PENDAHULUAN

- Dikembangkan tahun 1955-1956 oleh Allen Newell, Cliff Shaw dan Herbert Simon di RAND Corporation *sebagai struktur data utama untuk bahasa Information Processing Language (IPL*). IPL dibuat untuk mengembangkan program *artificial intelligence*

- Linked List adalah salah satu bentuk struktur data, berisi kumpulan data (node) yang **tersusun** secara sekuensial, **saling sambung-menyambung** dan **dinamis.**

- **List** akan disimpan dalam bagian memori komputer yang dinamakan **HEAP**

- *Field terakhir* dalam setiap *record* sebagai tempat penyimpanan alamat memori (pointer) untuk record berikutnya yang terkait dalam List

KONSEP DASAR LIST (SENARAI)

1. Inisialisasi senarai (list) menjadi kosong
2. Periksa apakah senarai kosong atau tidak
3. Periksa apakah senarai penuh atau tidak
4. Mencari jumlah elemen (panjang) pada senarai
5. Ambil elemen ika tidak kosong
6. Masukkan/mengganti elemen jika senarai tidak kosong/penuh

HAL-HAL PENTING YANG HARUS DIKETAHUI MENGENAI LINK LIST

1. Linked List saling t erhubung dengan bantuan variabel POINTER
2. Pointer yang menunjuk pada awal dari list yang disebut HEAD.
3. Pointer yang menunjuk pada akhir dari list yang disebut TAIL, kecuali untuk jenis circular.
4. Masing-masing data dalam Linked List disebut dengan NODE (SIMPUL)
5. Setiap simpul yang terbentuk selalu memiliki nilai NIL, kecuali jika simpul tersebut sudah ditunjuk oleh simpul yang lainnya (Link list belum terhubung).
6. Posisi simpul terakhir pada link list selalu bernilai NIL karena tidak menunjuk pada simpul yang lainnya, kecuali bentuk circular.

BAB VIII. SINGLE LINKED LIST (SENARAI BERTANTAI TUNGGAL)

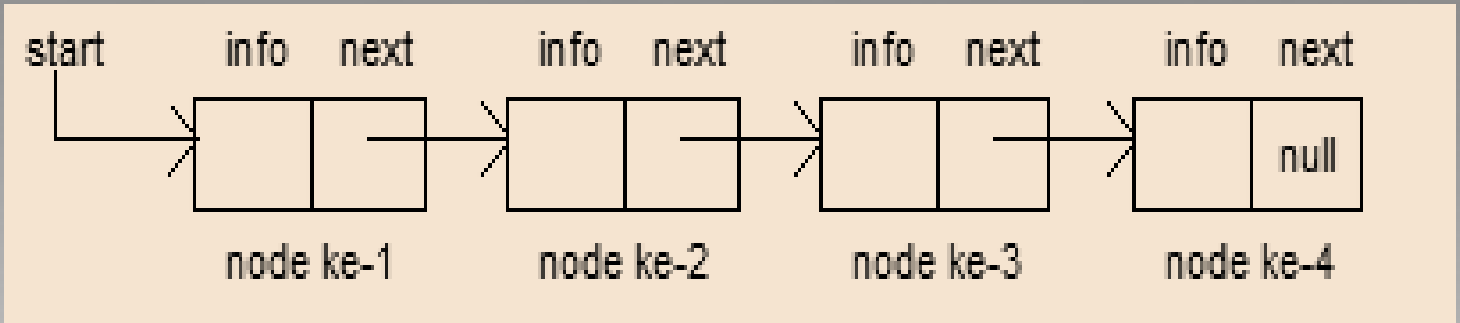
DEFINISI

Single Linked List (Senarai Berantai Tunggal) adalah linked list dimana semua simpul-simpulnya hanya memiliki 1 buah pointer (penunjuk) yang digunakan untuk mengkaitkan diri dengan simpul lain yang sejenis yang ada di sebelah kanannya di dalam sebuah senarai berantai yang sama.

Kebanyakan orang menyingkat Single Linked List (Senarai Berantai Tunggal) hanya dengan sebutan Linked List (Senarai Berantai).

Setiap simpul pada Senarai Berantai (Linked List) terdiri dari **2(dua) bagian yaitu :**

- **Info** berisi informasi tentang elemen **data** yang bersangkutan.
- **Next** (link field/next pointer field), berisi alamat dari elemen (node) selanjutnya yang dituju.



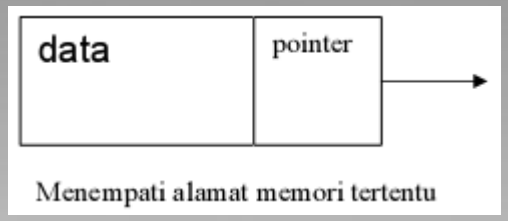
Pointer awal menunjuk ke simpul pertama dari senerai tersebut. Pointer dari suatu simpul yang tidak menunjuk simpul lain disebut **pointer kosong**, yang nilainya dinyatakan sebagai **NIL** (nil adalah kata baku Pascal yang berarti bahwa pointer 0 atau bilangan negatif).

Jadi hanya dengan sebuah pointer Awal saja maka kita bisa membaca semua informasi yang tersimpan dalam senerai.

ADA DUA JENIS SINGLE LINKED LIST :

1. Single Linked List (Non Circular)
2. Single Linked List Circular

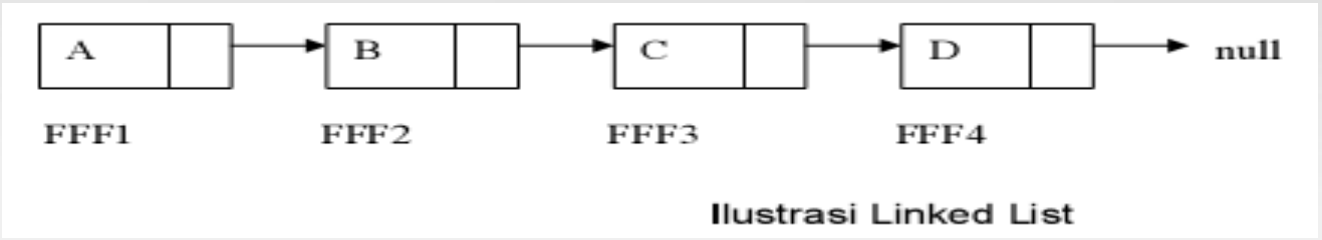
SINGLE LINKED LIST



Pengertian:

Single : artinya field pointer-nya hanya satu buah saja dan satu arah serta pada akhir node, pointernya menunjuk NIL

Linked List : artinya node-node tersebut saling terhubung satu sama lain.



SINGLE LINKED LIST CIRCULAR

Single Linked List yang pointer nextnya menunjuk pada dirinya sendiri. Jika Single Linked List tersebut terdiri dari beberapa node, maka pointer next pada node terakhir akan menunjuk ke node terdepannya.

Pengertian:

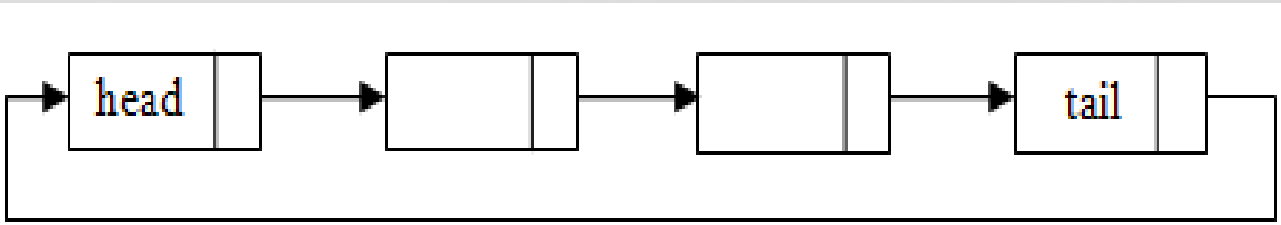
Single : artinya field pointer-nya hanya satu buah saja dan satu arah.

Circular : artinya pointer next-nya akan menunjuk pada dirinya sendiri sehingga berputar

Ilustrasi SINGLE LINKED LIST CIRCULAR

- Setiap node pada linked list mempunyai field yang berisi pointer ke node
  berikutnya, dan juga memiliki field yang berisi data.

- Pada akhir linked list, node terakhir akan menunjuk ke node terdepan sehingga linked list tersebut berputar. Node terakhir akan menunjuk lagi ke head.



Operasi Pada Linked List

Ada tiga jenis operasi pada linked list, yaitu :

1. *BUAT BARU*
2. *TAMBAH*
   a. Menambah di belakang
   b. Menambah di depan
   c. Menambah di tengah
3. *HAPUS*
   a. Menghapus simpul pertama
   b. Menghapus simpul di tengah
   c. Menghapus simpul di akhir
4. *CETAK/BACA* :
   a. Membaca maju
   b. Membaca mundur

1. Menambah Simpul

Untuk menjelaskan operasi ini baiklah kita gunakan deklarasi pointer dan simpul seperti di bawah ini

```
Type simpul = ^ data;
        Data = reecord
        Info : char;
        Berikut:simpul;
    End;
Var elemen : char;
    Awal, akhir, baru : simpul;
```

- **Menambah simpul baru di belakang/akhir** dari list.

  Penambahan di belakang maksudnya menambahkan simpul-simpul baru pada posisi **Tail**.

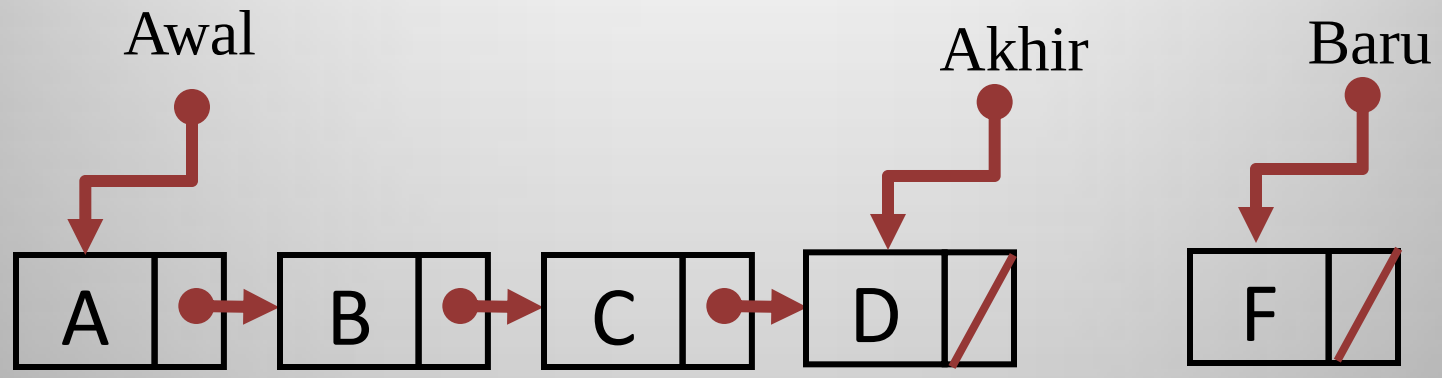
  Di bawah ini gambar ilustrasi penambahan simpul di belakang.

Untuk menyambung simpul yang ditunjuk oleh *akhir* dan *Baru*, pointer pada simpul yang ditunjuk oleh simpul *akhir* dibuat sama dengan *baru* kemudian pointer *akhir* dibuat sama dengan pointer *baru*.

Dalam hal ini perlu pula ditest apakah senarai berantai masih kosong atau tidak. Senarai berantai yang masih kosong ditandai dengan nilai pointer *Awal* yang nilainya sama dengan NIL

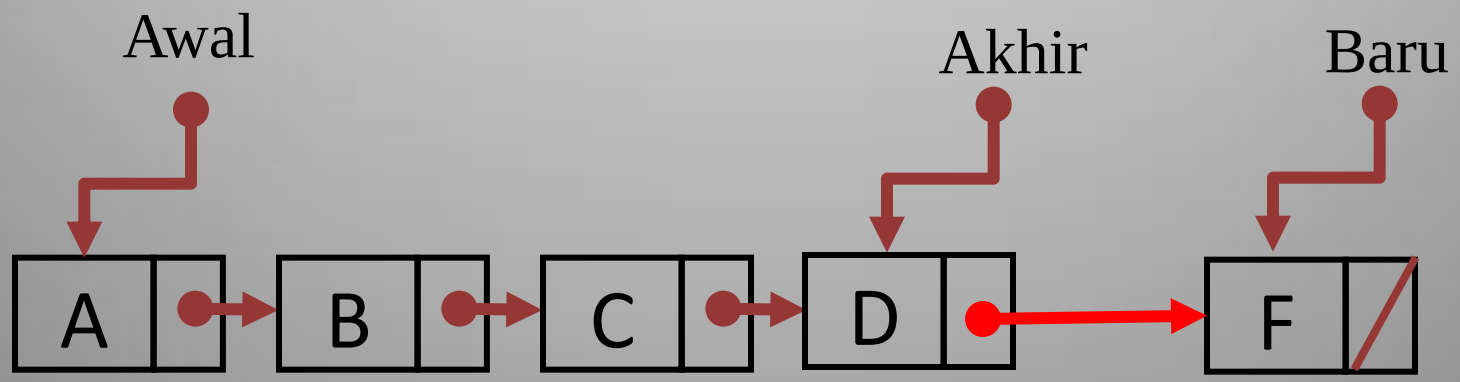
Illustrasi Penambahan Simpul di Belakang

a.



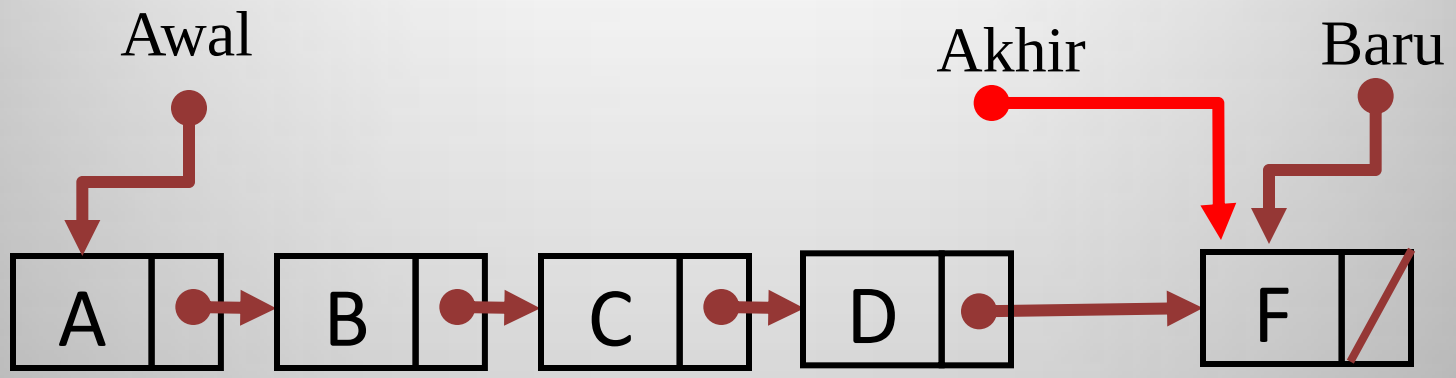

```
Akhir^.Berikut := Baru ;
```

b.



```
Akhir:= Baru ;
Akhir^.Berikut := nil
```

c.




```
Procedure tambah_belakang (var Awal, akhir : simpul; elemen : char );
Var Baru : simpul ;
Begin
 new(baru);
 baru^.info:=elemen;
 akhir^.berikut := baru;
 akhir:=baru;
 akhir^.berikut:= nil;
End;
```

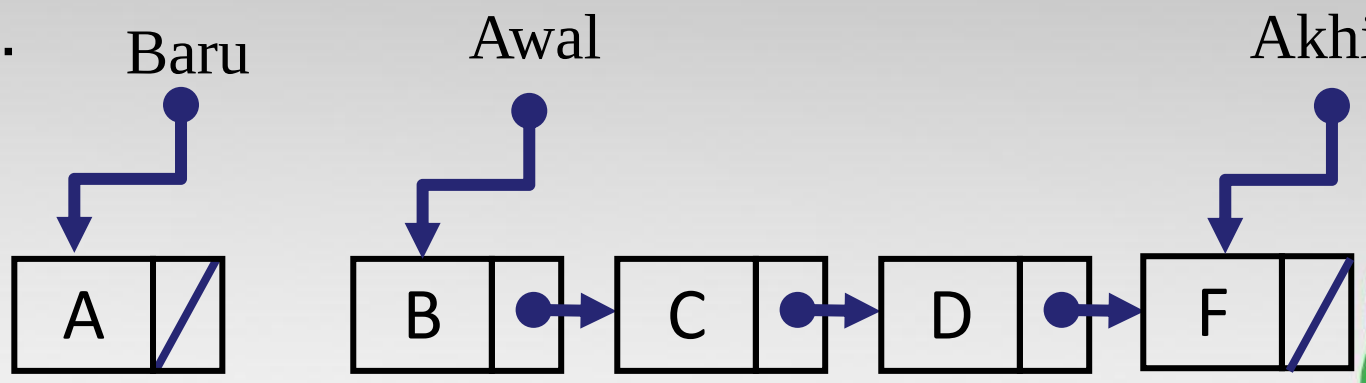
Menambah simpul baru di depan/awal

Penambahan di depan maksudnya menambahkan simpul-simpul baru pada posisi **Head**.

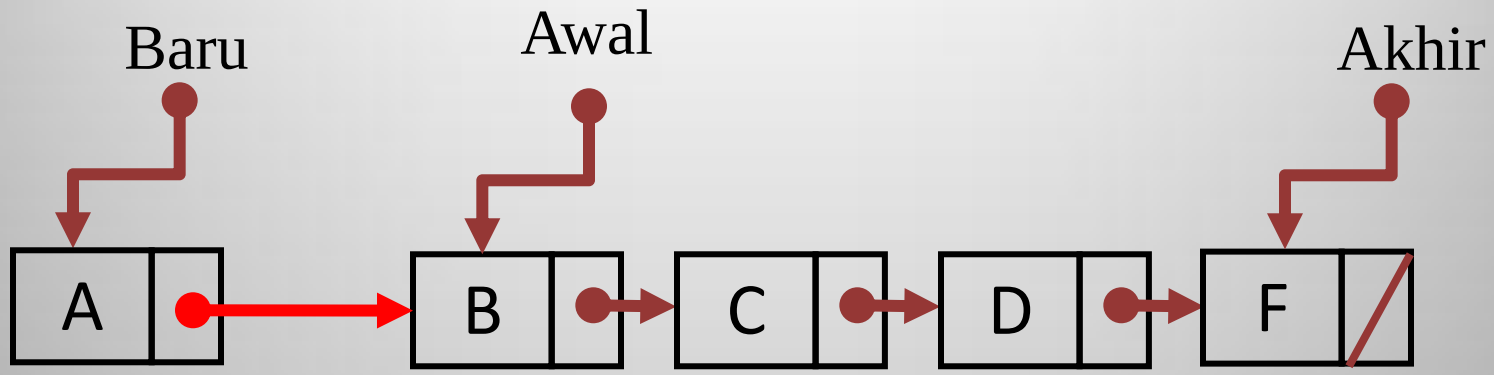
Di bawah ini gambar ilustrasi penambahan simpul di depan.

Illustrasi Penambahan Simpul di Depan/Awal

a.

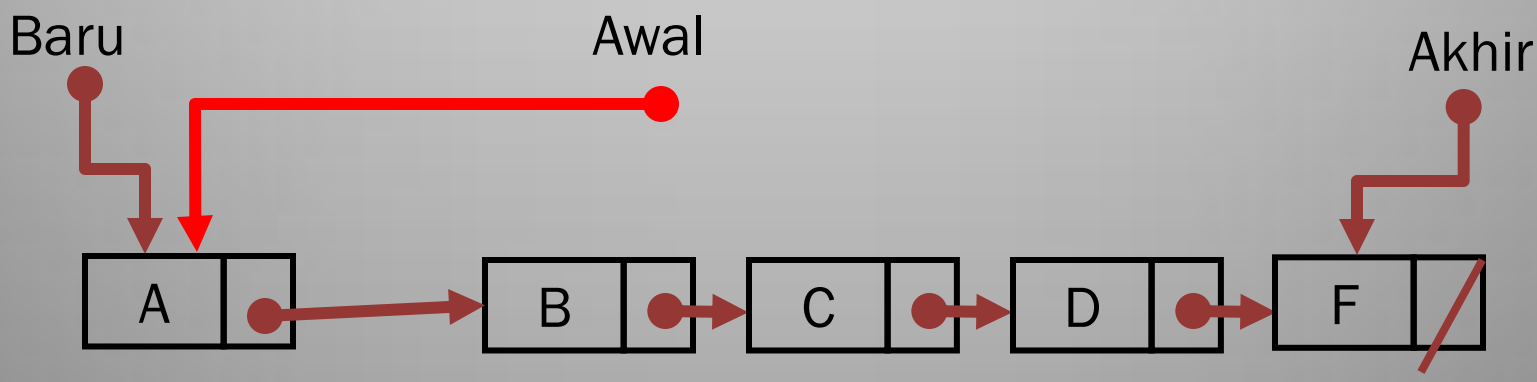


b. **`Baru^.Berikut := Awal ;`**



---

c. **Awal:= Baru ;**



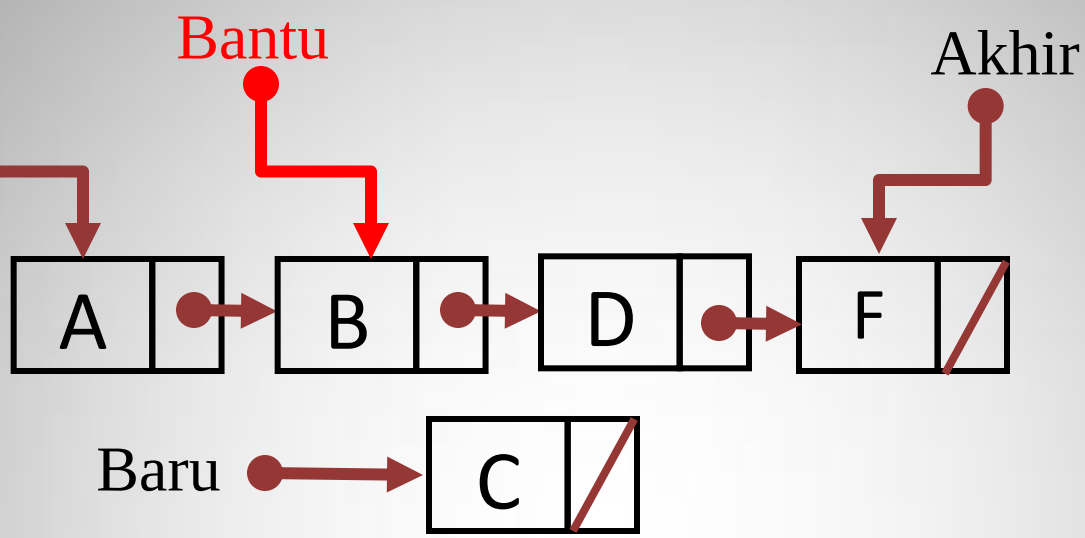
```
Procedure tambah_depan(var Awal, akhir : simpul; elemen : char );
Var Baru : simpul ;
Begin
        new(Baru);Baru^.Info := Elemen ;
        if Awal := nil then
                Akhir := Baru
        else
                Baru^.Berikut := Awal;
        Awal := Baru;
End;
```

Menambah simpul baru ditengah.

Pertama kali ditentukan di mana simpul baru akan ditambahkan pada posisi urutan simpul yang ke berapa. Hal ini dapat dilakukan dengan menempatkan simpul pointer bantu pada posisi tertentu. Kemudian pointer yang ditunjuk oleh pointer simpul baru dibuat sama dengan pada simpul yang ditunjuk oleh simpul bantu, selanjutnya pointer pada simpul yang ditunjnuk oleh simpul bantu, selalnjutnya pointer pada simpul yang ditunjuk pada simpul yang ditunjuk oleh simpul bantu dibuat sama dengan baru. Perhatikan bahwa urutan ini tidak boleh dibalik.

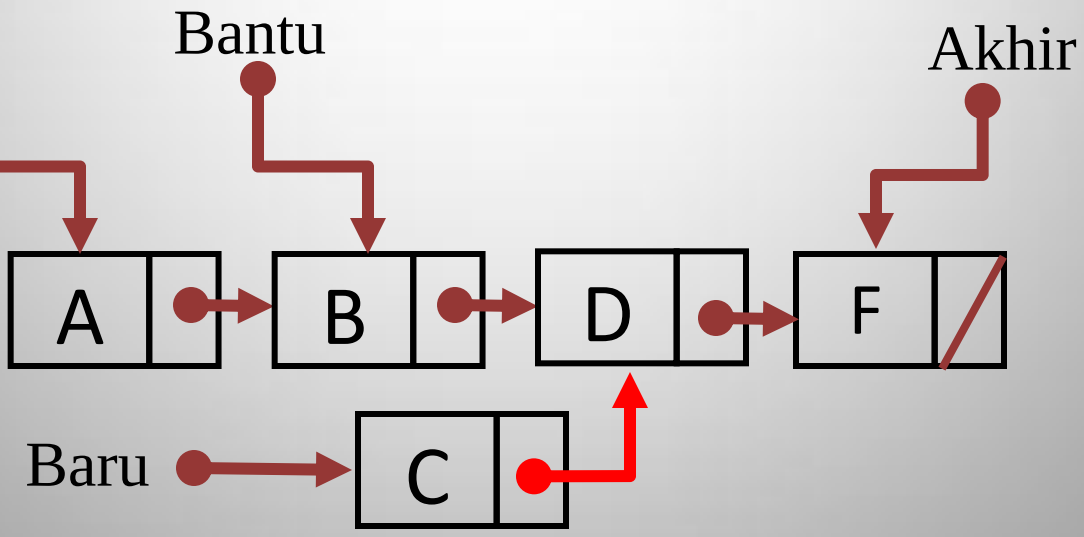
Illustrasi Penambahan Simpul di Tengah

a.

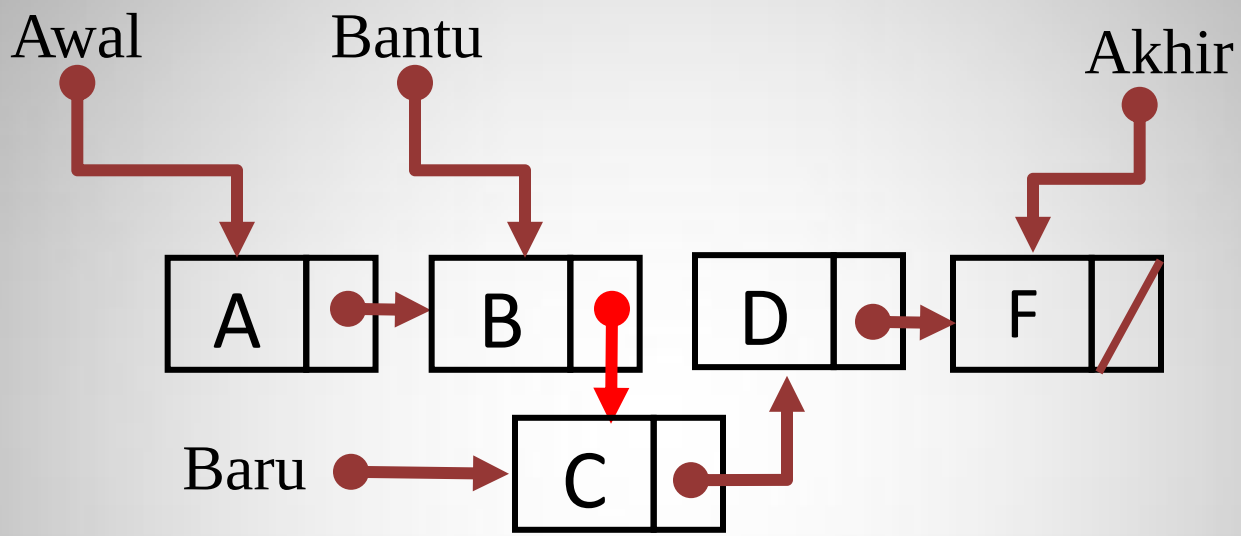



```
Baru^.Berikut := Bantu^.Berikut;
```

b.



c. **`Bantu^.Berikut := Baru ;`**



```
Procedure Tambah_Tengah (var awal,akhir:simpul ; elemen:char);
Var bantu,baru:simpul;
Begin
    new(baru);  baru^.info :=elemen;
      if awal=nil then   {senarai masih kosong}
      begin
          awal:= baru;  akhir:= baru;
     end
      else
          begin {mencari posisi dimana elemen akan disisipkan}
                bantu := awal;
                while elemen > baru^.berikut^.info do
                          bantu := bantu^.berikut;
                 {menyisipkan simpul baru}
                    baru^.berikut := bantu^.berikut;
                    bantu^.berikut:= baru;
          end; {endif}
  End;
```

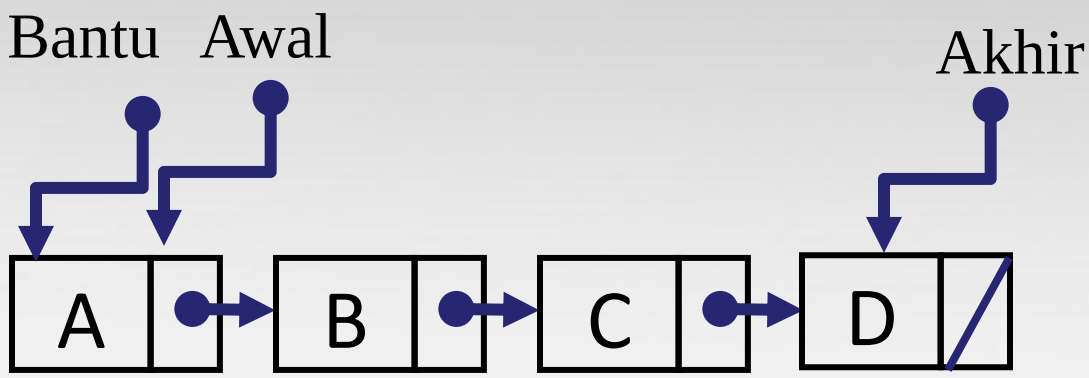
2 . Menghapus Simpul

- **Menghapus simpul pertama**

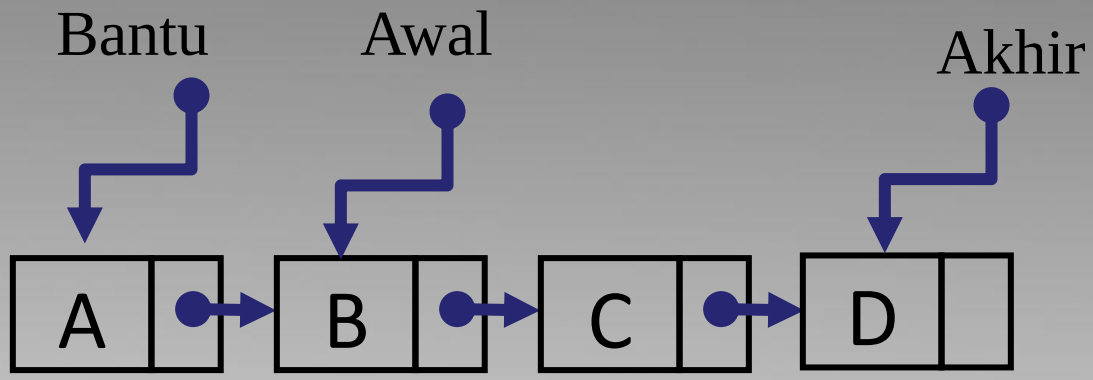
  Untuk menghapus simpul pertama, maka pointer Bantu kita buat sama dengan pointer Awal. Kemudian pointer Awal kita pindah ke simpul yang ditunjjuk oleh pointer pada simpul yang ditunjuk oleh pointer Bantu kita dispose.

Illustrasi Penghapusan Simpul di Pertama

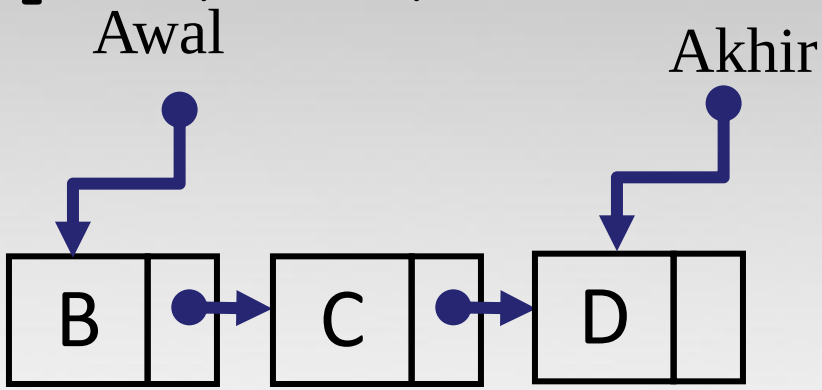
a.



b. **`Awal:= Bantu^.Berikut;`**



c. **`Dispose(bantu);`**



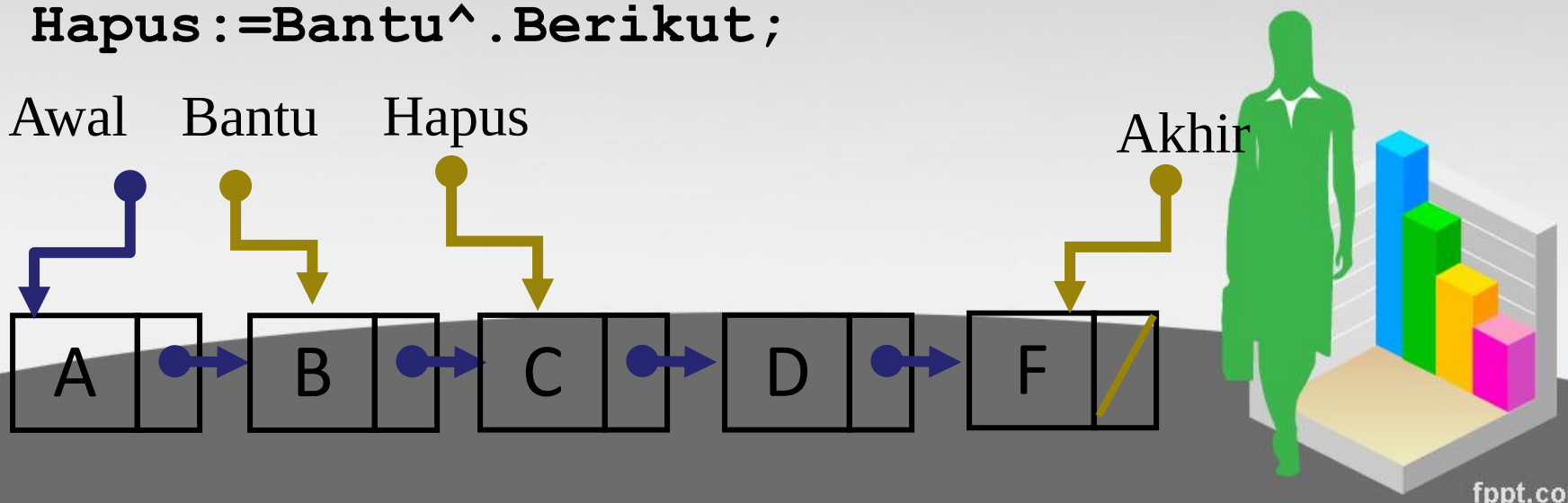
```
procedure Hapus_Awal(var Awal, Akhir : Simpul; Elemen :
char);
var Hapus, Bantu : Simpul;
begin
if Awal = nil then { jika senarai masih kosong }
writeln('Senarai masih kosong')
else if Awal^.Info = Elemen then
{ simpul pertama yang dihapus }
begin
Hapus := Awal;
Awal := Hapus^.Berikut; Hapus^.Berikut := 0;
dispose(Hapus);
end;
end;
```

Menghapus simpul di tengah

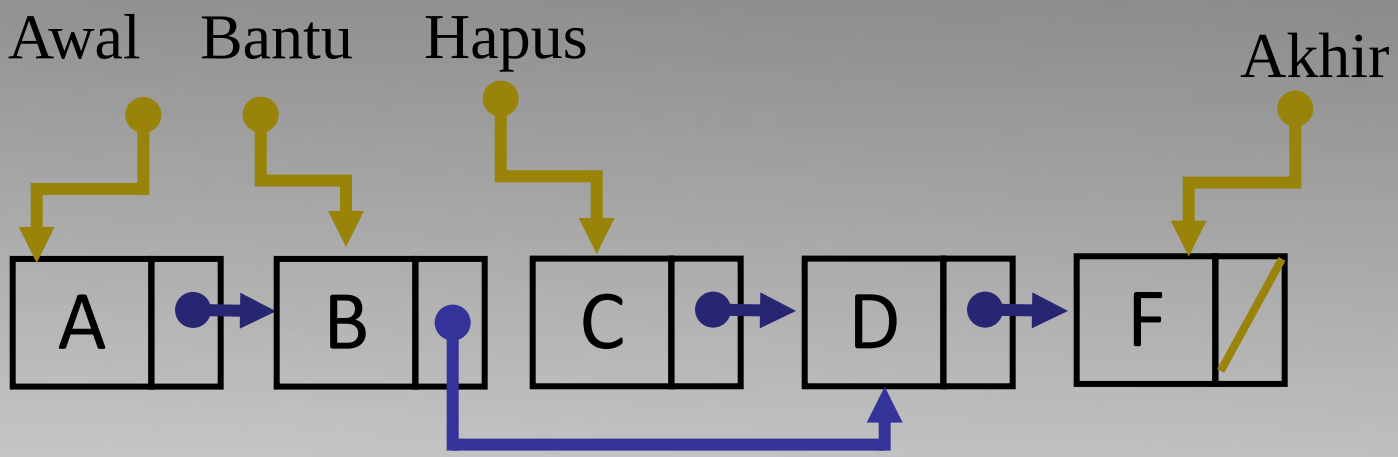
Untuk menghapus simpul tengah, letakkan pointer Bantu pada simpul di sebelah kiri simpul yang akan dihapus. Simpul yang akan dihapus kita tunjuk dengan pointer lain, misalnya Hapus. Kemudian pointer pada simpul yang ditunjuk oleh Bantu kita tunjukkan pada simpul yang ditunjuk oleh pointer pada simpul yang akan dihapus. Selanjutnya simpul yang ditunjuk oleh pointer Hapus kita Dispose.

Illustrasi Penghapusan Simpul di tengah

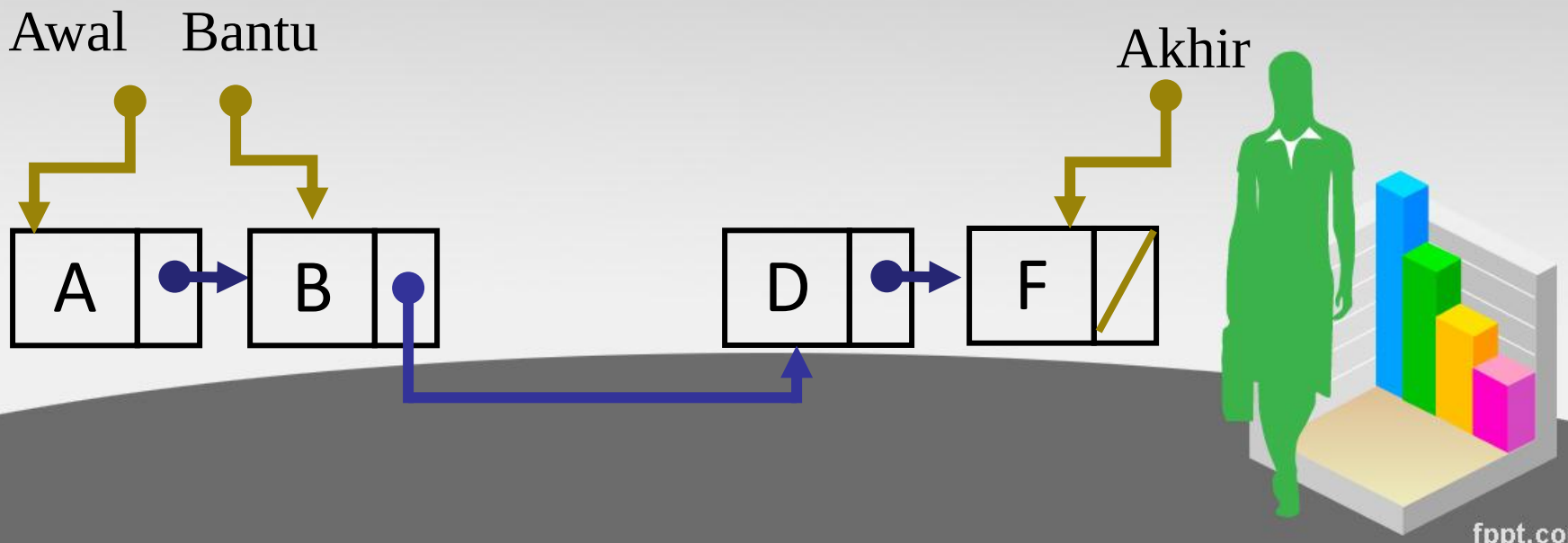
a.
```
Bantu:=Awal^.Berikut;
Hapus:=Bantu^.Berikut;
```



b. **`Bantu^.Berikut:=Hapus^.Berikut;`**



c. **`Dispose(hapus);`**



Procedure Hapus_Tengah

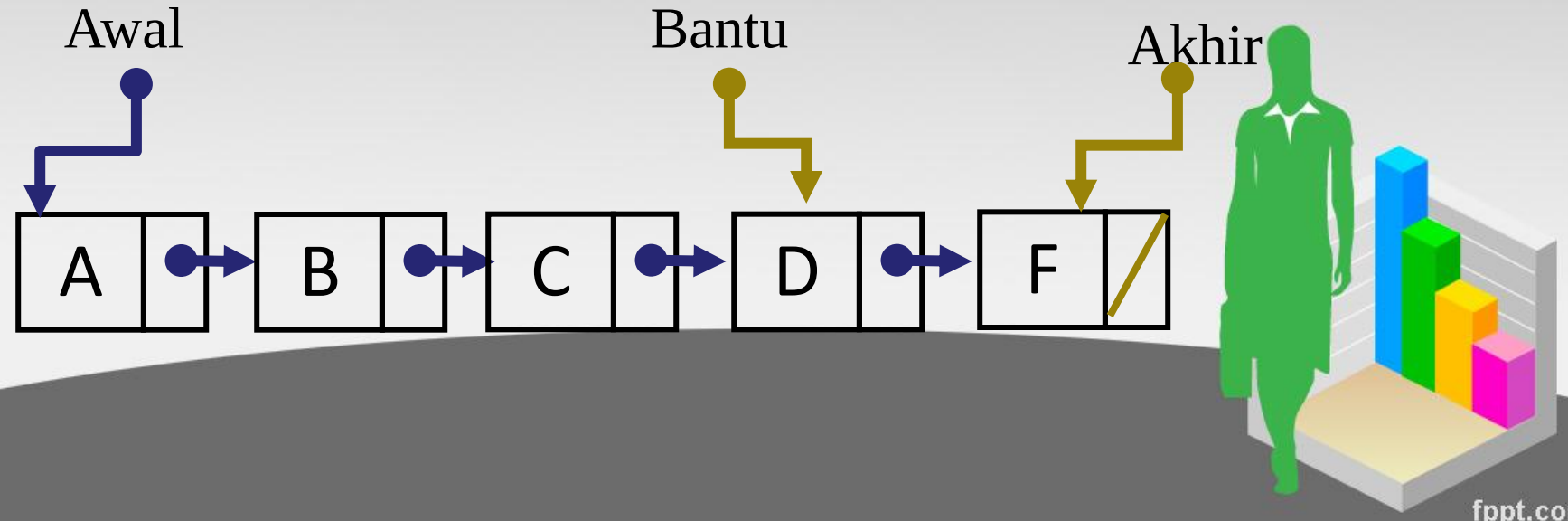
```
begin
Bantu := Awal;
{ mencari simpul yang akan dihapus }
while (Elemen <> Bantu^.Info) and
(Bantu^.Berikut <> nil) do
Bantu := Bantu^.Berikut;
Hapus := Bantu^.Berikut;
if Hapus <> nil then
{ simpul yang akan dihapus ketemu }
```

Menghapus simpul di akhir

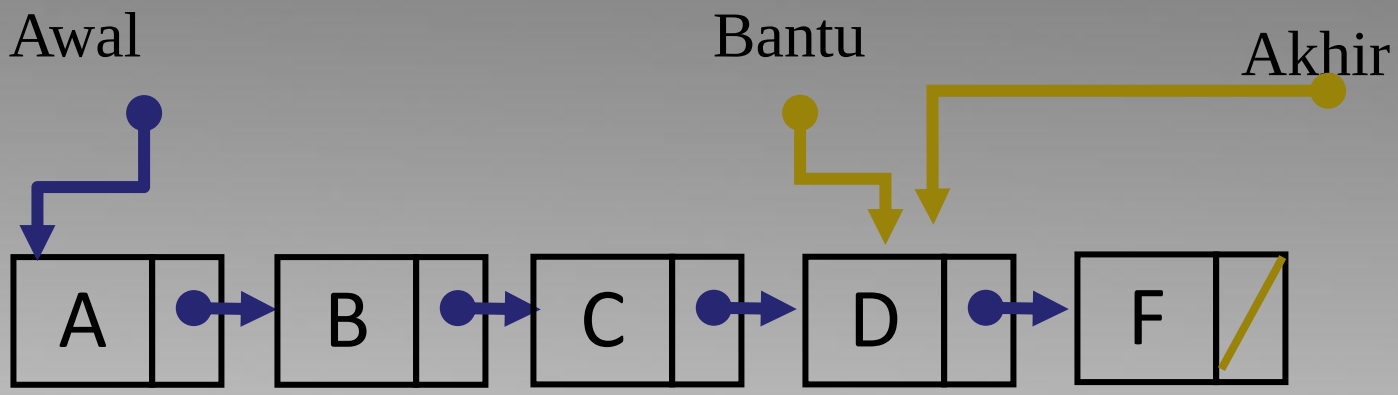
Untuk menghapus simpul terakhir, tidak bisa dilakukan hanya dengan mendispose simpul akhir. Karena dengan disposenya simpul akhir, maka hanya simpul akhir saja yang dihapus, tetapi simpulnya masih tetap bisa dimasup, karena masih ditunjuk oleh pointer dari simpul di sebeah kirinya. Sehingga untuk bisa menghapus simpul terakhir, selain simpul akhir kita hapus, maka pointer dari sebelah kirinya juga harus dijadikan nil.

Illustrasi Penghapusan Simpul di akhir

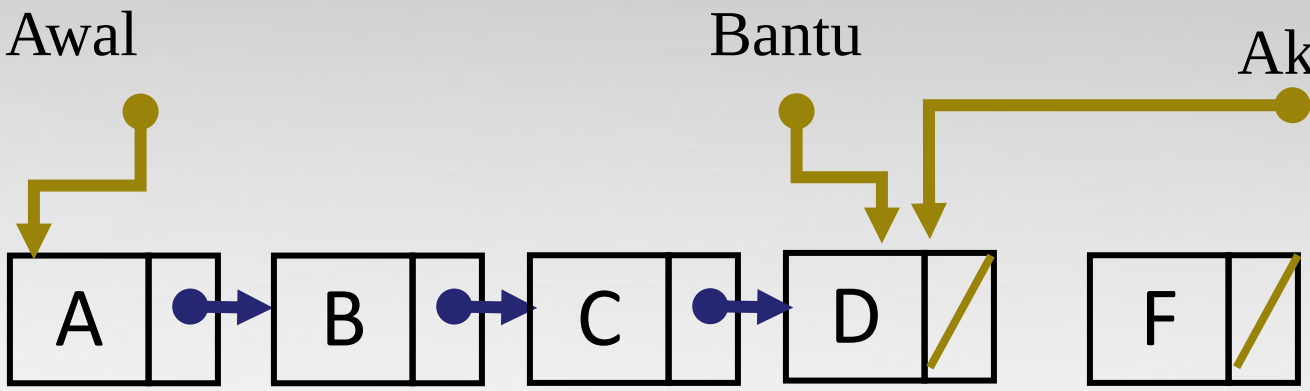
a. `Bantu:=Bantu^.Berikut;`



b. **`Akhir:=Bantu;`**



c. **`Bantu^.Berikut = nil;`**



Procedure Hapus_Akhir

```
begin
if Hapus <> Akhir then
{ simpul di tengah dihapus }
Bantu^.Berikut := Hapus^.Berikut;
else
{ simpul terakhir dihapus }
begin
Akhir := Bantu;
Akhir^.Berikut := nil;
end;
Hapus^.Berikut := 0;
dispose(Hapus);
end
else
{ simpul yang aka dihapus tidak ketemu }
writeln('Simpul yang akan dihapus tidak ketemu !');
end;
end;
```