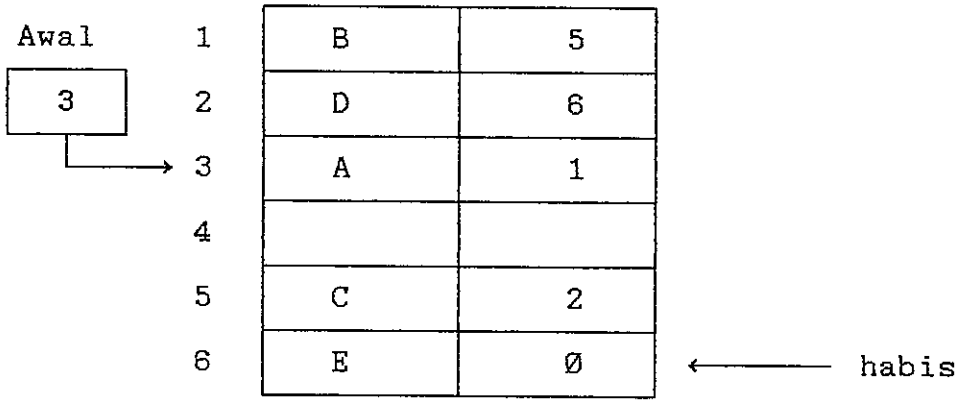
menunjuk ke suatu simpul yang nilainya sama dengan nol atau bilangan negatif).

Dalam mengakses senarai berantai digunakan suatu variabel pointer yang memuat alamat simpul pertama dalam senarai. Variabel pointer tersebut disebut pointer eksternal. Setiap simpul yang lain diakses oleh field pointer pada simpul sebelumnya. Dengan memperhatikan gambar 2.8 di atas terlihat bahwa hanya dengan satu buah pointer Awal saja maka semua informasi dalam senarai berantai dapat dibaca.

II.4.1 Penyajian Senarai Berantai

Meskipun secara skematis senarai berantai dapat digambarkan secara berurutan, tetapi dalam pengingat utama simpul-simpul yang berurutan tidak harus disimpan secara berurutan pula. Misal senarai dalam gambar 2.8 dinamakan *kandidat*. Senarai berantai kandidat dalam pengingat akan disajikan dengan cara sebagai beriut. Pertama kali *kandidat* memerlukan 2 larik linier (vektor), dinamakan vektor *Info* dan *Sambungan*, sedemikian sehingga vektor *Info* berisi komponen dalam medan informasi dan vektor *Sambungan* berisi medan pointer (alamat) ke simpul berikutnya. Disamping itu, *kandidat* juga memerlukan suatu perubah misal *Awal* yang berisi alamt simpul pertama dari senarai berantai, dan pointer *sentitel* yang bernama *kosong* = Ø, yang menunjukkan akhir sambungan senarai berantai.

Gambar 2.9 berikut ini merupakan contoh penyajian senarai berantai dalam pengingat utama.



Gambar 2.9

Dari gambar tersebut dapat dijelasakan sebagai berikut :

Awal = 3, maka Info [3] = A

Sambungan [3] = 1, maka Info [1] = B

Sambungan [1] = 5, maka Info [5] = C

Sambungan [5] = 2, maka Info [2] = D

Sambungan [2] = 6, maka Info [6] = E

Sambungan [6] = Ø, akhir senarai berantai

Dari contoh tersebut diperoleh untai 'ABCDE'

Untuk membuat suatu senarai berantai, dimulai dengan membuat simpul yang pertama dan menyimpan pointer ke dalam pointer eksternal. Kemudian membuat simpul kedua dan menyimpan pointernya pada field sambungan (*link*) dari simpul yang pertama. Proses tersebut diteruskan dengan membuat simpul baru dan menyimpan (menempatkan) pointernya

ke dalam field sambungan simpul sebelumnya. hingga dicapai akhir dari suatu senarai.

II.4.2 Senarai Berantai Sebagai Tumpukan

Operasi penambahan simpul baru di awal suatu senarai berantai sehingga simpul baru adalah simpul pertama serupa dengan operasi PUSH (memasukkan elemen) ke dalam suatu tumpukan. Dalam kedua kasus ini elemen yang ditambahakan adalah satu-satunya elemen dalam kumpulan elemen yang bisa segera diakses. Tumpukan hanya bisa diakses lewat elemen pertama yang menempati posisi teratas dalam tumpukan, dan senarai berantai hanya bisa diakses lewat pointer yang menunjuk ke elemen pertama. Demikian halnya dengan operasi POP (menghapus/mengambil elemen) dari suatu tumpukan. Dalam hal ini hanya elemen berikutnya menjadi elemen baru yang bisa segera diakses setelah elemen sebelumnya dihapus.