**ARRAY DINAMIS**

Sebelumnya telah dijelaskan mengenai variable bertipe array (array statis), suatu tipe data yang bersifat statis (urutan dan ukuran sudah pasti). Kelemahan dari array statis adalah penggunaan ruang memori yang sudah digunakan tidak dapat dihapus apabila nama variable array tersebut sudah tidak digunakan kembali dalam suatu program (penyebab kemubaziran).

Untuk pemecahannya maka digunakan struktur data dinamis dengan menggunakan variable dinamis. Variabel dinamis tidak dapat dideklarasikan secara eksplisit seperti halnya variable statis dan tidak dapat ditunjuk oleh identifier secara langsung, tetapi dapat ditunjuk secara khusus oleh variable dinamis yaitu **POINTER.**

Deklarasi secara umum untuk tipe data POINTER adalah sebagai berikut :

Type

Pengenal = ↑Simpul

Simpul = Type

1.

2.

(Nama var) : (↑Type data)

Keterangan :

Pengenal : Nama pengenal yang menyatakan data berupa pointer

Simpul : Menyatakan nama simpul

Type : Tipe data dari simpul

↑ : Tanda yang menyatakan bahwa pengenal memiliki tipe data pointer

**Contoh** **:**

* 1. Kamus :

Type

Point = ↑Data

Data = Record

< Nama\_Mhs : String,

Jurusan : String,

Semester : String >

Endrecord

DataMhs : Point

* 1. Kamus :

Jumlah\_data : ↑Integer

Nama\_Siswa : ↑String

Penjelasan:

* DataMhs merupakan variabel bertipe pointer
* Ketika program dikompilasi, variable DataMhs akan menempati lokasi tertentu dalam memori dan variabel tersebut belum menunjuk kesuatu simpul.
* Pointer yang belum menunjuk kesuatu simpul, nilainya dinyatakan sebagai nil guna mengalokasikan simpul dalam memori.
* Statemen yang digunakan untuk membuat sebuah simpul adalah **Alloc (variabel)**
* Catatan : Variabel merupakan nama peubah yang bertipe pointer
* Misal : Alloc (DataMhs); {Berarti kita membuat sebuah simpul yang ditunjuk oleh DataMhs}

Nil

|  |  |  |  |
| --- | --- | --- | --- |
| ? | ? | ? |  |

DataMhs

Keterangan : Masih tanda tanya karena belum terisi data

**Operasi pada Pointer**

1. Mengcopy pointer, artinya bahwa sebuah simpul akan ditunjuk oleh lebih dari sebuah pointer. Contoh :

Kamus :

Type

Point = ↑Data

Data = Record

< Nama\_Mhs : String

Jurusan : String>

Endrecord

T1, T2 : Point;

T1↑.NamaMhs ‘Benny’

T1↑.Jurusan ‘Informatika’

|  |  |  |
| --- | --- | --- |
| ? | ? |  |

Alloc (T1); T1

|  |  |  |
| --- | --- | --- |
| ? | ? |  |

Alloc (T2); T2

Akibatnya simpul diatas menjadi :

|  |  |  |
| --- | --- | --- |
| Benny | Informatika |  |

T1

|  |  |  |
| --- | --- | --- |
| ? | ? |  |

T2

Apabila T2  T1

|  |  |  |
| --- | --- | --- |
| Benny | Informatika |  |

T1

|  |  |  |
| --- | --- | --- |
| ? | ? |  |

T2

1. Mengcopy isi simpul, artinya bahwa dua/lebih simpul ditunjuk leh pointer yang berbeda tetapi mempunyai isi yang sama. Apabila T2 ↑ T1↑; maka dapat diillustrasikan sebagai berikut:

|  |  |  |
| --- | --- | --- |
| Benny | Informatika |  |

T1

|  |  |  |
| --- | --- | --- |
| Benny | Informatika |  |

T2

1. Menghapus pointer

Apabila pointer dihapus, maka lokasi semula yang ditempati oleh simpul yang ditunjuk oleh pointer tersebut akan bebas sehingga bisa digunakan oleh variable lain.

Bentuk : Dealloc (Variabel)

Misal : Dealloc (T1)

**LINKED LIST**

* Struktur ini terdiri dari rangkaian elemen yang saling berhubungan / berkaitan, dimana setiap elemen dihubungkan dengan elemen lainnya oleh sebuah pointer.
* Pointer, sel yang nilainya merupakan alamat sel yang lain dimana sel yang lain itu dapat berupa data atau berupa pointer juga
* Setiap elemen dalam linked list selalu berisi pointer
* Deklarasi Linked List

Type

Point = ↑Data

Data = Record

<Info : char

Next : Point >

Endrecord

awal, akhir: Point

Type

nama\_pointer = ↑Simpul

Simpul = Record

medan\_data : tipedata

medan\_sambungan : Namapointer

EndRecord

nama\_var\_pointer : nama\_pointer

Istilah – istilah

* Simpul, terdiri dari dua bagian :

1. Bagian/medan data (info)
2. Bagian/medan sambungan (pointer yang menunjuk kesimpul berikutnya)

* Awal (First), variable yang berisi alamat yang menunjuk lokasi simpul pertama linked list

|  |  |
| --- | --- |
| A |  |

Awal

* Nil / Null, Tidak bernilai yaitu menyatakan tidak mengacu kealamat manapun.

|  |
| --- |
|  |

Nil

* Akhir, sebuah simpul yang tidak menunjuk pada simpul terakhir

|  |  |
| --- | --- |
| A |  |

|  |  |
| --- | --- |
| B |  |

Awal

Akhir

* Linked List kosong dikenali dengan

Awal 🡨 nil

Awal

* Elemen terakhir linked list dikenali dengan : Akhir↑.Next 🡨 Nil
* Linked List terdiri dari 3 macam yaitu :

1. Single Linked List
2. Double Linked List
3. Circular Linked List
   1. **Single Linked List**

Adalah linked list dengan simpul berisi satu link / pointer yang mengacu ke simpul berikutnya. Skema Simpul Single Linked List :

|  |  |
| --- | --- |
|  |  |

Operasi – operasi pada single linked list :

* 1. Penciptaan

Awal dan akhir diberi nil.

Akhir

Awal

* 1. Penyisipan

1. Penyisipan didepan
   * + **List kosong {Awal=nil}**

|  |  |
| --- | --- |
|  |  |

Baru

|  |  |
| --- | --- |
| 1 |  |

Baru

Akhir

|  |  |
| --- | --- |
| 1 |  |

Awal

Baru

* + - **List tidak kosong {Awal ≠ Nil}**

Mula-mula keadaan list sebagai berikut:

Akhir

|  |  |
| --- | --- |
| 1 |  |

Awal

|  |  |
| --- | --- |
| 3 |  |

|  |  |
| --- | --- |
| 2 |  |

|  |  |
| --- | --- |
| 4 |  |

Baru

Akhir

|  |  |
| --- | --- |
| 1 |  |

Awal

|  |  |
| --- | --- |
| 3 |  |

|  |  |
| --- | --- |
| 2 |  |

|  |  |
| --- | --- |
| 3 |  |

|  |  |
| --- | --- |
| 2 |  |

|  |  |
| --- | --- |
| 4 |  |

Baru

Akhir

Awal

|  |  |
| --- | --- |
| 4 |  |

|  |  |
| --- | --- |
| 3 |  |

|  |  |
| --- | --- |
| 2 |  |

|  |  |
| --- | --- |
| 1 |  |

Baru

1. Penyisipan di tengah
   * + List Kosong {Awal = Nil} (Sama dengan penyisipan di depan)
     + List tidak kosong {Awal ≠ Nil}

**Mula-mula keadaan list sebagai berikut**:

Akhir

|  |  |
| --- | --- |
| 1 |  |

Awal

|  |  |
| --- | --- |
| 4 |  |

|  |  |
| --- | --- |
| 2 |  |

Asumsi data : 3 akan disisipkan setelah data ‘4’

|  |  |
| --- | --- |
| 3  Bantu |  |

Baru

Akhir

|  |  |
| --- | --- |
| 1 |  |

Awal

|  |  |
| --- | --- |
| 4 |  |

|  |  |
| --- | --- |
| 2 |  |

|  |  |
| --- | --- |
| 3 |  |

Baru

Bantu

Akhir

|  |  |
| --- | --- |
| 1 |  |

Awal

|  |  |
| --- | --- |
| 4 |  |

|  |  |
| --- | --- |
| 2 |  |

|  |  |
| --- | --- |
| 3 |  |

Baru

Akhir

Awal

|  |  |
| --- | --- |
| 4 |  |

|  |  |
| --- | --- |
| 3 |  |

|  |  |
| --- | --- |
| 2 |  |

|  |  |
| --- | --- |
| 1 |  |

1. Penyisipan di akhir
   * + List kosong {Awal = Nil} (Sama dengan penyisipan di depan)
     + List tidak kosong (Awal ≠ Nil}

**Mula-mula keadaan list sebagai berikut**:

Akhir

|  |  |
| --- | --- |
| 3 |  |

|  |  |
| --- | --- |
| 2 |  |

Awal

|  |  |
| --- | --- |
| 1 |  |

|  |  |
| --- | --- |
| 4 |  |

Baru

Akhir

|  |  |
| --- | --- |
| 3 |  |

Awal

|  |  |
| --- | --- |
| 1 |  |

|  |  |
| --- | --- |
| 2 |  |

|  |  |
| --- | --- |
| 4 |  |

Baru

Akhir

Baru

|  |  |
| --- | --- |
| 1 |  |

|  |  |
| --- | --- |
| 2 |  |

|  |  |
| --- | --- |
| 3 |  |

|  |  |
| --- | --- |
| 4 |  |

Awal

Procedure sisip\_depan\_single(Input elemen : tipedata, I/O awal, akhir : nama\_pointer)

{I.S. : data yang akan disisipkan (elemen), pointer penunjuk awal dan pointer penunjuk akhir sudah terdifinisi}

{F.S. : menghasilkan satu simpul yang disisipkan di depan pada single linked list}

Kamus :

baru : nama\_pointer

Algoritma :

Alloc(baru)

baru↑.info 🡨 elemen

If (awal = nil)

Then

baru↑.next 🡨 nil

akhir 🡨 baru

Else

baru↑.next 🡨 awal

EndIf

awal 🡨 baru

EndProcedure

Procedure sisip\_belakang\_single(Input elemen : tipedata, I/O awal, akhir : nama\_pointer)

{I.S. : data yang akan disisipkan (elemen), pointer penunjuk awal dan pointer penunjuk akhir sudah terdifinisi}

{F.S. : menghasilkan satu simpul yang disisipkan di belakang pada single linked list}

Kamus :

baru : nama\_pointer

Algoritma :

Alloc(baru)

baru↑.info 🡨 elemen

baru↑.next 🡨 nil

If (awal = nil)

Then

awal 🡨 baru

Else

akhir↑.next 🡨 baru

EndIf

akhir 🡨 baru

EndProcedure

Procedure sisip\_tengah\_single(Input elemen : tipedata, I/O awal, akhir : nama\_pointer)

{I.S. : data yang akan disisipkan (elemen), pointer penunjuk awal dan pointer penunjuk akhir sudah terdifinisi}

{F.S. : menghasilkan satu simpul yang disisipkan di tengah pada single linked list}

Kamus :

baru,bantu : nama\_pointer

ketemu : boolean

datasisip : tipedata

Algoritma :

If (awal = nil)

Then

Alloc(baru)

baru↑.info 🡨 elemen

baru↑.next 🡨 nil

awal 🡨 baru

akhir 🡨 baru

Else

Input(datasisip)

bantu 🡨 awal

ketemu 🡨 false

While (not ketemu and bantu ≠ nil) do

If (datasisip = bantu↑.info)

Then

ketemu 🡨 true

Else

bantu 🡨 bantu↑.next

EndIf

EndWhile

If (ketemu)

Then

Alloc(baru)

baru↑.info 🡨 elemen

If (bantu = akhir)

Then

sisip\_belakang\_single(elemen,awal,akhir)

Else

baru↑.next 🡨 bantu↑.next

bantu↑.next 🡨 baru

EndIf

Else

Output(“Data yang akan disisipkan tidak ada”);

EndIf

EndIf

EndProcedure

Procedure isi\_elemen(Output elemen : tipedata)

{I.S. : user memasukkan data yang akan disisipkan (elemen)}

{F.S. : mengahsilkan data yang akan disisipkan (elemen)}

Kamus :

Algoritma :

Input(elemen)

EndProcedure

{Algoritma Utama}

Algoritma\_penyisipan\_Single\_Linked\_List

Kamus :

{prototype}

Procedure isi\_elemen(Output elemen : tipedata)

Procedure sisip\_depan\_single(Input elemen : tipedata, I/O awal, akhir : nama\_pointer)

Procedure sisip\_belakang\_single(Input elemen : tipedata, I/O awal, akhir : nama\_pointer)

Procedure sisip\_tengah\_single(Input elemen : tipedata, I/O awal, akhir : nama\_pointer)

{deklarasi}

nama\_pointer = ↑simpul

simpul = Record

< info : tipedata,

next : nama\_pointer>

EndRecord

awal, akhir : nama\_pointer {pointer penunjuk}

menu : integer

elemen : tipedata

Algoritma :

{penciptaan list}

awal 🡨 nil

akhir 🡨 nil

{penyisipan}

Repeat

output(“Menu Pilihan”)

output(“==========”)

output(“1. Sisip Depan”)

output(“2. Sisip Tengah”)

output(“3. Sisip Belakang”)

output(“0. Keluar”)

Input(menu)

Depend on (menu)

(menu = 1) : isi\_elemen(elemen)

sisip\_depan\_single(elemen, awal, akhir)

(menu = 2) : isi\_elemen(elemen)

sisip\_tengah\_single(elemen, awal, akhir)

(menu = 3) : isi\_elemen(elemen)

sisip\_belakang\_single(elemen, awal, akhir)

EndDepend

Until (menu = 0)

* 1. Penghapusan

1. **Penghapusan di awal**

Penghapusan data di awal adalah proses menghapus elemen pertama (awal), sehingga variable awal akan berpindah ke elemen data berikutnya. Ada 3 kondisi yang perlu diperhatikan yaitu kondisi linked list masih kosong, kondisi linked list hanya memiliki 1 data, dan kondisi linked list yang memiliki data lebih dari 1 elemen

* Kondisi linked list kosong



Pada kondisi ini proses penghapusan tidak bisa dilakukan

* Kondisi linked list memiliki hanya 1 data{Satu simpul}

Akhir

|  |  |
| --- | --- |
| A |  |

Awal

Berikan satu variable Bantu yang sama menunjuk ke simpul awal/akhir

Awal

Akhir

|  |  |
| --- | --- |
| A |  |

Bantu

Kemudian berikan nilai null untuk variable awal dan variable akhir

Simpan terlebih dahulu nilai dari elemennya kedalam suatu variabel

|  |  |
| --- | --- |
| A |  |

Bantu Awal Akhir

|  |  |
| --- | --- |
| A |  |

Bantu

Setelah itu simpul Bantu dapat dihapus

* Kondisi linked list memiliki data lebih dari 1 data



Kemudian alamat data awal diisikan ke suatu variabel pembantu (phapus).



setelah itu pindahkan awal ke data berikutnya.



Setelah itu hapus/hancurkan data di posisi phapus. Sehingga linked list menjadi seperti di bawah ini.



1. **Penghapusan di tengah**

Penghapusan akan dilakukan pada data 3 dari banyak data sebanyak 4.



Kemudian cari posisi elemen sebelum elemen posisi hapus, kemudian simpan dalam variabel bantu.



Kemudian simpan alamat elemen posisi hapus dalam suatu variabel dengan nama bantu2.



Kemudian pindahkan field next dari bantu ke alamat yang ditunjuk oleh field next dari bantu2.



Hapus elemen data yang ditunjuk dengan bantu2.



Setelah langkah tersebut, maka elemen telah terhapus.



1. **Penghapusan di akhir**

Kondisi linked list memiliki lebih dari 1 data.



Karena posisi hapus adalah data terakhir, maka nanti posisi akhir harus pindah ke posisi sebelumnya. Oleh karena itu harus dicari posisi data sebelum data terakhir, sebut dengan variabel bantu.



Kemudian pindahkan variable akhir ke simpul yang ditunjuk oleh varibel Bantu dan variable Bantu pindahkan ke simpul yang ditunjuk oleh variable akhir field next-nya.

Akhir

Awal

Bantu

|  |  |
| --- | --- |
| 5 |  |

|  |  |
| --- | --- |
| 9 |  |

|  |  |
| --- | --- |
| 15 |  |

|  |  |
| --- | --- |
| 20 |  |

Simpan elemen variable simpul Bantu. Kemudian lakukan penghapusan pada simpul yang ditunjuk variable Bantu, sehingga hasil akhirnya seperti dibawah ini.



4. **Penelusuran Single Linked List**

Penelusuran berarti menampilkan semua data yang ada di dalam linked list dari posisi awal sampai dengan akhir. Untuk itu diperlukan suatu variable pembantu (sebut saja variable p) yang akan menelusuri data sampai data terakhir.

Langkah-langkah penelusuran adalah :

* Isi variable p dengan awal.
* Selama p tidak NULL, maka tampilkan info yang ada di elemen yang ditunjuk variable p, kemudian p dipindahkan ke elemen berikutnya

5. **Pencarian**

Langkah-langkah untuk melakuan pencarian data dalam linked list tidak begitu beda dengan langkah-langkah pencarian data pada array. Karena dengan linked list tidak dapat diakses secara acak, maka pencarian yang dilakukan adalah pencarian secara sekuensial.

Contoh : Data yang akan dicari = 9



Berarti diperlukan pointer bantuan (Bantu) untuk menelusuri simpul awal sampai dengan simpul terakhir untuk mendapatkan data yang dicari.



Setelah simpul ditelusuri data 9 akan ditemukan pada simpul yang ke 2

6. **Pengurutan**

Langkah pengurutan data dalam linked list sama saja dengan pengurutan data dalam array. Berikut ini adalah implementasi pengurutan data dalam linked list dengan algoritma bubble sort dan selection sort.

Contoh : Pengurutan menggunakan buble sort secara ascending



Berikan variable bantu1 yang menunjuk ke simpul awal dan variable bantu2 yang menunjuk ke simpul yang ditunjuk oleh bantu1 field next-nya.



Kemudian bandingkan nilai yang ada pada bantu1 field info dengan nilai yang ada pada bantu2 field info, jika nilainya lebih besar maka tukarkan nilainya.



Pindahkan variabel bantu2 ke bantu2 field next-nya dan variable bantu1 ke bantu1 field-nextnya kemudian bandingkan kembali nilai yang ada pada field infonya, jika nilainya lebih besar maka tukarkan kembali nilainya.



Hal ini terus berulang selama bantu2 tidak sama dengan nil (kosong) dan bantu1 field next-nya tidak sama dengan nil sehingga data yang ada pada single linked list akan terurut.

1. Penghapusan diAwal

If (Awal = nil) then

Output(‘Tidak ada penghapusan, List kosong… ’)

Else

If (Awal = Akhir) then

Phapus🡨Awal

Awal🡨nil

Akhir🡨nil

Dealloc(phapus)

Else

Phapus🡨Awal

Awal🡨phapus↑.next

X🡨phapus↑.info

Dealloc(phapus)

Endif

Endif

1. Penghapusan di tengah
2. Penghapusan di akhir

If (Awal = nil) then

Output(‘Penghapusan tidak terjadi, list kosong… ’)

Else

If (Awal = Akhir) then

Phapus🡨Awal

Awal🡨nil

Akhir🡨nil

Dealloc(phapus)

Else

Phapus🡨awal

While (phapus↑.next ≠ akhir) do

Phapus🡨phapus↑.next

EndWhile

Akhir🡨phapus

Phapus🡨phapus↑.next

X🡨phapus↑.info

Dealloc(phapus)

Endif

Endif

1. Pencarian

Input (x)

Ditemukan 🡨 False

If (awal ≠ nil) then

i 🡨 1

Bantu 🡨 awal

While ((Bantu ≠ nil ) And (Not(ditemukan)) do

If (Bantu↑.info = x) then

Ditemukan🡨true

Else

Bantu🡨Bantu↑.next

Endif

i🡨i+1

Endwhile

Else

Output(‘List Kosong..’)

Endif

1. Pengurutan

If (Awal ≠ nil) then

Bantu 🡨 Awal

While (Bantu1↑.next ≠ nil) do

Bantu2🡨Bantu1↑.next

While (Bantu2≠nil) do

If (Bantu1↑.info > Bantu2↑.info) then

Temp🡨 Bantu1↑.info

Bantu1↑.info🡨 Bantu12↑.info

Bantu2↑.info🡨Temp

Endif

Bantu2🡨Bantu2↑.next

Endwhile

Bantu1🡨Bantu1↑.Next

Endwhile

Else

Output(‘List Kosong…..’)

Endif

* 1. **Double Linked List**

Double Linked List adalah suatu linked list yang mempunyai 2 penunjuk yaitu penunjuk ke data sebelumnya dan berikutnya. Perhatikan gambar di bawah ini :



Deklarasi secara umum double linked list :

Type

Point = Data

Data = Record

<Info : char

Prev, Next : point >

Endrecord

Awal, Akhir: Point

Jika dilihat 1 elemen listnya, maka secara umum struktur dari elemen listnya adalah sebagai berikut :



Dari gambar di atas, untuk setiap elemen terdiri dari 3 buah field yaitu kiri (prev), info (data), dan kanan (next). Field kiri dan kanan merupakan sebuah pointer ke data struktur elemen (Data).

Beberapa operasi yang dapat dilakukan dalam double linked list adalah :

1. Penciptaan

Memberikan awal dan akhir dengan nilai nil. Kondisi awal ketika awal dan akhir telah dideklarasikan.



1. Penyisipan
   1. Penyisipan awal

Operasi ini berguna untuk menambahkan elemen baru di posisi pertama. Langkah pertama untuk penambahan data adalah pembuatan elemen baru dan pengisian nilai info-nya. Pointer yang menunjuk ke data tersebut dipanggil dengan nama **baru**. Kondisi setelah ada pembuatan elemen baru tersebut adalah :



Ada 2 kondisi yang harus diperhatikan dalam penambahan data di awal yaitu :

* 1. Ketika linked list masih kosong

Kalau kondisi linked list masih kosong, maka elemen baru akan menjadi awal dan akhir linked list. Perhatikan gambar di bawah ini :

* Kondisi sebelum disisipkan



* Kondisi setelah operasi penambahan

Operasi penambahan awal ketika linked list masih kosong adalah dengan mengisikan alamat pointer baru ke pointer awal dan pointer akhir. Dan memberikan nilai nil pada Baru↑.next (Baru↑.next🡨nil). Lihat gambar di bawah ini



* 1. Ketika linked list sudah mempunyai data

Kondisi linked list ketika sudah mempunyai data elemen dan elemen yang baru telah dibuat, dapat dilihat pada gambar di bawah ini.



Proses penambahan data di awal linked list adalah :

* + Hubungkan Baru↑.next agar menunjuk ke awal



* + Hubungkan Awal↑.prev agar menunjuk ke posisi pointer baru



* + Pindahkan pointer awal ke pointer baru



* 1. Penyisipan tengah

Operasi penyisipan data di tengah linked list adalah suatu operasi menambah data di posisi tertentu di dalam linked list. Contohnya adalah jika ingin menyisipkan data di posisi ke-3 atau ke-4.

Untuk proses tersebut ada 2 hal yang harus diperhatikan yaitu :

* + - 1. Kondisi linked list masih kosong atau posisi penyisipan kurang dari/sama dengan 1

Jika kondisi ini terjadi, maka langkah yang dilakukan adalah sangat mudah yaitu dengan memanggil proses penambahan data awal atau akhir. (untuk lebih jelas lihat penambahan data awal atau akhir ketika kondisi linked list masih kosong).

* + - 1. Kondisi linked list sudah mempunyai data

Langkah untuk penyisipan data ketika linked list sudah mempunyai data adalah:

* + Cari posisi pointer pada data ke-***posisi sisip***.

Caranya adalah dengan menelusuri linked list sebanyak ***posisi sisip*** kali. Contoh linked list :



Ketika pencarian posisi pointer pada data ke-***posisi sisip*** dicari, maka ada dua kemungkinan yaitu posisi sisip ada di dalam linked list atau diluar linked list (kalau mengisi posisi lebih besar dari banyak data). Oleh karena itu pencarian pointer posisi sisip ada dua kemungkinan. Perhatikan contoh di bawah ini.

**Contoh 1** : Penyisipan di posisi 3 dengan data yang telah ada adalah 4 buah.

* + - Pointer bantu diisi dengan data awal, pos diisi 1 (data pertama)



* + - Jika pos belum sama dengan posisi sisip, maka pindah ke data berikutnya serta pos ditambah 1.



* + - Karena pos masih lebih kecil dari posisi sisip, maka pindahkan pos dan bantu ke posisi berikutnya dan pos ditambah 1.



* + - Karena pos telah sama dengan posisi sisip maka perulangan pencarian telah selesai. Itu menunjukan posisi penyisipan adalah di posisi yang ditunjuk oleh bantu.

**Contoh 2** : Penyisipan di posisi 10 dengan data yang telah ada adalah 4 buah.

* + - Pointer bantu diisi dengan data awal, pos diisi 1 (data pertama)



* + - Jika pos belum sama dengan posisi sisip, maka pindah ke data berikutnya dan pos ditambah 1.



* + - Karena pos masih lebih kecil dari posisi sisip, maka pindahkan pointer bantu ke posisi berikutnya dan variabel pos ditambah 1.



* + - Karena pos masih lebih kecil dari posisi sisip, maka pindahkan pos dan bantu ke posisi berikutnya dan pos ditambah 1.



* + - Karena pos masih lebih kecil dari posisi sisip, maka pindahkan pos dan bantu ke posisi berikutnya dan pos ditambah 1.



* + - Karena bantu mencapai nilai NULL, maka perulangan harus berhenti karena itu menunjukan posisi sisip ada di luar linked list. Kalau ini terjadi, maka penambahan dapat dilakukan di posisi terakhir.
  + Setelah posisi penyisipan (bantu) ditemukan, maka periksa apakah posisi penyisipan (bantu) bernilai NULL atau tidak. Jika posisi penyisipan (bantu) bernilai NULL berarti posisi sisip berada di luar atau melebihi linked list. Oleh karena itu berarti penambahan datanya dilakukan dengan melakukan operasi penambahan akhir.
  + Jika posisi penyisipan (bantu) tidak sama dengan NULL, maka itu berarti posisi penyisipan ada di dalam jangkauan linked list. Proses yang dilakukan untuk penyisipannya adalah :
    - Buat elemen baru di memori dan isi infonya (contoh data info = 8).



* + - Untuk mempermudah proses penyambungan/penyisipan data baru, maka buat variabel pointer baru dengan nama bantu2 untuk memegang data di sebelah kiri dari posisi sisip (bantu↑.prev). (Contoh posisi penyisipan adalah 3)



* + - Isi baru↑.prev dengan pointer bantu2



* + - Isi/sambungkan baru↑.next ke pointer bantu



* + - Isi/sambungkan bantu↑.prev ke posisi pointer baru



* + - Isi/sambungkan bantu2↑.next ke posisi pointer baru



* 1. Penyisipan akhir

Operasi ini berguna untuk menambahkan elemen baru di posisi akhir. Langkah pertama untuk penambahan data adalah pembuatan elemen baru dan pengisian nilai infonya. Pointer yang menunjuk ke data tersebut dipanggil dengan nama **baru**. Kondisi di setelah ada pembuatan elemen baru tersebut adalah :



Ada 2 kondisi yang harus diperhatikan dalam penambahan data di akhir yaitu :

* + - 1. Ketika linked list masih kosong

Kalau kondisi linked list masih kosong, maka elemen baru akan menjadi awal dan akhir linked list. Perhatikan gambar di bawah ini :

* Kondisi sebelum penambahan



* Kondisi setelah operasi penambahan

Operasi penambahan awal ketika linked list masih kosong adalah dengan mengisikan alamat pointer baru ke pointer awal dan pointer akhir. Lihat gambar di bawah ini.



* + - 1. Ketika linked list sudah mempunyai data

Kondisi linked list ketika sudah mempunyai data elemen dan elemen yang baru telah dibuat, dapat dilihat di gambar di bawah ini.



Proses penambahan data di akhir linked list adalah :

* + Hubungkan akhir↑.next agar menunjuk ke pointer baru



* + Hubungkan baru↑.prev agar menunjuk ke posisi pointer akhir



* + Pindahkan pointer akhir ke pointer baru



1. Penghapusan
   1. Penghapusan di awal

Operasi ini berguna untuk menghapus data pada posisi pertama. Ada 3 keadaan yang mungkin terjadi ketika akan melakukan proses hapus yaitu :

* + 1. Kondisi linked list masih kosong

Jika kondisi ini terjadi, maka proses penghapusan data tidak bisa dilakukan karena linked list masih kosong.

* + 1. Kondisi linked list hanya memiliki 1 data

Langkah yang perlu dilakukan ketika ingin melakukan proses penghapusan linked list yang memiliki hanya 1 data adalah dengan langsung menghapus data dari memori dan kemudian pointer awal dan akhir di-NULL-kan. Untuk lebih jelas perhatikan urutan penghapusannya di bawah ini :

* + Kondisi data sebelum dihapus



* + Proses penghapusan yaitu dengan menghilangkan data dari memori dengan perintah free(awal) atau free(akhir).

|  |  |  |
| --- | --- | --- |
|  | menjadi |  |

Kemudian pointer awal dan akhir diisi dengan NULL.



c. Kondisi linked list memiliki data lebih dari 1 buah

Untuk operasi penghapusan data di posisi pertama pada double linked list yang mempunyai data lebih dari 1 buah adalah :

* + Simpan pointer yang akan dihapus (awal) ke suatu pointer lain yang diberi nama pointer **bantu**.



* + Pindahkan pointer awal ke data berikutnya (bantu↑.next atau Awal↑.next).



* + Field kiri dari awal yang baru (Awal↑.prev) di-NULL-kan.



* + Langkah terakhir adalah hapus elemen yang ditunjuk pointer bantu.



* + Setelah data dihapus, maka kondisi linked list adalah seperti di gambar di bawah ini.



* 1. Penghapusan di tengah

Untuk melakukan proses penghapusan di tengah linked list, ada 3 kondisi yang perlu diperhatikan yaitu :

* + 1. Kondisi ketika linked list masih kosong atau ketika posisi hapus lebih kecil dari 1.

Ketika kondisi ini terjadi, maka proses penghapusan tidak bisa dilakukan karena data masih kosong atau karena posisi hapus diluar jangkauan linked list (posisi kurang dari 1).

* + 1. Kondisi ketika posisi hapus sama dengan 1 (hapus data pertama)

Ketika kondisi ini terjadi, maka proses yang dilakukan adalah proses penghapusan di posisi awal (hapusawal).

* + 1. Kondisi ketika posisi hapus lebih besar dari 1

Langkah-langkah untuk penghapusan data di tengah linked list yang posisi hapusnya lebih besar dari 1 adalah :

* + Cari pointer yang menunjuk ke data pada posisi ke-***posisi hapus***. Ketika kondisi ini, ada 2 kemungkinan yang bisa terjadi yaitu posisi hapus ada di dalam jangkauan linked list atau di luar linked list. Untuk lebih jelas perhatikan 2 contoh di bawah ini :
    - **Contoh 1** : Kondisi posisi hapus ada di dalam jangkauan linked list, contoh penghapusan pada posisi 3.
    - Pointer bantu diisi dengan data awal, pos diisi 1 (data pertama)



* + - Jika pos belum sama dengan posisi hapus, maka pindah ke data berikutnya dan pos ditambah 1.



* + - Karena pos masih lebih kecil dari posisi hapus, maka pindahkan bantu ke posisi berikutnya dan pos ditambah 1.



* + - Karena pos telah sama dengan posisi hapus maka perulangan pencarian posisi penghapusan telah selesai. Itu menunjukan posisi penghapusan adalah di posisi yang ditunjuk oleh bantu.
    - **Contoh 2** : Kondisi posisi hapus ada di luar/melebihi jangkauan linked list, contoh penghapusan pada posisi 10 tetapi dalam linked list hanya ada 4 buah data.
    - Pointer bantu diisi dengan data awal, pos diisi 1 (data pertama)



* + - Jika pos belum sama dengan posisi hapus, maka pindah ke data berikutnya dan pos ditambah 1.



* + - Karena pos masih lebih kecil dari posisi hapus, maka pindahkan pointer bantu ke posisi berikutnya dan variabel pos ditambah 1.



* + - Karena pos masih lebih kecil dari posisi hapus, maka pindahkan pos dan bantu ke posisi berikutnya dan pos ditambah 1.



* + - Karena pos masih lebih kecil dari posisi hapus, maka pindahkan pos dan bantu ke posisi berikutnya dan pos ditambah 1.



* + - Karena bantu mencapai nilai NULL, maka perulangan harus berhenti karena itu menunjukan posisi hapus ada di luar linked list. Kalau ini terjadi, maka penghapusan tidak dapat dilakukan
  + Setelah pointer posisi penghapusan telah ditemukan, maka langkah selanjutnya adalah pemeriksaan apakah posisi penghapusan (bantu) tersebut bernilai NULL (diluar jangkauan linked list). Jika posisi penghapusan (bantu) bernilai NULL maka proses penghapusan tidak bisa dilakukan. Tetapi jika bantu tidak bernilai NULL, maka lanjutkan ke langkah berikutnya.
  + Periksa juga apakah pointer posisi penghapusan (bantu) sama dengan posisi akhir, jika benar maka proses penghapusan yang dilakukan adalah proses penghapusan akhir.
  + Tetapi jika posisi penghapusan tidak sama dengan posisi akhir itu menunjukan posisi penghapusan ada di tengah, maka proses yang dilakukan adalah :
    - Untuk mempermudah penghapusan, maka simpan pointer yang menunjuk ke data sebelum posisi penghapusan (bantu2=bantu↑.prev) dan data setelah posisi penghapusan (bantu3= bantu↑.next).



* + - Isi/sambungkan bantu2↑.next ke posisi pointer bantu3.



* + - Isi/sambungkan bantu3↑.prev ke posisi pointer bantu2.



* + - Hapus dari memori data yang ditunjuk oleh pointer bantu.



* + - Setelah data di pointer bantu dihapus, maka kondisi linked list adalah



* 1. Penghapusan di akhir

Operasi ini berguna untuk menghapus data pada posisi terakhir. Ada 3 keadaan yang mungkin terjadi ketika akan melakukan proses hapus yaitu :

* + 1. Kondisi linked list masih kosong

Jika kondisi ini terjadi, maka proses penghapusan data tidak bisa dilakukan karena linked list masih kosong.

* + 1. Kondisi linked list hanya memiliki 1 data

Langkah yang perlu dilakukan ketika ingin melakukan proses penghapusan linked list yang memiliki hanya 1 data adalah dengan langsung menghapus data dari memori dan kemudian pointer awal dan akhir di-NULL-kan. Untuk lebih jelas perhatikan urutan penghapusannya di bawah ini :

* + Kondisi data sebelum dihapus



* + Proses penghapusan yaitu dengan menghilangkan data dari memori dengan perintah free(awal) atau free(akhir).

|  |  |  |
| --- | --- | --- |
|  | menjadi |  |

Kemudian pointer awal dan akhir diisi dengan NULL.



* + 1. Kondisi linked list memiliki data lebih dari 1 buah

Untuk operasi penghapusan data di posisi terakhir pada double linked list yang mempunyai data lebih dari 1 buah adalah :

* + Simpan pointer yang akan dihapus (akhir) ke suatu pointer lain yang diberi nama pointer **bantu**.



* + Pindahkan pointer akhir ke data sebelumnya (bantu↑.prev atau akhir↑.prev).



* + Field kanan dari akhir baru (akhir↑.next) di-NULL-kan.



* + Langkah terakhir adalah hapus elemen yang ditunjuk pointer bantu.



* + Setelah data dihapus, maka kondisi linked list adalah seperti di gambar di bawah ini.



1. Pencarian Data

Pencarian dilakukan dengan memeriksa data yang ada dalam linked list dengan data yang dicari. Pencarian dilakukan dari data pertama sampai data ditemukan atau pointer pencari (bantu) telah mencapai NULL yang menandakan bahwa data yang dicari tidak ditemukan.

Agar lebih jelas perhatikan ilustrasi di bawah ini, dengan contoh data adalah :



Ada 2 kondisi yang dihasilkan oleh proses pencarian yaitu

* 1. Pencarian dimana data yang dicari dapat ditemukan

Kasus : data yang akan dicari adalah data 9.

* + Isi bantu dengan pointer data awal dan posisi diisi 1



* + Jika data info yang ditunjuk oleh pointer bantu tidak sama dengan yang dicari, maka bantu pindah ke data berikutnya dan variabel posisi ditambah 1.



* + Karena info yang ditunjuk oleh bantu belum sama dengan yang dicari, maka bantu pindah lagi ke data berikutnya dan variabel posisi ditambah 1



* + Karena data info yang ditunjuk oleh bantu sama dengan data yang dicari, maka pencarian selesai dan data ditemukan pada lokasi yang dimiliki oleh variabel posisi.
  1. Pencarian dimana data yang dicari tidak ditemukan

Kasus : data yang dicari adalah 20.

* + Isi bantu dengan pointer data awal dan posisi diisi 1



* + Jika data info yang ditunjuk oleh pointer bantu tidak sama dengan yang dicari, maka bantu pindah ke data berikutnya dan variabel posisi ditambah 1.



* + Karena info yang ditunjuk oleh bantu belum sama dengan yang dicari, maka bantu pindah lagi ke data berikutnya dan variabel posisi ditambah 1



* + Karena info yang ditunjuk oleh bantu belum sama dengan yang dicari, maka bantu pindah lagi ke data berikutnya dan variabel posisi ditambah 1



* + Karena info yang ditunjuk oleh bantu belum sama dengan yang dicari, maka bantu pindah lagi ke data berikutnya dan variabel posisi ditambah 1



Karena bantu bernilai NULL maka pencarian tidak perlu dilanjutkan lagi karena itu berarti data tidak ditemukan

1. Pengurutan Data

Langkah-langkah untuk pengurutan data pada data yang berbentuk double linked list dapat dilihat di bawah ini.

Contoh data :



Proses pengurutannya adalah :

1. Langkah ke-1 adalah pegang/isi p1 dengan data pertama (awal). Langkah ini dilakukan untuk mencari nilai terkecil pertama, caranya dengan membandingkan data yang ditunjuk oleh p1 dengan data yang ditunjuk p2. Pointer p2 akan berjalan dari data setelah p1 sampai data terakhir. Jika terjadi kondisi info yang ditunjuk oleh p1 lebih besar dari data info yang ditunjuk oleh p2, maka lakukan pertukaran data info p1 dan p2.



Karena p1🡪info > dari p2🡪info maka lakukan pertukaran.



Kemudian pindahkan p2 ke data berikutnya.



Bandingkan p1🡪info dengan p2🡪info. Karena p1🡪info > p2🡪info maka lakukan pertukaran.



Pindahkan lagi p2 ke data berikutnya.



Karena p1🡪info > p2🡪info maka lakukan pertukaran.



Pindahkan lagi p2 ke data berikutnya.



Karena p2 telah mencapai NULL, maka berarti p2 telah mencapai akhir. Jika p2 telah mencapai akhir maka pindahkan p1 ke data berikutnya dan p2 dipindahkan ke data setelah p1.



1. Lakukan perulangan langkah 1 sebanyak banyak data – 1.
   1. **Circular Linked List**