BAB IV

POINTER DAN STRUKTUR DATA DINAMIK

TUJUAN PRAKTIKUM

1. Mengerti dalam penggunaan pointer.
2. Dapat membuat larik dinamik dan daftar berkait

TEORI PENUNJANG

Pointer

Tipe pointer adalah data yang berisi suatu alamat yang menunjuk ke lokasi tertentu. Bila pointer berisi alamat dirinya sendiri maka pointer tidak menunjuk ke manapun disebut nil.

Bentuk umum dari deklarasi tipe pointer:

- Untuk pointer **bertipe**:

  <nama_var> : ^<tipe_data>;

- Untuk pointer **tidak bertipe**:

  <nama_var> : pointer;

Suatu pointer dapat menunjuk ke data **bertipe** *elementer*, *terstruktur*, *pointer yang lain*, atau *tidak bertipe*. Jika suatu pointer *tidak menunjuk ke mana-mana*, pointer itu dinamakan *dangling*, sedangkan bagian memori yang tidak dapat diakses karena tidak ada pointer yang menunjuk dinamakan *garbage* (sampah).

Dalam Pascal, pointer dapat diisi dengan nilai yang berasal dari:

1. NIL
2. Fungsi Ptr
3. Operator @
4. Prosedur New dan GetMem
5. Pointer yang lain

Reserved word NIL

NIL merupakan reserved word dalam Pascal, di mana pointer yang bernilai NIL dianggap tidak menunjuk alamat memori manapun.NIL biasa digambarkan dengan lambang ground.

Fungsi Ptr

Sintaks:

Function Ptr(Seg, Ofs : word) : pointer;

dengan *Seg* : segmen memori.

Ofs : offset memori.

Fungsi Ptr mengembalikan pointer dari segmen dan offset yang dimasukkan.

Operator @

Sintaks:

<nama_var>:=@<variabel_yang_alamatnya_diambil>;

Operator ini digunakan untuk mengambil alamat variabel yang akan ditunjuk.

Prosedur New dan GetMem

Sintaks:

New(var *P* : pointer);

GetMem(var *P* : pointer, *size* : word);

Dengan *P* : pointer yang akan diisi.

Size : ukuran yang dipesan.

Prosedur **New** digunakan untuk memesan memori untuk *pointer bertipe*, sedangkan prosedur **GetMem** untuk *pointer tidak bertipe*. Kedua prosedur ini akan membentuk suatu variabel dinamik yang diletakkan dalam *Heap*. **Heap** adalah memori-memori di komputer yang belum dialokasikan, yaitu memori yang tidak digunakan oleh DOS, oleh program-program resident, oleh program Turbo Pascal, internal stack yang digunakan oleh Turbo Pascal dan variabel-variabel di data segmen.

Pointer yang belum digunakan sebaiknya diisi dengan NIL, dan untuk pointer yang telah menunjuk sebuah alamat yang sudah dipesan memorinya, isinya dapat dimanipulasi melalui pointer.

Contoh:

```
Type
PenunjukKaryawan = ^CatatanKaryawan
Catatankaryawan=Record
        Kode : string[5];
        Nama : string[25];
        Gaji : Real;
        End;
Var
Datakaryawan : Penunjukkaryawan;
```

Pada deklarasi ini, tipe data PenunjukKaryawan adalah tipe data pointer yang menunjuk ke suatu record CatatanKaryawan dan deklarasi dari record ini dapat diletakkan dibawahnya.

Variabel dinamik dapat dihapus dari *heap* menggunakan prosedur standar *Dispose* dan prosedur standar *Mark* dan *Release*. Prosedur standar Mark hanya akan memberi tanda saja dan yang digunakan untuk menghapusnya adalah prosedur standar *Release*.

Larik Dinamik

Larik dinamik bentuknya seperti larik statik biasa, hanya dialokasikan ke *heap* dengan prosedur standar *New*.

Daftar Berkait

Suatu daftar Berkait (*linked list*) adalah suatu simpul(*node*) yang menunjuk ke simpul berikutnya di dalam suatu urutan. Suatu simpul dapat berupa struktur data record. Suatu node minimal harus mempunyai dua buah komponen., yaitu:

1. Satu atau lebih field yang berisi data didaftar berkait.
2. Satu atau lebih field berupa pointer yang menunjuk ke node lainnya. Field yang berupa pointer ini disebut dengan kait(*link*).

Untuk mendefinisikan linked list biasa digunakan record, dengan sintaks:

```
Type <nama_pointer> = ^<nama_rec>;
    <nama_rec> = record
                    data1 : <tipe_data1>;
                    data2 : <tipe_data2>;
                    . . . .
                    next : <nama_pointer>;
                end;
```

Masing-masing kotak pada linked list disebut node (simpul), node paling depan disebut head, node belakang disebut tail. Untuk mengisi linked list kosong maupun mengakhiri linked list biasanya digunakan nilai NIL.

Operasi-operasi yang terdapat pada linked list antara lain:

1. Inisialisasi; yaitu mengisi variabel list dengan NIL.
2. Menambah node.
3. Menyisipkan node.
4. Menghapus node yang berisi data.
5. Membaca data dari node.
6. Menghapus seluruh isi list.

Membuat daftar berkait.

Suatu daftar berkait harus mempunyai ujung awal dari simpul (*node*) dan ujung akhir dari node. Ujung awal diperlukan supaya dapat mengetahui awal dari node dan unjung akhir diperlukan supaya mengetahui sampai dimana daftar berkait berakhir. Ada beberapa cara menunjukkan akhir dari node, yaitu :

1. Memberikan suatu nilai data yang unik yang tidak pernah terjadi pada data sesungguhnya. Data ini disebut dengan data *sentinel* dan simpulnya disebut dengan simpul boneka (*dummy node*). Bila node berisi dengan data ini, maka dianggap sebagai akhir dari daftar berkait.
2. Menetapkan pointer di node yang terakhir dengan nilai *Nil*. Kata cadangan *Nil* ini menunjukkan suatu nilai ponter yang tidak menunjuk ke mana pun. Bila pointer di node berisi nilai Nil maka berarti merupakan akhir dari daftar berkait.

Ada berbagai jenis *linked list*, contoh diatas merupakan seranai beranai tunggal (*Single Link List*). Adapun jenis-jenis link *list* antara lain adalah :

1. *Single Link List* / Link *list* satu arah (*One Way List*)

Disebut demikian karena pada setiap simpul hanya memiliki satu buah *field* yang berhubungan dengan simpul berikutnya. Dalam pembuatan *Single Link List* dapat menggunakan 2 metode, yaitu:

- LIFO (*Last In First Out*), aplikasinya : *Stack* (Tumpukan)
  LIFO adalah suatu metode pembuatan *Link List* dimana data yang masuk paling akhir adalah data yang keluar paling awal.
- FIFO (*First In First Out*), aplikasinya : *Queue* (Antrian)
  LIFO adalah suatu metode pembuatan *Link List* dimana data yang masuk paling awal adalah data yang keluar paling awal juga.

Link *list* ini memiliki beberapa variasi lain diantaranya :

a. *Header Single Link List* : Jenis *single link list* yang memiliki simpul tambahan pada awal simpul yang berguna untuk informasi tambahan. Contoh dibawah ini merupakan *header single link list* yang pada simpul *header*-nya berisi informasi mengenai banyaknya simpul di dalam *list*

b. *Circular Single Link List* : Jenis single link *list* yang tidak pernah mempunyai *tail* atau tidak pernah NIL selalu berputar Head = Tail;

c. *Header Circular Single Link List* : Jenis *circular single link list* yang memiliki simpul tambahan di awal sebagai informasi tambahan. Contoh dibawah ini merupakan *circular header single link list* yang pada simpul headernya berisi informasi mengenai banyaknya simpul di dalam *list.*

2. *Double Link List* / *Link list* dua arah (*Two Way List*)

Link List ini memiliki dua buah *field* yang digunakan untuk menunjuk ke simpul sebelumnya dan ke simpul sesudahnya. Banyak digunakan untuk mempermudah proses pencarian simpul dalam suatu seranai beranai. *Link list* ini memiliki beberapa variasi lain diantaranya :

a. *Header Double Link List* : Jenis *double link list* yang memiliki simpul tambahan pada awal simpul yang berguna untuk informasi tambahan.
b. *Circular Double Link List* : Jenis *double link list* yang tidak pernah mempunyai *tail* atau tidak pernah NIL selalu berputar Head = Tail;
c. *Header Circular Double Link List* : Jenis *circular double link list* yang memiliki simpul tambahan di awal sebagai informasi tambahan.