

## MODUL VII & VIII

# STRUKTUR DATA

<b>Judul</b>	<b>LINKED LIST</b>	
<b>Penyusun</b>	<b>Distribusi</b>	<b>Perkuliahan</b>
<b>Nixon Erzed</b>	Teknik Informatika Universitas Esa Unggul	Pertemuan – VII & VIII

Tujuan :

Mahasiswa mengenal dan memahami representasi data dengan Singly Linked List

Materi :

- Variabel Pointer
- Definisi Linked List
- Model Singly Linked List
- Karakteristik Singly Linked List
- Deklarasi single Linked List
- Model deskriptif Singly Linked List
- Operasi Dasar
- Penciptaan
- Penghancuran simpul
- Inisialisasi linked list
- Pemeriksaan list kosong
- Varian Singly Linked List
- Operasi pada Linked List
  - Membangun Linked List
  - Penelusuran Linked List
  - Penyisipan simpul
    - sebagai simpul pertama
    - setelah simpul tertentu
    - sebagai simpul terakhir
  - Menghapus simpul

## VARIABEL POINTER

Walaupun array sangat bagus untuk mengimplementasikan random access, tapi sifatnya yang statis menyebabkan tidak array tidak dapat diterapkan pada berbagai aplikasi. Selain itu dalam manajemen memory umumnya array dialokasikan secara fixed, sehingga tidak dapat didealokasi ketika program masih dieksekusi. Artinya ruang memory yang dipakai oleh array yang sudah tidak digunakan lagi, tidak dapat dihapus atau dialokasikan untuk data lain, selama program masih dijalankan.

Untuk memecahkan masalah tersebut, dapat digunakan variabel pointer. Type data pointer bersifat dinamis, variabel akan dialokasikan hanya pada saat dibutuhkan dan setelah tidak dibutuhkan dapat didealokasikan kembali.

### Array vs Pointer

Pada tabel berikut ini diberikan perbedaan antara variabel bertipe array dan variabel bertipe pointer.

Kriteria	Array	Pointer
Sifat	Statis	Dinamis
Ukuran	Pasti	Sesuai kebutuhan
Alokasi variabel	Saat program dijalankan sampai selesai	Dapat diatur sesuai kebutuhan

### Deklarasi variabel pointer

Suatu model variabel pointer terdiri dari :

- pointer
- simpul
- field pointer pada simpul
- variabel pointer

*Pointer* merupakan pendefinisian type data pointer yang akan diacu oleh variabel yang akan dideklarasikan.

*Simpul*, satu kesatuan paket data, dimana setiap paket/rekord data akan terdiri dari bagian data dan bagian pointer. Simpul bersifat dinamik, yang diciptakan ketika diperlukan dan dapat segera dihancurkan begitu tidak diperlukan lagi.



## Variabel Biasa vs. Variabel Pointer

Variabel pointer adalah variabel yang menunjuk ke alamat memory yang digunakan untuk menampung data yang akan digunakan dalam proses.

	Alamat fisik	Peta memory
Lokasi memory paling akhir →	FFFFF H	
	00E2C H	100
	00E2B H	85
Lokasi memory paling awal →	00000 H	

Jika P adalah variabel pointer yang menunjuk ke lokasi 00E2C H, maka P secara teknis akan berisi address fisik tersebut dan digunakan sebagai acuan mengakses data 100. Setiap dilakukan aksi terhadap P maka akan diakses isi memory pada lokasi P tersebut (nilai P tidak berubah karena P adalah alamat).

Berbeda halnya dengan suatu variabel biasa A yang ditempatkan di memory, jika dilakukan aksi terhadap A, maka nilai A akan dimanipulasi sesuai aksi yang dilakukan terhadapnya.

## MODEL DESKRIPTIF LINKED LIST

### Pemodelan Linked List dengan Array

Misalkan diberikan sebuah array  $X [ i ]$  sebagai berikut :

*Model 1 (belum diurutkan):*

	i = 1	i =2	i =3	i =4	i =5	i =6	i =7	i =8
$X [ i ]$	21	3	54	17	121	11	32	51

Array  $X [ i ]$  tersebut ingin diurutkan dari data terkecil hingga data terbesar. Dalam model yang umum, akan dihasilkan sebagai berikut :

*Model 2 (diurutkan dengan cara memindahkan posisi)*

	i = 1	i =2	i =3	i =4	i =5	i =6	i =7	i =8
$X [ i ]$	3	11	17	21	32	51	54	121

Dimana hasil tersebut didapat setelah melalui proses pemindahan atau mengubah posisi data. Pengurutan data tersebut juga dapat dilakukan dengan cara berikut :

*Model 3 (diurutkan dengan melengkapi penunjuk)*

awal = 2

	i = 1	i =2	i =3	i =4	i =5	i =6	i =7	i =8
$X [ 1, i ]$	21	3	54	17	121	11	32	51
$X [ 2, i ]$	7	6	5	1	Nil	4	8	3

Pada model-3, array  $X [ i ]$  yang awalnya 1 dimensi diubah menjadi 2 dimensi dimana ini setiap elemen data akan diberi data indeks/posisi dari data berikutnya. Untuk posisi data yang pertama akan ditunjuk oleh sebuah variabel kepala.

Pada pengurutan dengan cara yang ke-dua ini tidak terjadi pemindahan data, dengan kata lain data tetap berada posisi semula sebelum terjadi pengurutan.

## Modul Struktur Data

Keterurutan data diidentifikasi oleh  $X[2, i]$  yang dalam hal ini kita sebut saja *penunjuk data berikutnya*.

Pembacaan data pada model 3 tersebut, adalah sebagai berikut :

- Awal : adalah variabel kepala ( $awal = 2$ ) , yang menunjukkan lokasi/posisi data terkecil pada posisi ke-2; yaitu  $X[1, 2] = 3$
- Selanjutnya  $X[1, 2]$  akan didampingi oleh  $X[2, 2]$  yang berisi posisi data terkecil berikutnya yaitu pada posisi 6; yaitu  $X[1, 6] = 11$ .
- Selanjutnya  $X[1, 6]$  akan didampingi oleh  $X[2, 6]$  yang berisi posisi data terkecil berikutnya yaitu pada posisi 4; yaitu  $X[1, 4] = 17$ .
- Selanjutnya  $X[1, 4]$  akan didampingi oleh  $X[2, 4]$  yang berisi posisi data terkecil berikutnya yaitu pada posisi 1, yaitu  $X[1, 1] = 21$ .
- Demikian seterusnya hingga  $i = 5$ , akan didapat  $X[1, 5]$  yang didampingi oleh informasi posisi *nil* (nihil) pada  $X[2, 5]$ , yang artinya  $X[1, 5]$  adalah data yang terbesar.

Algoritma untuk menuliskan data dari array  $X[b, i]$  dimana  $b = 1, 2$  dan  $i = 1, 2, \dots, 8$  sehingga dihasilkan data terurut adalah sebagai berikut :

```
Begin
  |
  |  $i \leftarrow awal$ 
  | while  $X[2, i] \neq nil$  do
  |   |
  |   | begin
  |   |   |
  |   |   |  $write(X[1, i])$ 
  |   |   |  $i \leftarrow X[2, i]$ 
  |   |   |
  |   |   | end-while
  |   | end-while
  | end-while
end-algoritma
```

Bandingkanlah dengan algoritma menuliskan array  $X[i]$  (*dari model-2*) berikut ini.

```
Begin
  |
  |  $i \leftarrow 1$ 
  | while  $i \leq n$  do
  |   |
  |   | begin
  |   |   |
  |   |   |  $write(X[i])$ 
  |   |   |  $i \leftarrow i + 1$ 
  |   |   |
  |   |   | end-while
  |   | end-while
  | end-while
end-algoritma
```

## Representasi Linked List

Perhatikan contoh data terurut yang direpresentasikan model-3 :

awal = 2

	i = 1	i = 2	i = 3	i = 4	i = 5	i = 6	i = 7	i = 8
X [1, i]	21	3	54	17	121	11	32	51
X [2, i]	7	6	5	1	Nil	4	8	3

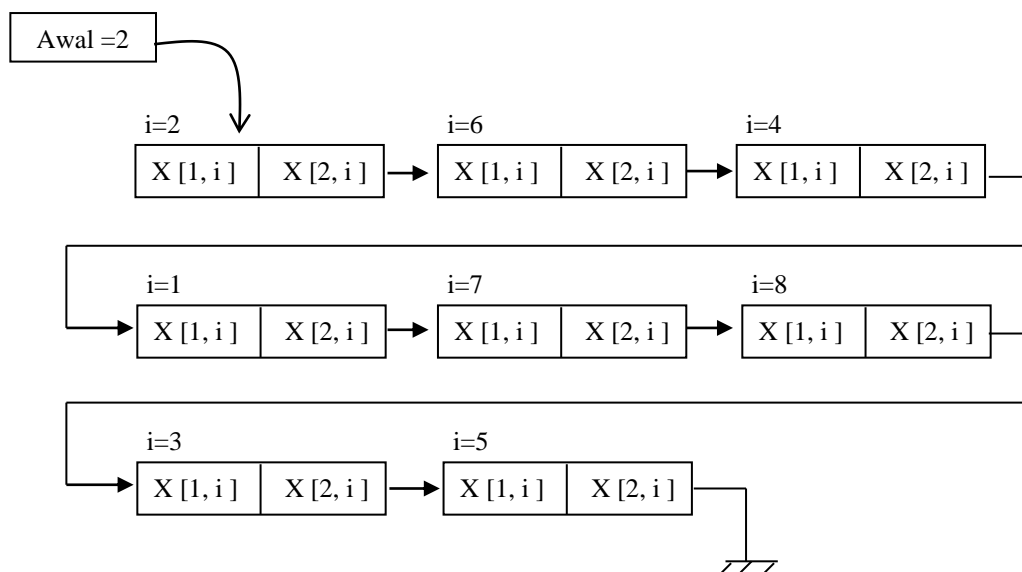
jika kita bayangkan satu set item data yang terdiri dari :

Item data : X[1, i]

Penunjuk elemen berikutnya : X[2, i]

indeks posisi/lokasi : i

sebagai sebuah simpul bebas, maka dapat digambarkan sebagai berikut :



Dalam bentuk Linked List yang sesungguhnya indeks posisi  $i$  adalah alamat fisik ruang memory yang ditempati oleh data.

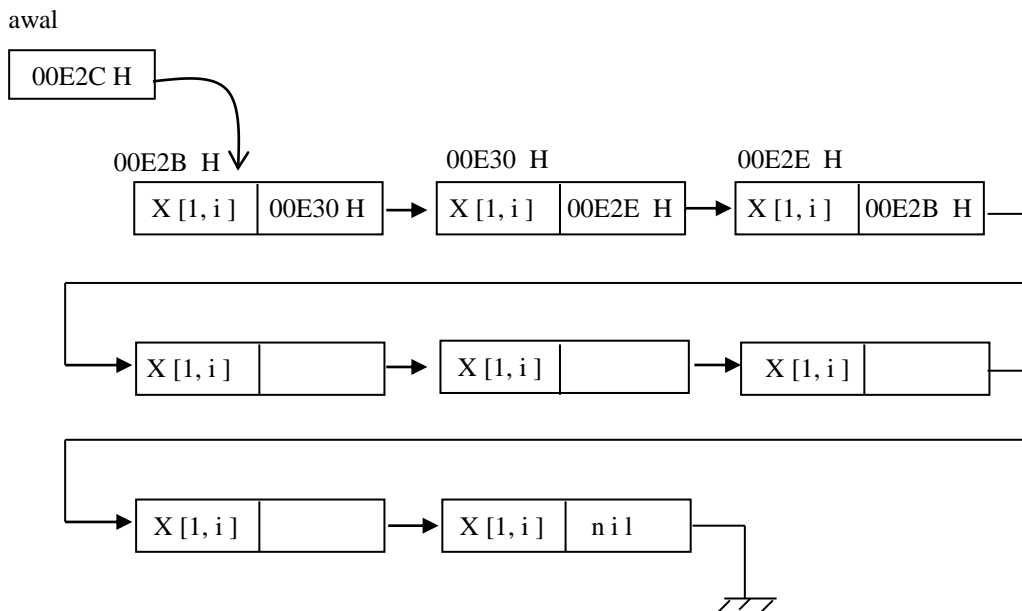
Dengan asumsi array tersebut dialokasikan secara kontigu yang dimulai pada address fisik memory 00E2B H, maka untuk Linked List yang sesungguhnya indeks posisi  $i$  adalah 00E2B H untuk  $i = 1$ , 00E2C H untuk  $i = 2$  dan seterusnya.

**Modul Struktur Data**

Perhatikan model pemetaan memory berikut ini :

Alamat fisik	Peta memory
FFFFF H	
00E32 H	51
00E31 H	32
00E30 H	11
00E2F H	121
00E2E H	17
00E2D H	54
00E2C H	3
00E2B H	21
00001 H	
00000 H	

Untuk penunjuk elemen berikutnya juga akan diisi oleh address fisik lokasi memory





## DEFINISI DAN ISTILAH

### Linked List

Linked List adalah struktur berupa rangkaian elemen saling terkait dimana tiap elemen dihubungkan dengan elemen lainnya oleh suatu pointer. Pointer adalah alamat fisik elemen. Linked dengan pointer menyebabkan data bertetangga secara logik walaupun tidak secara fisik (tidak kontigu). Dalam hal ini terdapat model kaitan Predesesor Suksesor. Syarat Linked List : harus diketahui alamat simpul pertama.

### Predesesor

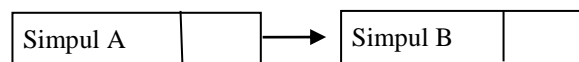
Predesesor adalah simpul yang menjadi pendahulu suatu simpul lainnya.

### Suksesor

Suksesor adalah simpul yang menjadi pengikut suatu simpul lainnya.

Simpul A adalah  
predesesor Simpul B

Simpul B adalah  
suksesor Simpul A



### Simpul

Simpul terdiri dari dua bagian , yaitu :

- bagian Data
- Bagian pointer yang menunjuk kesimpul berikutnya

### First / Head

Variabel First atau Head berisi alamat/pointer yang menunjuk lokasi simpul pertama Linked List, yang digunakan sebagai acuan awal penelusuran linked list

**Nil atau Null** → Tidak bernilai

Untuk menyatakan suatu variabel pointer tidak menunjuk kemanapun.

### Simpul Terakhir

Simpul terakhir linked list berarti simpul yang tidak memiliki suksesor. Field pointer pada simpul terakhir bernilai Nil

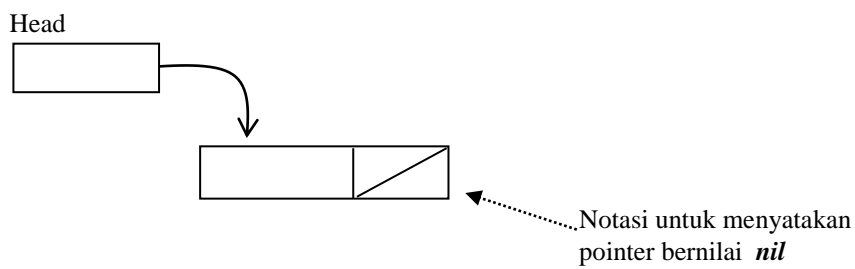
## MODEL SINGLY LINKED LIST

Singly Linked List adalah tipe linked list yang paling sederhana, dimana setiap simpul hanya memiliki satu link/pengkait atau pointer dengan simpul berikutnya.

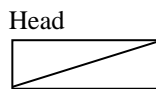
### Simpul List



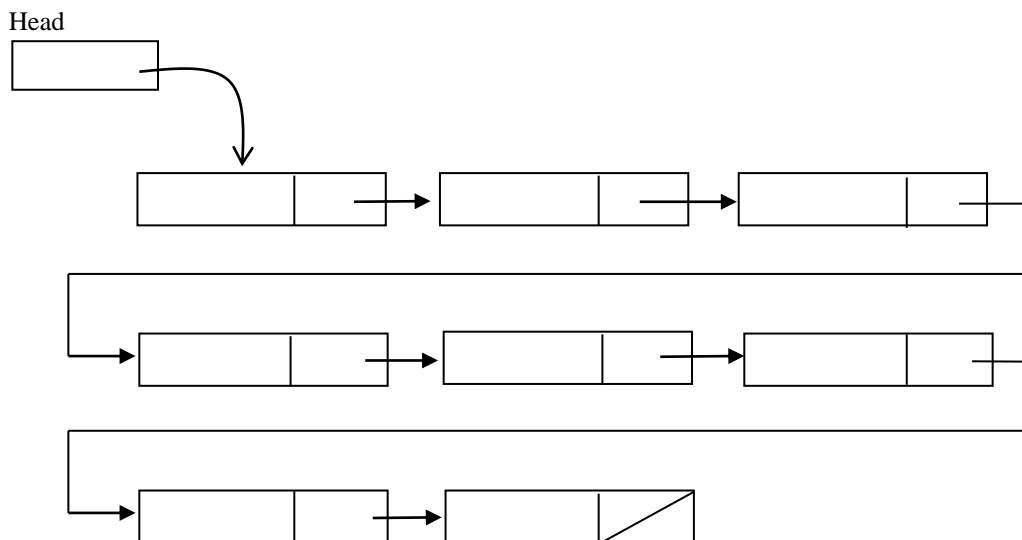
### Linked List dengan hanya satu simpul



### Linked List Kosong



### Linked List dengan serangkaian elemen.



## 1. Karakteristik Singly Linked List

Setiap simpul pada Singly Linked List hanya memiliki satu pointer, sehingga :

1. penelusuran hanya dapat dilakukan ke satu arah
2. penelusuran selalu dimulai dari simpul pertama
3. hanya dapat merepresentasikan secara baik data-data yang tersusun dalam satu lajur (satu dimensi).
4. tidak dapat merepresentasikan data yang tersusun dalam model matriks dimana setiap elemennya saling independen.

## 2. Deklarasi Singly Linked List

Type

```
< nama Pointer > = ^ < nama record >
```

```
< nama record > = record
```

```
    |
    |   < item1 >   : < type data 1 >
    |   < item2 >   : < type data 2 >
    |   . . .
    |   < item n >   : < type data n >
    |   < nama field pointer > : < nama Pointer >
    |
end;
```

var

```
< nama Variabel > : < nama Pointer >
```

Pada model deklarasi tersebut, item **data** boleh lebih dari satu field, sementara field linked (pointer) hanya satu.

### Contoh :

1. Diberikan sebuah himpunan bilangan bulat, deklarasi singly linked list yang sesuai untuk data tersebut :

Type

```
P = ^DataBil
```

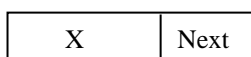
```
DataBil = record
```

```
    X      : integer ;
```

```
    Next   : P;
```

```
end;
```

Pada deklarasi setiap simpul memiliki 1(satu) field data yaitu X dan 1 (satu) field pointer yaitu Next.



2. Diberikan sebuah matriks 4 kolom dan  $n$  baris yang berisi data nilai ujian (UTS, UAS, Tugas, dan Nilai Akhir). Deklarasi Singly Linked List yang sesuai untuk data tersebut :

```

Type
daftar = ^ nilai
nilai = record
    Uts      : real;
    Uas      : real;
    Tugas    : real;
    N_akhir  : real;
    Next     : daftar;
end;
```

Pada deklarasi diatas, setiap simpul **nilai** memiliki 4 field data (nilai uts, uas, tugas, n\_akhir) dan satu field pointer.

Uts	Next
Uas	
Tugas	
N_akhir	

### 1.3 Model Deskriptif Singly Linked List

Perhatikan model pemetaan memory berikut ini :

Alamat fisik	Peta memory
FFFF H	
036A0 H	11
02132 H	51
02131 H	32
0102F H	121
00E2C H	3
00E2B H	21
00E2A H	54
00E01 H	17
00000 H	

## Modul Struktur Data

Dari pemetaan tersebut secara terurut akan didapatkan data sebagai berikut :

Alamat	00E2C H	036A0 H	00E01 H	00E2B H	02131 H	02133 H	00E2A H	0102F H
data	3	11	17	21	32	51	54	121

Jika data-data tersebut dikelola dengan suatu list berkait, maka :

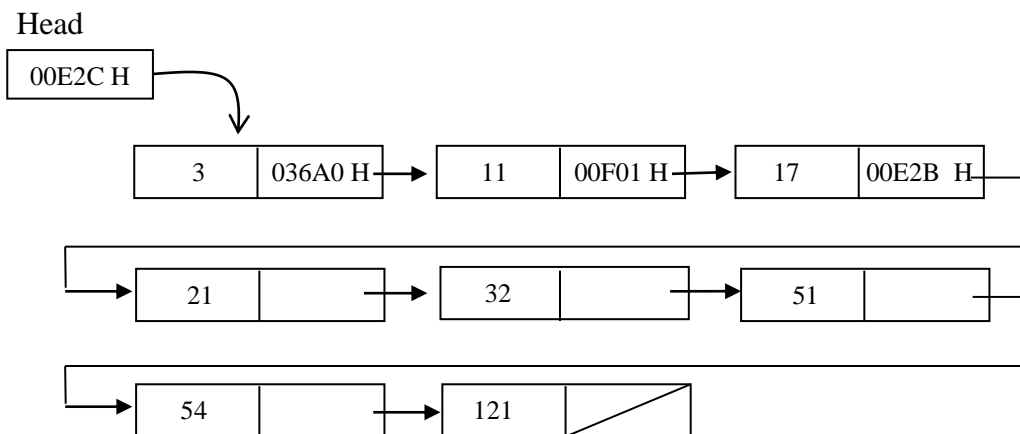
*Head* = 00E2C H;

simpul 1 : Data = 3, pointer = 036A0 H

simpul 2 : data = 11, pointer = 00F01 H

:

dan seterusnya.



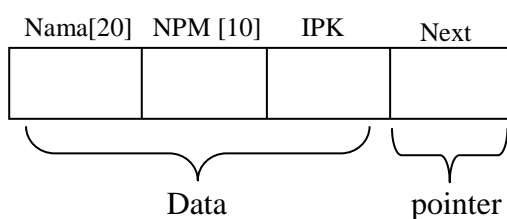
## Contoh lain Singly Linked List

Untuk mengelola suatu list mahasiswa dapat dideklarasikan sebagai berikut :

```

Type
  PointerMhs = ^TabelMhs
  Mhs = record
    Nama : string [ 20 ]
    NPM  : string [ 10 ]
    IPK  : real
    Next : PointerMhs
  End;
Var
  DataMhs : PointerMhs
  
```

Simpul yang direpresentasikan oleh deklarasi tersebut adalah sebagai berikut :



## OPERASI DASAR PADA LINKED LIST

Operasi paling mendasar yang dilakukan pada linked list adalah :

1. Penciptaan simpul
2. Penghancuran simpul
3. Inisialisasi Linked List
4. Pemeriksaan Linked List kosong

### 1. PENCIPTAAN SIMPUL

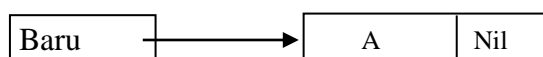
Penciptaan simpul berarti melakukan alokasi memori sesuai dengan kebutuhan sebuah simpul dan mengidentifikasi alamat fisik dari memory yang dialokasikan. Yang perlu diingat bahwa penciptaan simpul dilakukan ketika sudah ada data yang akan disimpan pada simpul tersebut.

Misalnya ingin diciptakan sebuah simpul sesuai dengan deklarasi berikut :

```
Type
P = ^DataBil
DataBil = record
    X      : integer ;
    Next  : P;
end;
```

Algoritma penciptaan simpul dalam format Pascal Like adalah sebagai berikut :

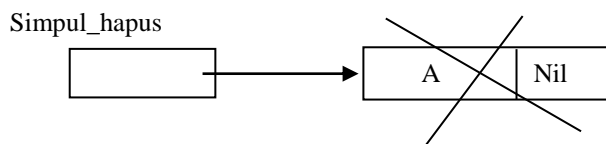
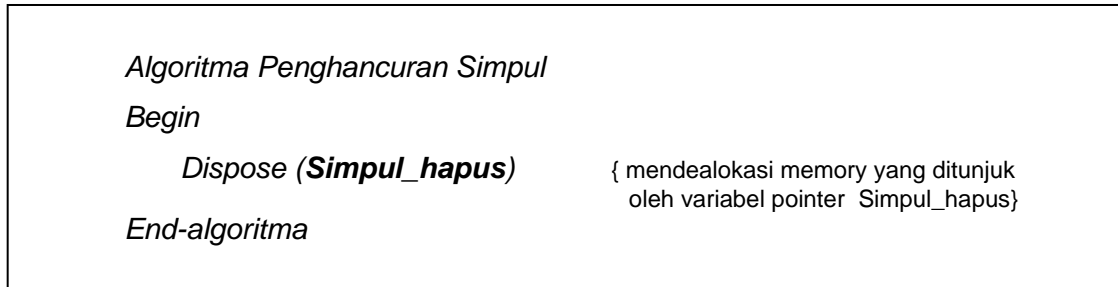
```
Algoritma Penciptaan Simpul
Var baru : P
Begin
    Read (A)           { baca data dari papan kunci simpan di var. A}
    New(Baru )         { ciptakan simpul, & identifikasi pointer baru}
    Baru^ . X ← A      { berdasarkan pointer Baru, simpan data A ke field X}
    Baru^ . Next ← nil { inisialisasi field Next dengan nil }
End-algoritma
```



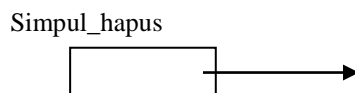
Yang harus diingat pada saat instruksi penciptaan simpul *New (Baru)* dieksekusi, akan dialokasikan ruang memory sesuai ukuran simpul dan diidentifikasi alamat fisik memory yang kemudian (alamat tersebut) disimpan dalam variabel pointer **Baru** .

## 2. Penghancuran Simpul

Penghancuran simpul berarti melakukan dealokasi ruang memory yang sudah tidak digunakan lagi sebagai akibat penghapusan simpul (instruksi penghapusan tidak serta merta mendealokasi ruang memory yang ditempat simpul tersebut :



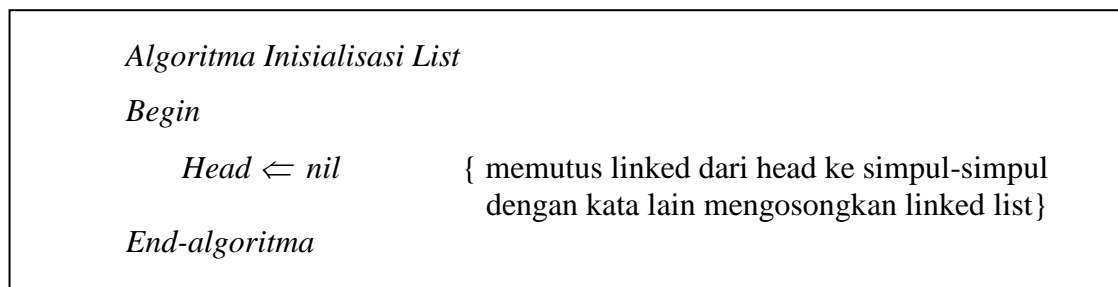
Sehingga hasilnya menjadi



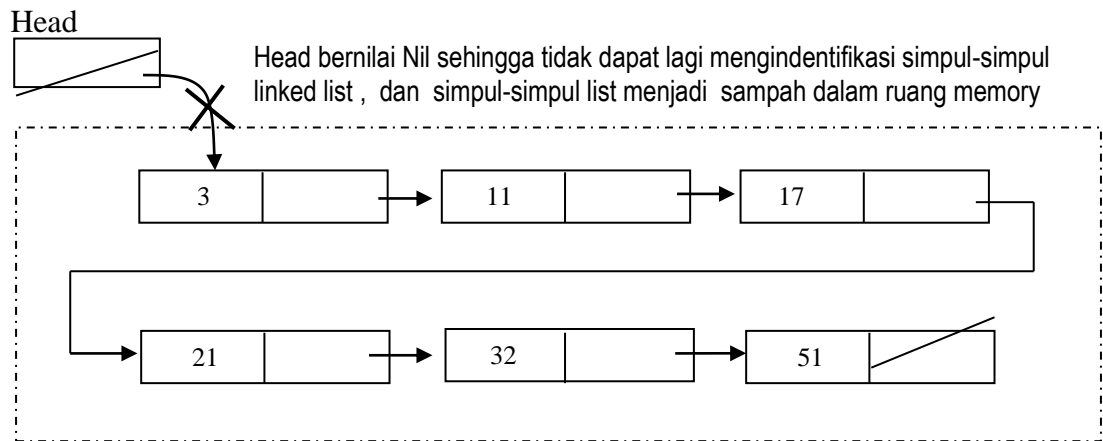
## 3. Inisialisasi Linked List Kosong

Inisialisasi Linked List berarti mengosongkan Linked List. Karena pengidentifikasian simpul-simpul linked list berdasarkan pointer yang tersimpan dalam field pointer pada simpul yang mendahuluinya (predecessor).

Jika dilakukan pemutusan suatu *linked* mengakibatkan seluruh simpul yang berada dibelakangnya tidak dikenali lagi. Jika pemutusan linked dilakukan di pointer yang menunjuk simpul pertama (memutus linked dari *Head*), maka seluruh simpul tidak akan dikenali, dengan kata lain linked list menjadi kosong.



Dalam kasus ini memory yang sebelumnya ditempati oleh simpul-simpul linked list tetap dikuasai oleh simpul tersebut karena tidak diikuti perintah penghancuran.



Harus diperhatikan, bahwa simpul-simpul yang tidak lagi dapat diidentifikasi, akan mengakibatkan penguasaan memory oleh data yang tdaik berguna

#### 4. Pemeriksaan Linked List Kosong

Dengan memperhatikan sifat variabel *pointer* Head pada contoh diatas, untuk mengetahui apakah sebuah linked list kosong cukup dengan memeriksa apakah **Head bernilai Nil** atau tidak.

*Fungsi List\_Kosong: boolean*

*Begin*

*If Head  $\Leftarrow$  nil*

*then*

*List\_kosong  $\Leftarrow$  true*

*End-if*

*End-algoritma*

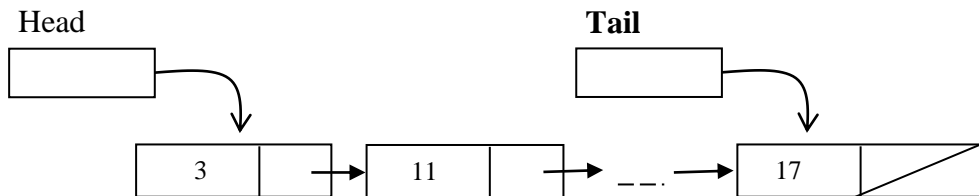


## VARIAN SINGLY LINKED LIST DAN DOUBLE LINKED LIST

1. Singly Linked List dengan pointer Head dan Tail
2. Singly Linked List Sirkular
3. Double Linked List

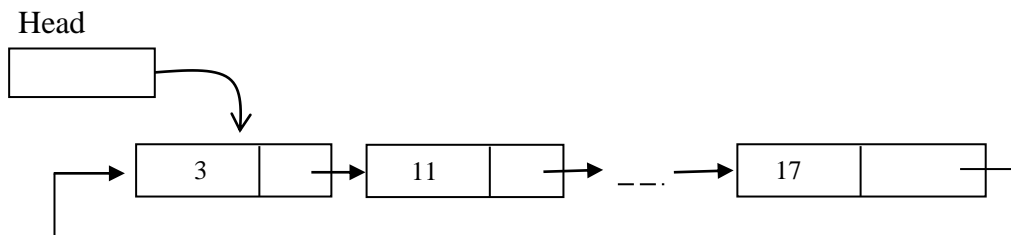
### 1. Singly Linked List dengan pointer Head dan Tail

Untuk kebutuhan kemudahan operasi terhadap linked list, Singly Linked List dapat dilengkapi dengan dua pointer yaitu Head dan Tail. Dengan adanya pointer Tail, maka operasi penambahan data diakhir list tidak memerlukan penelusuran keseluruhan linked list. Sehingga keberadaan pointer Tail akan menghemat kebutuhan waktu proses.



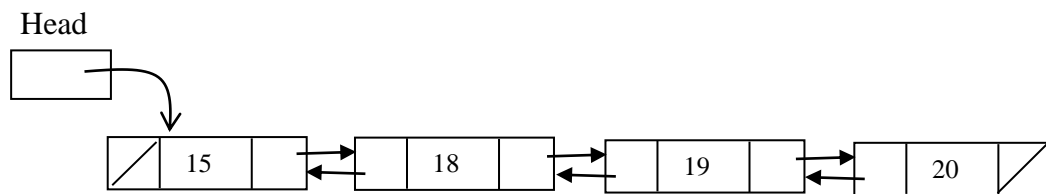
### 2. Singly Linked List Sirkular

Umumnya pointer pada simpul terakhir list bernilai **nil**. Untuk kebutuhan tertentu yang berkaitan dengan operasi/pengolahan data secara sirkular, maka pointer pada simpul terakhir akan menunjuk ke simpul I.



### 3. Double Linked List

Agar penelusuran linked list dapat dilakukan secara maju dan mundur, maka simpul linked list dapat dilengkapi dengan pointer Prev. Model ini tidak lagi disebut sebagai Singly Linked List, tapi dikenal sebagai Double Linked List.



## MEMBANGUN SINGLY LINKED LIST

Tahapan pembangunan list :

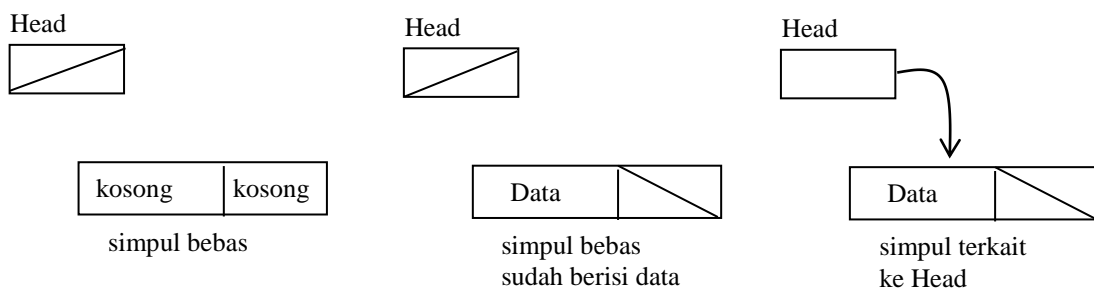
- Menciptakan simpul
- Mengisikan data ke simpul
- mengkaitkan dengan simpul sebelumnya (predesesor-nya)

### 1. Menciptakan simpul yang pertama :

Sifat khusus penciptaan simpul yang pertama adalah : linked ditujukan ke variabel pointer kepala (head).

Langkah-langkah :

- Ciptakan sebuah simpul bebas
- Isikan data
- Kaitkan simpul ke Head



Jika dideklarasikan simpul sebagai berikut :

```
Type
  Ptr = ^Simpul
  Simpul = record
    Data : Integer
    Next : Ptr
  End;

Var
  Head, Kaitan : Ptr
```

Untuk menciptakan simpul yang pertama dapat dilakukan dengan algoritma berikut (Pascal Like).

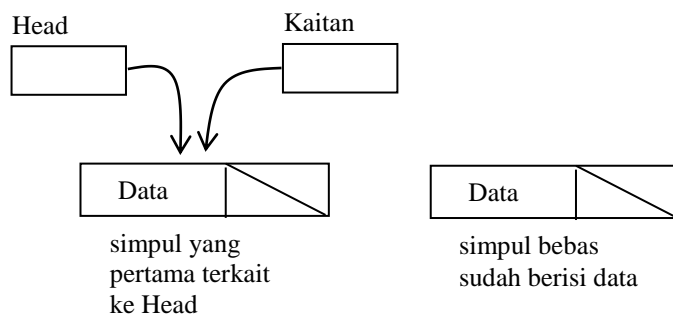
```
Algoritma Create_Simpul_Pertama
var A : integer
begin
    Read (A)
    New ( Ptr )           { menciptakan simpul bebas }
    Ptr^.Data := A       { mengisi data }
    Ptr^.Next := nil     { Inisialisasi field pointer dengan nil }
    Head := Ptr          { mengkaitkan simpul bebas dengan Head }
end-algoritma
```

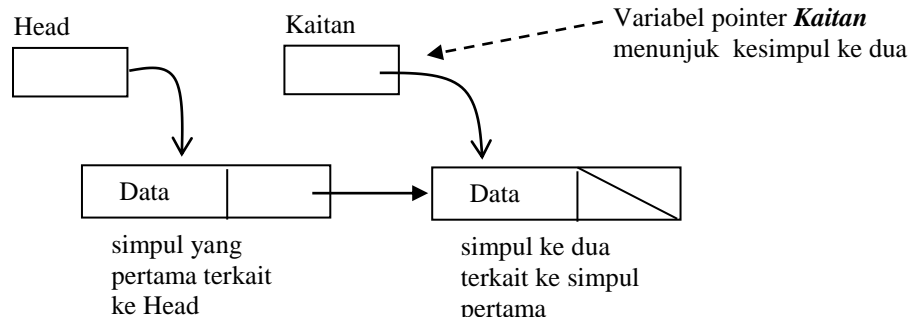
**2. Menciptakan simpul yang ke dua :**

Sifat khusus penciptaan simpul yang ke dua dan seterusnya adalah : linked ditujukan ke field pointer pada simpul sebelumnya. Untuk mendeteksi simpul sebelumnya diperlukan sebuah variabel pointer khusus (misalkan variabel tersebut diberi nama **Kaitan**), yang isinya selalu diperbaharui sehingga menunjuk ke simpul yang terakhir. Dalam kasus ini untuk mengkaitkan simpul yang ke dua masih dapat menggunakan **Head**. Tapi pada simpul ke tiga dan seterusnya tidak dapat lagi karena **Head** selalu menunjuk variabel yang pertama.

Langkah-langkah :

1. Ciptakan sebuah simpul bebas
2. Isikan data
3. Kaitkan simpul ke simpul sebelumnya





Untuk menciptakan simpul yang kedua dan seterusnya dapat dilakukan dengan algoritma berikut (Pasca Like).

```
Algoritma Create_Simpul_berikut
var A : integer
begin
    Read (A)
    New ( Ptr )           { menciptakan simpul bebas }
    Ptr^.Data := A       { mengisikan data }
    Ptr^.Next := nil     { Inialisasi field pointer dengan nil }
    Kaitan^.next := Ptr  { mengkaitkan simpul bebas dengan
                          predessornya /simpul pendahulunya}
    Kaitan := Ptr        { memperbaharui isi variabel pointer Kaitan
                          dengan alamat simpul yang terakhir }
end-algoritma
```

Karena dalam penciptaan simpul ke dua dan seterusnya tersebut dilakukan setelah penciptaan simpul yang pertama maka diperlukan penyesuaian pada algoritma penciptaan simpul yang pertama, yaitu mengisikan alamat simpul pertama ke variabel pointer **Kaitan** untuk digunakan melinked simpul baru dengan simpul yang pertama, sebagai berikut : menambahkan instruksi **Kaitan := Ptr** setelah instruksi **Head := Ptr**

Lengkapnya algoritma membangun linked list adalah sebagai berikut :

```

Algoritma Create_Linked_List
var A : integer; ada : boolean
begin
    Read (A)           { membaca data ke-1 }
    New ( Ptr )        { menciptakan simpul bebas yang ke 1 }
    Ptr^.Data := A     { mengisi data }
    Ptr^.Next := nil   { Inisialisasi field pointer dengan nil }

    Head := Ptr       { mengkaitkan simpul bebas dengan head }
    Kaitan := Ptr     { menginisialisasi Kaitan dengan alamat simpul pertama }

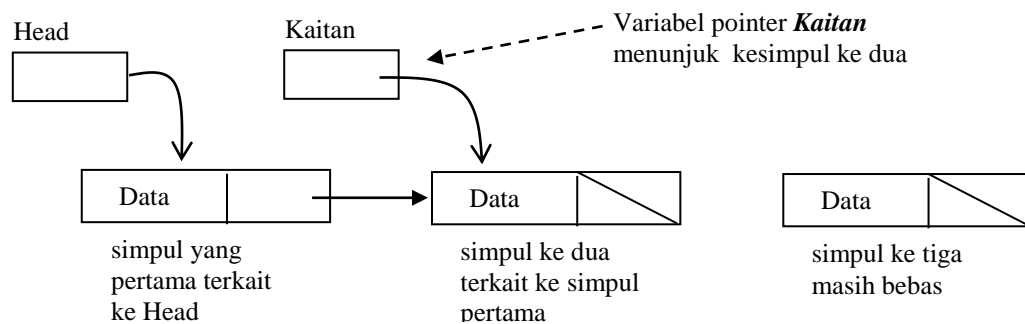
    Periksa_data(ada) { memanggil prosedur memeriksa apakah masih ada data }

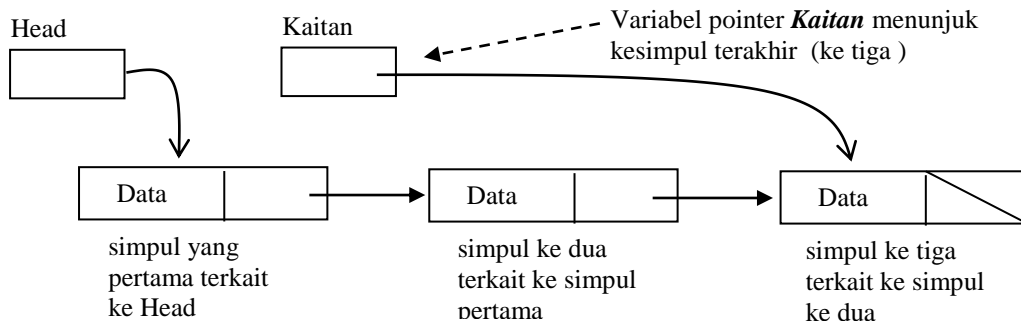
    While ada do     { looping dilakukan selama variabel boolean bernilai TRUE }
    begin
        Read (A)
        New ( Ptr )   { menciptakan simpul bebas berikutnya }
        Ptr^.Data := A { mengisi data }
        Ptr^.Next := nil { Inisialisasi field pointer dengan nil }

        Kaitan^.next := Ptr { mengkaitkan simpul bebas dengan predessornya /simpul pendahulunya}
        Kaitan := Ptr     { memperbaharui isi variabel pointer Kaitan dengan alamat simpul yang terakhir }
        Periksa_data(ada) { memanggil prosedur memeriksa apakah masih ada data }
    end-while
end-algoritma
    
```

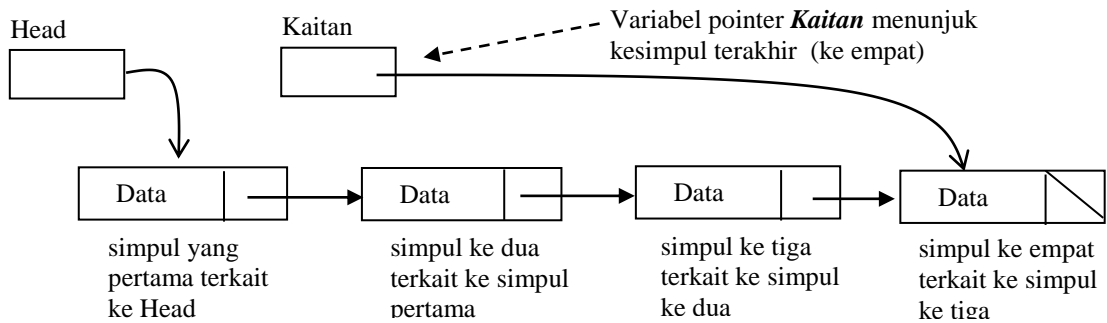
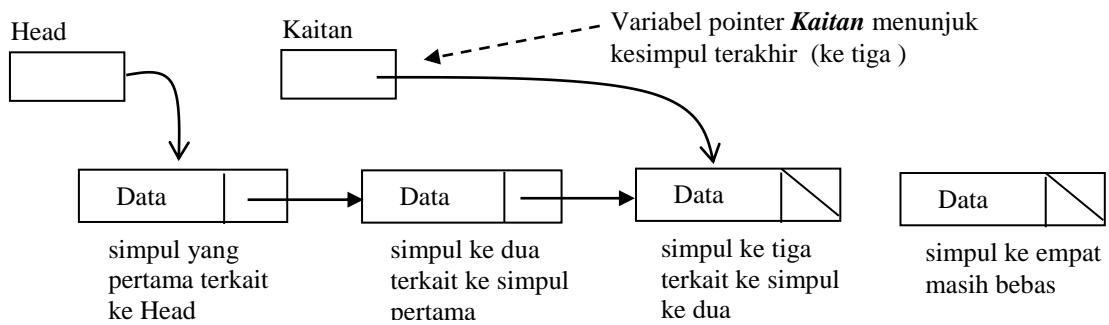
Penciptaan dan linked simpul ke-3 dst dapat diilustrasikan sebagai berikut :

**Simpul ke-3**





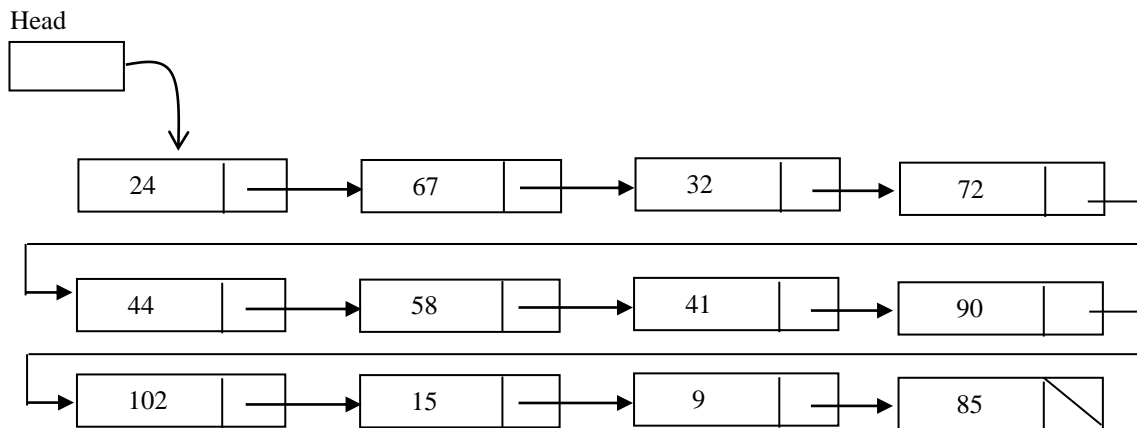
### Simpul ke-4



## PENELUSURAN LINKED LIST

Penelusuran linked list ialah mengunjungi seluruh elemen linked list dari simpul yang pertama sampai dengan simpul terakhir. Terminasi penelusuran adalah jika ditemukan simpul dengan field pointer bernilai **Nil**.

Misalnya diberikan sebuah Linked List sebagai berikut :



Algoritma umum untuk menelusuri lis tersebut adalah sebagai berikut (struktur simpul yang digunakan sesuai deklarasi dihalaman 2) :

```

Algoritma Penelusuran_List
  var k : ptr           { var. pointer untuk penelusuran }
  Begin
    k := Head           { menginisialisasi k dengan alamat
                        simpul yang pertama }
    While k <> nil do   { looping akan dilakukan selama pointer
    begin               tidak nil }
      Visit ( k )     { aksi yang didefinisikan terhadap simpul
                        yang dikunjungi }
      k := k^.next    { memperbaharui variabel penelusuran
                        dengan alamat simpul berikutnya }
    end-while
  end-algoritma
  
```



## **Modul Struktur Data**

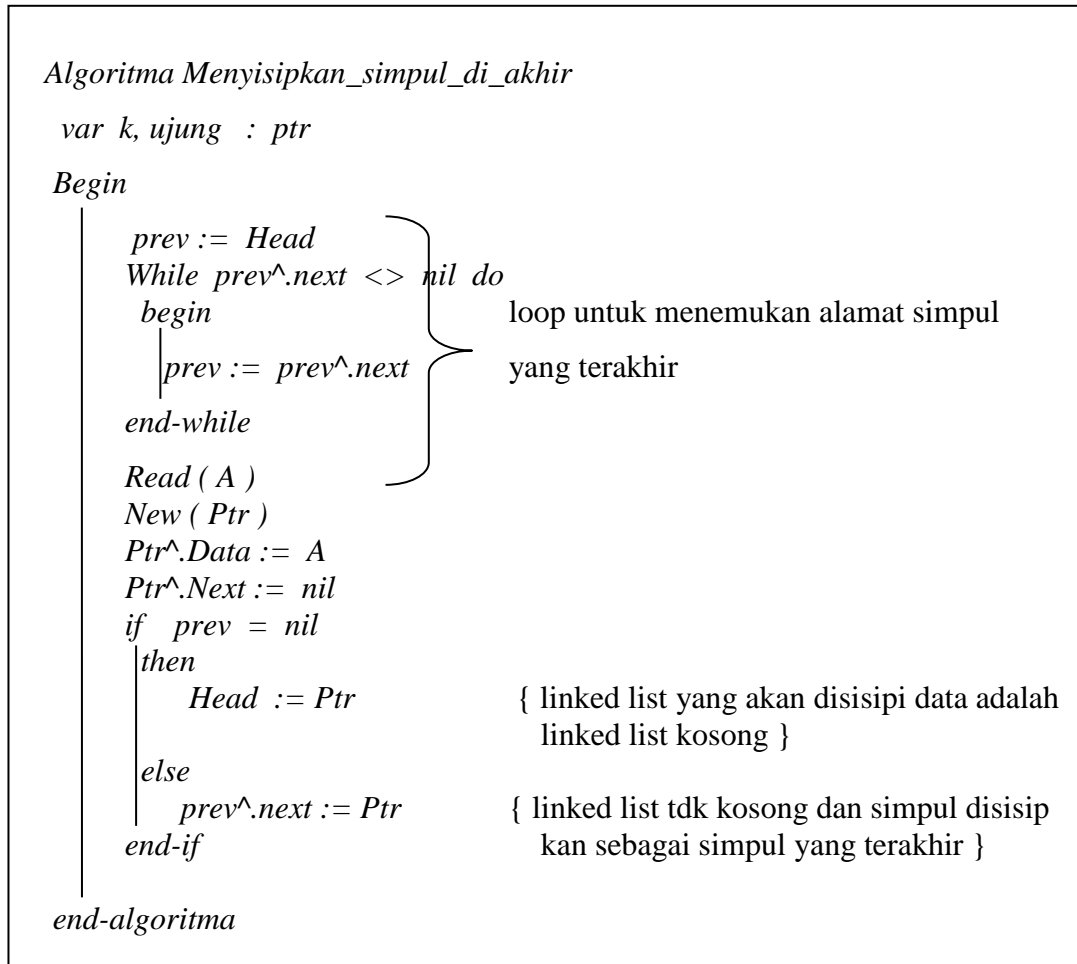
Misalnya penelusuran dimaksudkan untuk mencari data terbesar/maksimum dari semua data yang diorganisasikan oleh linked list ke layar :

```
Algoritma Penelusuran_List
var k : ptr
Begin
  k := Head
  maks := -9999
  While k <> nil do
    begin
      if maks < k^.Data
      | then maks := k^.Data
      end-if
      k := k^.next
    end-while
end-algoritma
```

## MENYISIPKAN DAN MENGHAPUS SIMPUL

Untuk menyisipkan dan menghapus sebuah simpul langkah pertama yang harus dilakukan adalah menemukan lokasi peyisipan atau data yang akan dihapus.

Menyisipkan simpul baru menjadi simpul terakhir :



### Menyisipkan simpul di awal

*Algoritma Menyisipkan\_simpul\_di\_awal*

*var k : ptr*

*Begin*

*Read ( A )*

*New ( Ptr )*

*Ptr^.Data := A*

*Ptr^.Next := Head*

*Head := Ptr*

*end-algoritma*

### Menyisipkan simpul di lokasi setelah data X

*Algoritma Menyisipkan\_simpul*

*var k, lokasi : ptr*

*Begin*

*k := Head*

*prev := Head*

*While (k^.Data <> X) and k <> nil do*

*begin*

*prev := k*

*k := k^.next*

*end-while*

*Read ( A )*

*New ( Ptr )*

*Ptr^.Data := A*

*if prev = nil*

*{ linked list kosong }*

*then*

*Ptr^.Next := nil*

*Head := ptr*

*else*

*If k = nil*

*then*

*Ptr^.next := nil*

*Prev^.next := ptr*

*{ menyisipkan di ujung }*

*Else*

*Ptr^.Next := prev^.next*

*prev^.next := Ptr*

*{ menyisipkan di tengah }*

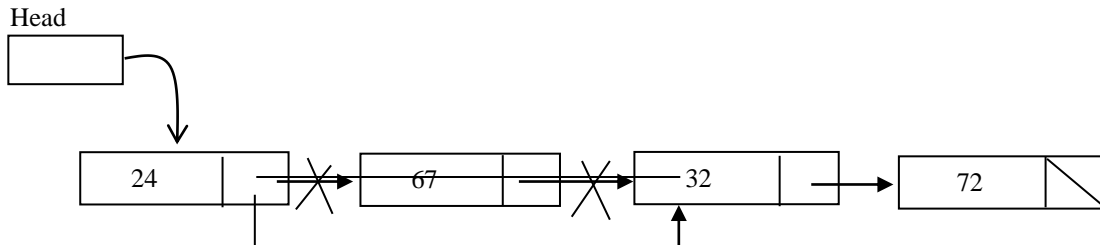
*end-if*

*end-if*

*end-algoritma*

## Menghapus Simpul

Pada penghapusan simpul, isi field pointer simpul predesesor-nya diisi dengan alamat simpul suksesor-nya



Misalnya akan dihapus simpul yang bernilai 67 maka pointer dari simpul bernilai 24 (predesesor dari simpul 67) diganti dengan alamat simpul bernilai 32 (suksesor dari simpul 67). Alamat simpul bernilai 32 tersimpan di field pointer pada simpul bernilai 67.

### *Algoritma Menghapus\_simpul*

*var k, prev : ptr*

*Begin*

*k := Head*

*Prev := Head*

*While (k^.Data <> X) and (k <> nil) do*

*begin* { X adalah data yang akan dihapus }

*prev := k*

*k := k^.next*

*end-while*

*if k^.Data = X* { Jika k = nil berarti tidak ada data tsb }

*then*

*if k = head* { jika simpul dihapus adalah simpul I }

*then*

*head := head^.next*

*else*

*prev^.next := k^.next*

*end-if*

*end-if*

*dispose(k)* { memusnahkan/dealokasi simpul dihapus dari memory }

*end-algoritma*