MODUL VIII
Struktur Data

Judul	OPERASI DASAR PADA LINKED LIST	
Penyusun	**Distribusi**	**Perkuliahan Online**
Nixon Erzed	Teknik Informatika Universitas Esa Unggul	Online – VIII

Tujuan :

Setelah mengikuti kuliah ini, mahasiswa dapat mengenal variabel pointer sebagai pembentuk struktur Linked List, memahami representasi data dengan Singly Linked List dan mengenal operasi-operasi pada Linked List

Materi :

1. Operasi Dasar
   a. Penciptaan & Penghancuran simpul
   b. Inisialisasi linked list
   c. Pemeriksaan list kosong
2. Operasi Lanjut pada Linked List
   - Membangun Linked List
   - Penelusuran Linked List
   - Penyisipan simpul
     - sebagai simpul pertama
     - setelah simpul tertentu
     - sebagai simpul terakhir
   - Menghapus simpul

OPERASI DASAR PADA LINKED LIST

1. PENCIPTAAN SIMPUL

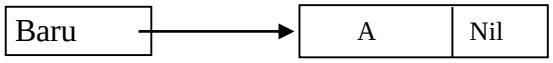
Penciptaan simpul berarti melakukan alokasi memori sesuai dengan kebutuhan sebuah simpul dan mengidentifikasi alamat fisik dari memory yang dialokasikan. Yang perlu diingat bahwa penciptaan simpul dilakukan ketika sudah ada data yang akan disimpan pada simpul tersebut.

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 sebegai 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
```

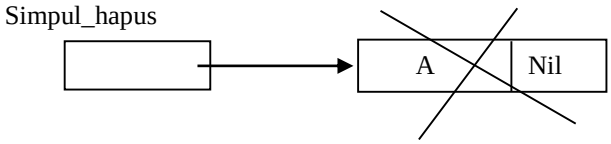


Yang harus diingat pada saat instruksi penciptaan simpul *New (Baru)* dieksekusi, akan dialokasikan ruang memory sesuai ukuran simpul dan diindetifikasi alamat fisik memory yang kemudian (alamat tersebut) disimpan dalam variabel pointer **Baru** .

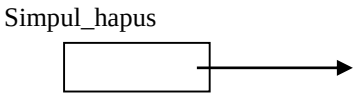
2. Penghancuran Simpul

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 :

```
Algoritma Penghancuran Simpul
Begin
    Dispose (Simpul_hapus)        { mendealokasi memory yang ditunjuk
                                    oleh variabel pointer  Simpul_hapus}
End-algoritma
```



Sehingga hasilnya menjadi



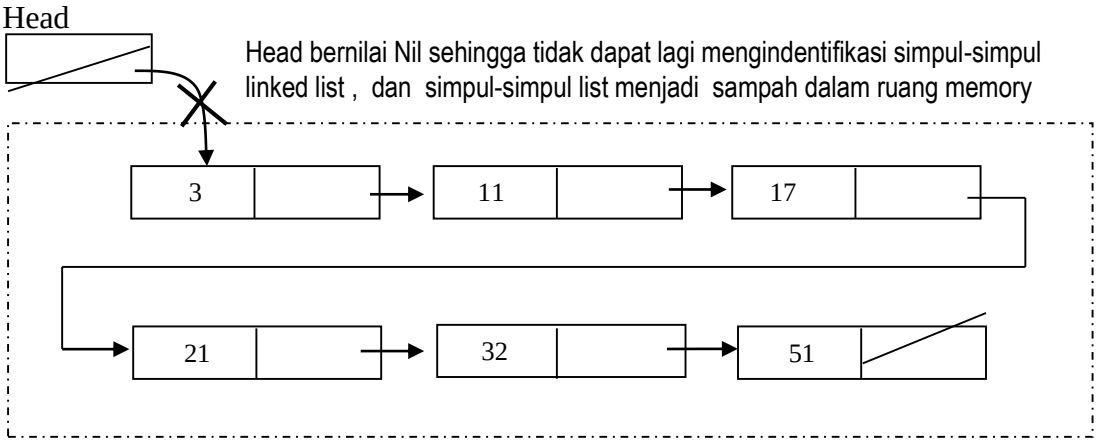
3. Inisialisasi Linked List Kosong

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

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

```
Algoritma Inisialisasi List
Begin
    Head ⇐ nil          { memutus linked dari head ke simpul-simpul
                          dengan kata lain mengosongkan linked list}
End-algoritma
```

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 diindetifikasi, akan mengakibatkan penguasaan memory oleh data yang tdaik berguna

4. Pemeriksaan Linked List Kosong

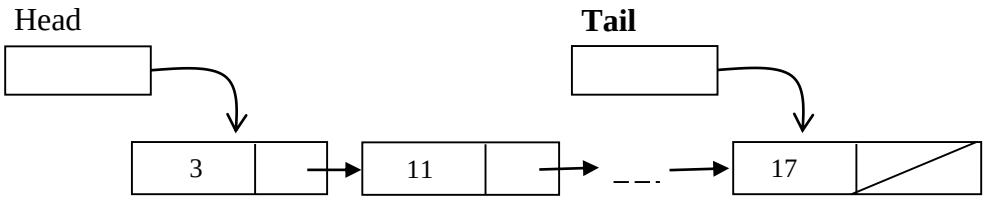
Dengan memperhatikan sifat variabel *pointer* Head pada contoh diatas, untuk mengetahui apakah sebuah linked list kosong cukup dengan memeriksa apakah **Head bernilai *Nil* atau tidak**.

```
Fungsi  List_Kosong: boolean
Begin
    If  Head ⇐ nil
     then
          List_kosong ⇐ true
   End-if
End-algoritma
```

VARIAN SINGLY LINKED LIST DAN DOUBLE LINKED LIST

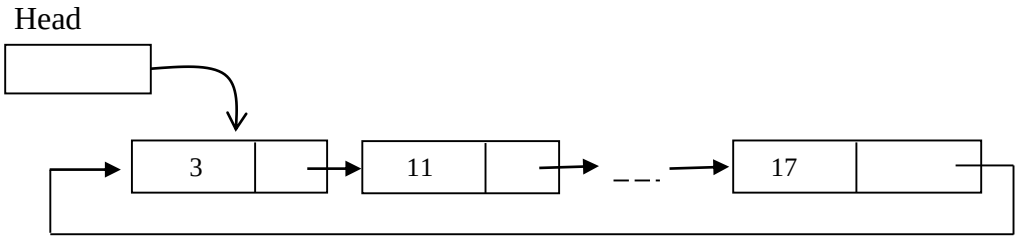
1. Singly Linked List dengan pointer Head dan Tail

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.



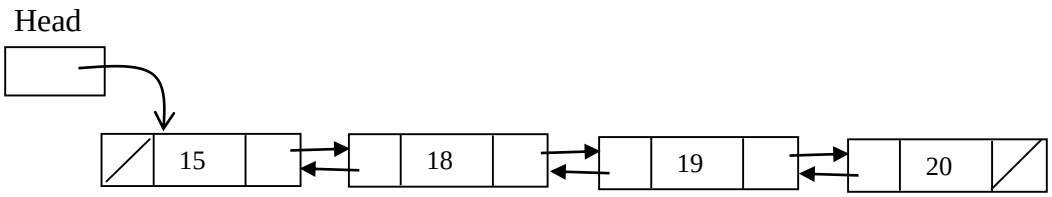
2. Singly Linked List Sirkular

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 penulusuran 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 Lingked List.



MEMBANGUN SINGLY LINKED LIST

Tahapan pembangunan list :

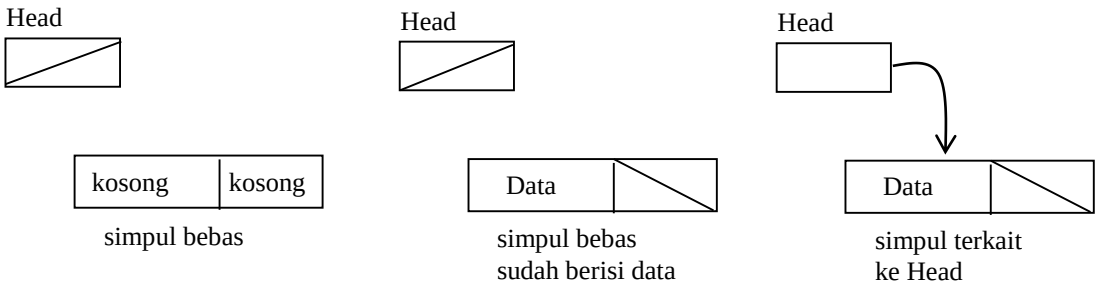
a. Menciptakan simpul
b. Mengisikan data ke simpul
c. mengkaitkan dengan simpul sebelumnya (predesesor-nya)

1. Menciptakan simpul yang pertama :

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

Langkah-langkah :

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



Jika dideklarasikan simpul sebagai berikut :

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

 Var
   Head, Kaitan  :  Ptr
```

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

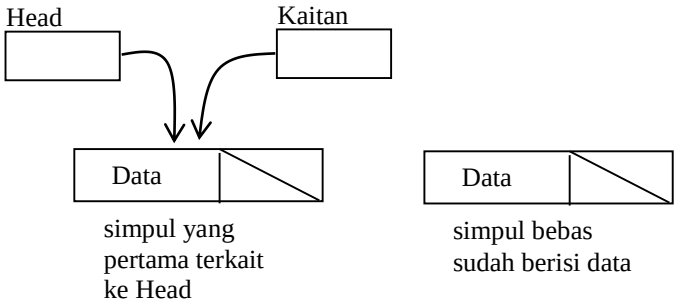
```
Algoritma Create_Simpul_Pertama
var   A :  integer
begin
        Read (A)
        New ( Ptr )                 { menciptakan simpul bebas }
        Ptr^.Data  :=  A            { mengisikan data }
        Ptr^.Next  :=  nil          { Inisialisasi field pointer dengan nil }
        Head  :=  Ptr  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

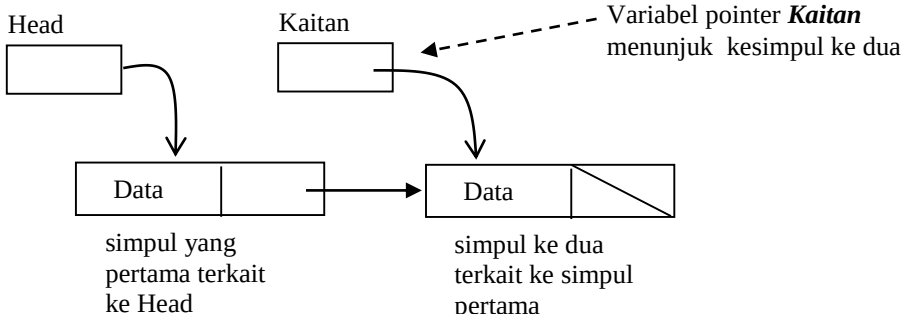
2. Menciptakan simpul yang ke dua :

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 varibel 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





Penciptaan simpul yang kedua dan seterusnya dapat dilakukan dengan algoritma berikut:

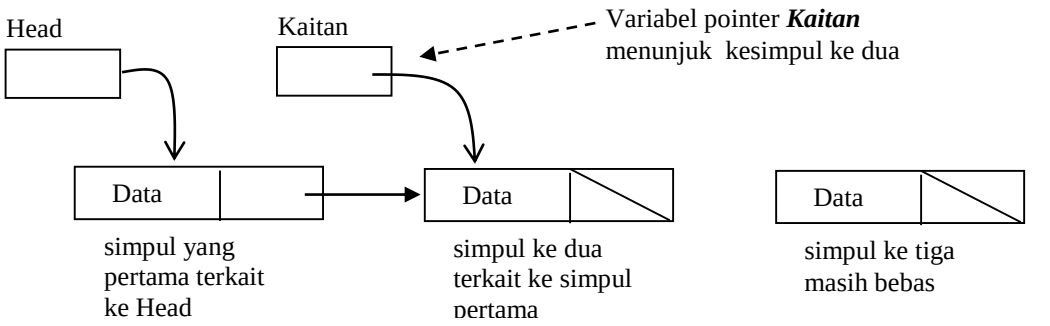
```
Algoritma Create_Simpul_berikut
var   A :  integer
begin
        Read (A)
        New ( Ptr )                 { menciptakan simpul bebas }
        Ptr^.Data  :=  A            { mengisikan data }
        Ptr^.Next  :=  nil          { Inisialisasi field pointer dengan nil }
        Kaitan^.next  :=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

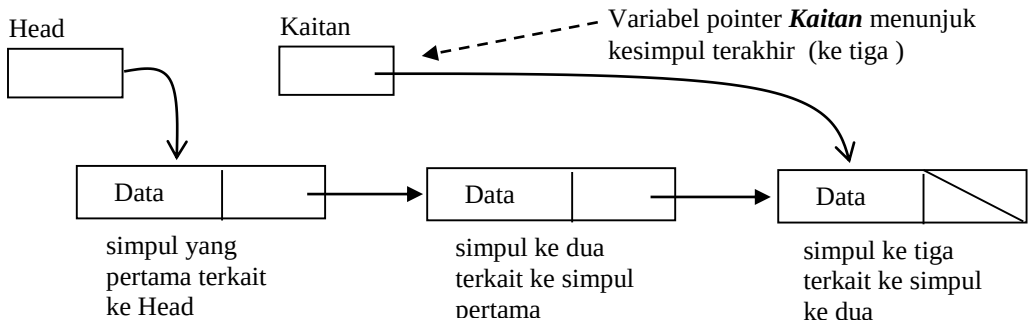
Karena dalam penciptaan simpul ke dua dan seterusnya tersebut dilakukan setelah penciptaan simpul yang pertama maka diperlukan penyesuaian pada algoritma penciptaan simpul yang pertama, yaitu mengisikan alamat simpul pertama ke variabel pointer ***Kaitan*** untuk digunakan me*linked* simpul baru dengan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

```
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            { mengisikan 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
        begin                         bernilai  TRUE  }
            Read (A)
            New ( Ptr )             { menciptakan simpul bebas berikutnya }
            Ptr^.Data  :=  A        { mengisikan 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 alamt simpul yang terakhir
            Periksa_data(ada)       { memanggil prosedur memeriksa apakah
                                      masih ada data }
        end-while
end-algoritma
```

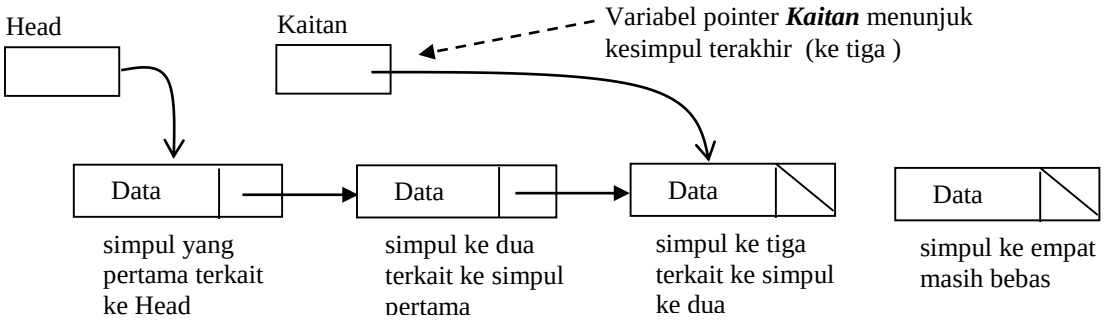
Penciptaan dan linked simpul ke-3 dst dapat diilustrasikan sebagai berikut :

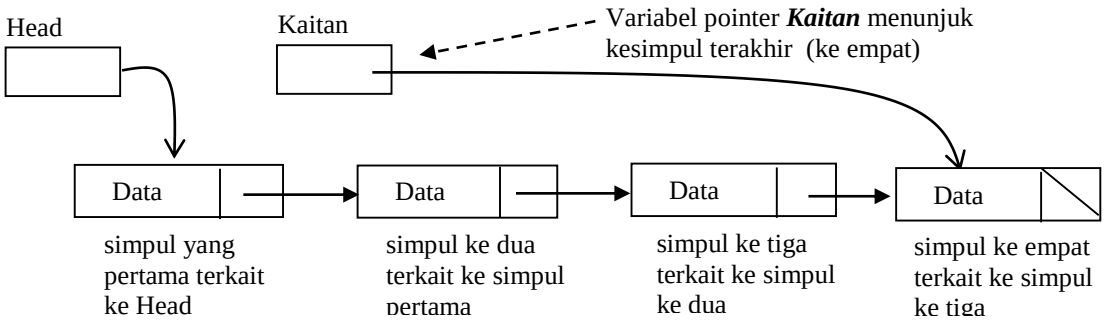
Simpul ke-3





Simpul ke-4

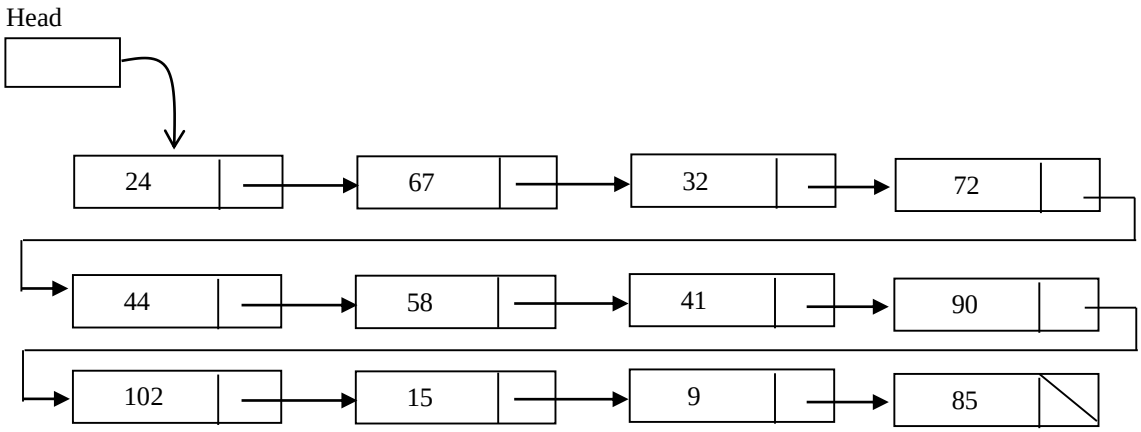




PENELUSURAN LINKED LIST

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
```

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 menghapuskan sebuah simpul langkah pertama yang harus dilakukan adalah menemukan lokasi peyisipan atau data yang akan dihapus.
Menyisipkan simpul baru menjadi simpul terakhir :

```
Algoritma Menyisipkan_simpul_di_akhir
 var  k, ujung   :  ptr
Begin
    prev :=  Head
    While  prev^.next  <>  nil  do
     begin                                  loop untuk menemukan alamat simpul
       prev :=  prev^.next                  yang terakhir
    end-while

    Read ( A )
    New ( Ptr )
    Ptr^.Data :=  A
    Ptr^.Next :=  nil
    if   prev  =  nil
    then
         Head  := Ptr              { linked list yang akan disisipi data adalah
                                     linked list kosong }
    else
         prev^.next := Ptr         { linked list tdk kosong dan simpul disisip
                                     kan sebagai simpul yang terakhir }
    end-if

end-algoritma
```

Menyisipkan simpul di awal

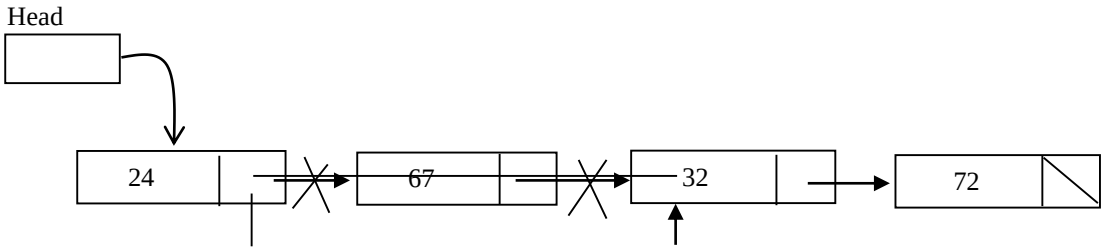
```
Algoritma Menyisipkan_simpul_di_awal
 var  k :  ptr
Begin
      Read ( A )
      New  ( Ptr )
      Ptr^.Data :=  A
      Ptr^.Next := Head
      Head  := Ptr
end-algoritma
```

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                                 { menyisipkan di ujung }
              Ptr^.next := nil
              Prev^.next  :=  ptr
           Else                                 { menyisipkan di tengah }
               Ptr^.Next := prev^.next
               prev^.next := Ptr
          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
```