UNIVERSITAS MERCU BUANA

**MODUL PERKULIAHAN**

# Algoritma dan Struktur Data

## Modul Standar untuk digunakan dalam Perkuliahan di Universitas Mercu Buana

| Fakultas | Program Studi | Tatap Muka | Kode MK | Disusun Oleh |
|---|---|---|---|---|
| Fakultas Ilmu Komputer | Sistem Informasi | 7-9 | W181700005 | Sulis Sandiwarno, S.Kom.,M.Kom |


**Abstract**

Ilmu komputer saat ini sedang berkembang dengan sangat pesat. Kemajuan dalam bidang ilmu Komputer ditunjukan dari beberapa aspek, seperti pada bidang penelitian. Penelitian ini berfokus pada tujuan dan manfaat mempelajari penelitian tersebut.


**Kompetensi**

Dalam Pembahasan Kali Ini Dalam Pertemuan – 7-9 Matakuliah Algoritma dan Struktur Data akan membahas mengenai Struktur

# Pengenalan Struktur

Structure (struktur) adalah kumpulan elemen data yang digabungkan menjadi satu kesatuan. Dengan kata lain, structure merupakan bentuk struktur data yang dapat menyimpan variabel-variabel dalam satu nama. Masing-masing elemen data dikenal dengan sebuah field. Masing-masing field dapat memiliki tipe data yang sama ataupun berbeda. Variabel-variabel tersebut memiliki kaitan satu sama lain. Walaupun field-field tersebut berada dalam satu kesatuan, namun masing-masing field tersebut tetap dapat diakses secara individual atau satu persatu. Field-field digabungkan menjadi satu dengan tujuan untuk kemudahan dallam operasinya. Struktur membantu mengatur data yang rumit, khususnya dalam program besar, karena struktur akan membuat sekelompok elemen data diperlakukan sebagai suatu unit.

Struktur atau *structure* adalah sekumpulan variabel yang masing – masing tipe datanya bias berbeda, dan dikelompokkan ke dalam satu nama. Struktur juga dikenal sebagai *record* dalam *C.* Struktur membantu mengatur data – data yang rumit, khususnya dalam program yang besar karena struktur membirkan sekelompok variabel diperlakukan sebagai satu unit.

Struktur ini sering digunakan untuk mendefinisikan suatu *record* yang disimpan didalam *file.* Struktur termasuk kedalam tipe data yang dibangkitkan, yang disusun menggunakan obyek dengan tipe lain.

Bentuk umum dari struktur adalah sebagai berikut:

```
struct mhs
{
    char *nama;
    char  *nim;
    int uts, uas;
    float akhir;
    char mutu;
}
```

Kata kunci *struct* menunjukkan definisi struktur, dan identifikasi menunjukkan *structure tag.* Dengan demikian terdapat tipe data baru bernama *struct mhs*¸yang terdiri dari nama mahasiswa, nilai ujian tengan semester, akhir semester, nilai akhir dan huruf mutu yang masing – masing disebut sebagai *field.*

# Deklarasi pada Struktur

Pendeklarasian structure selalu diawali kata baku struct diikuti nama structure dan deklarasi field-field yang membangun structure di antara pasangan tanda kurung kurawal buka dan kurung kurawal tutup yang diakhiri dengan tanda titik koma ( ; ). Jika terdapat field dengan tipe data yang sama, dapat dideklarasikan bersama dengan dipisahkan oleh tanda koma ( , ) sehingga tidak perlu menuliskan tipe datanya berulang-ulang.

Bentuk umum

```
struct nama_struct

{

    <tipe_data> nama_field_1;

    <tipe_data> nama_field_2;

    ...

    <tipe_data> nama_field_n;

}
```

Contoh-1 Program Struktur adalah sebagai berikut:

```
#include<stdio.h>
#include<conio.h>
#include<iostream.h>
void main(){
        int i;
    struct{
        char nim[5];
    char nama[15];
        float nilai;
} mhs[5];

clrscr();
```

```
            for(i=1; i<=2; i++){
   cout<<"masukan NIM = ";
            cin>>mhs[i].nim;
   cout<<"masukan Nama = ";
            cin>>mhs[i].nama;
   cout<<"masukan Nilai Akhir = ";
            cin>>mhs[i].nilai;
}
cout<<endl;
cout<<"data Mahasiswa"<<endl;

for(i=1; i<=2; i++){
             cout<<"Data Ke - "<<i<<endl;
             cout<<"NIM = "<<mhs
             [i].nim<<endl;
             cout<<"Nama = "<<mhs
             [i].nama<<endl;
cout<<"Nilai Akhir = "<<mhs
             [i].nilai<<endl;
             cout<<endl;
}
getch();
}
```

```
#include<stdio.h>
#include<conio.h>
#include<iostream.h>

main()
{
  int i, j=1;
  struct
  {
          char nim[20];
          char nama[30];
          float nilai1;
          float nilai2;
          float nilai3;
          float nilai4;
          float hasil;
```

```
 } mhs[35];

 clrscr();
 for(i=0; i<1; i++)
 {
        cout<<"                 Rekap Nilai Mahasiswa Menggunakan Structure"<<endl;
        cout<<"                ---------------------------------------------"<<endl;
        cout<<"  "<<endl;
        cout<<"Masukan NIM          = "; cin>>mhs[i].nim;
        cout<<"masukan Nama         = "; cin>>mhs[i].nama;
        cout<<"masukan Nilai Absensi = "; cin>>mhs[i].nilai1;
        cout<<"masukan Nilai Tugas   = "; cin>>mhs[i].nilai2;
        cout<<"masukan Nilai UTS     = "; cin>>mhs[i].nilai3;
        cout<<"masukan Nilai UAS     = "; cin>>mhs[i].nilai4;
        mhs[i].hasil = (mhs[i].nilai1 * 0.10)+ (mhs[i].nilai2 * 0.20 ) + (mhs[i].nilai3 * 0.30) + (mhs[i].nilai4 *
0.40);
        cout<<endl;
 }

 for(i=0; i<1; i++)
 {
        cout<<"                      Hasil Rekap Nilai Mahasiswa "<<endl;
        cout<<"                     ----------------------------"<<endl;
        cout<<"Data Mahasiswa Ke - "<<j++<<endl;
        cout<<"NIM Mahasiswa       = "<<mhs[i].nim<<endl;
        cout<<"Nama Mahasiswa      = "<<mhs[i].nama<<endl;
        cout<<"Nilai Absensi       = "<<mhs[i].nilai1<<endl;
        cout<<"Nilai Tugas         = "<<mhs[i].nilai2<<endl;
        cout<<"Nilai UTS           = "<<mhs[i].nilai3<<endl;
        cout<<"Nilai UAS           = "<<mhs[i].nilai4<<endl;
        cout<<"Nilai Akhir         = "<<mhs[i].hasil<<endl;

        cout<<"============================================================"<<endl;
   cout<<"Tulis kan Nama Pembuat      : Sulis Sandiwarno (Sebagai Contoh)"<<endl ;
        cout<<"Tulis kan NIM Pembuat       : 41806010036 (Sebagai Contoh)"<<endl ;
        cout<<"Tulis kan Nama Program Studi : Sistem Informasi (Sebagai Contoh)"<<endl ;

        cout<<endl;
 }

 getch();
}
```

# Struktur dalam Struktur

Suatu struktur juga dapat mengandung suatu struktur yang lain. Artinya *field – field* dalam suatu *structure* merupakan *structure* juga. Misalkan biodata mahasiswa yang terdiri dari NIM, Nama, Alamat dan Tanggal Lahir. Alamat terdiri dari Nama Jalan, Kota dan Kode Pos. Tanggal lahir juga terdiri dari Tanggal, Bulan dan Tahun.

Suatu struktur juga dapat mengandung suatu struktur yang lain. Artinya field-field dalam suatu structure merupakan suatu structure juga. Misalkan biodata Mahasiswa yang terdiri dari NIM, Nama, Alamat, dan Tanggal Lahir. Alamat terdiri dari Nama Jalan, Kota, dan Kode Pos. Demikian juga halnya dengan Tanggal terdiri dari Tanggal, Bulan, dan Tahun. Dengan demikian maka struktur dibagi dalam tiga struktur, yaitu:

```
struct tinggal
{
   char jalan[40];
   char kota[15];
   char pos[5];
};

struct tgl_lahir
{
   int tanggal;
   int bulan;
   int tahun;
};

struct mahasiswa
{
   char nim[12];
   char nama[25];
   tinggal alamat;
   tgl_lahir lahir;
};
```

Contoh Program *struct dalam struct* adalah sebagai berikut :

```
#include<conio.h>
#include<stdio.h>

struct nilai
{
        int nilai1;
        int nilai2;
        int nilai3;
};
```

```
struct mahasiswa
{
        char nama[50];
        char nim[50];
        char alamat[100];
        char asal[100];
        struct nilai_akhir;
};

struct mahasiswa mhs;

void main()
{
        cout<<"Nama  : ";gets(mhs.nama);
        cout<<"NIM     : ";gets(mhs.nim);
        cout<<"Alamat: ";gets(mhs.alamat);
        cout<<"Asal    : ";gets(mhs.asal)
        cout<<"Nilai1  : ";cin>>mhs.nilai.nilai1;
        cout<<"Nilai2  : ";cin>>mhs.nilai.nilai2;
        cout<<"Nilai3  : ";cin>>mhs.nilai.nilai3;
        cout<<"==================================="<<endl;

        cout<<"Nama  :"<<mhs.nama<<endl;
        cout<<"NIM     :"<<mhs.nim<<endl;
        cout<<"Alamat:"<<mhs.alamat<<endl;
        cout<<"Asal    :"<<mhs.asal<<endl;
        cout<<"Nilai 1 :"<<mhs.nilai.nilai1<<endl;
        cout<<"Nilai 2 :"<<mhs.nilai.nilai2<<endl;
        cout<<"Nilai 3 :"<<mhs.nilai.nilai3<<endl;
getch();

}
```

Contoh - 2

```
#include<stdio.h>
#include<iostream.h>
#include<conio.h>


struct Tinggal
{
  char Jalan[40];
  char Kota[15];
  char Pos[5];
};


struct Tgl_Lahir
```

```
{
  int Tanggal;
  int Bulan;
  int Tahun;
};
struct Mahasiswa
{
  char Nim[9];
  char Nama[25];
  Tinggal Alamat;
  Tgl_Lahir Lahir;
};
void main()
{
  Mahasiswa Mhs;
  cout<<"NIM   : "; cin.getline(Mhs.Nim,9);
  cout<<"Nama          : "; cin.getline(Mhs.Nama,25);
  cout<<"Alamat        :\n";
  cout<<"\tJalan       : "; cin.getline(Mhs.Alamat.Jalan,40);
  cout<<"\tKota        : "; cin.getline(Mhs.Alamat.Kota,15);
  cout<<"\tKode Pos    : "; cin.getline(Mhs.Alamat.Pos,5);
  cout<<"Tanggal Lahir          :\n";
  cout<<"\tTanggal     : "; cin>>Mhs.Lahir.Tanggal;
  cout<<"\tBulan       : "; cin>>Mhs.Lahir.Bulan;
  cout<<"\tTahun       : "; cin>>Mhs.Lahir.Tahun;
  cout<<"\n\nMencetak Kembali Nilai Anggota\n\n";
  cout<<"NIM   : "<<Mhs.Nim;
  cout<<"\nNama        : "<<Mhs.Nama;
  cout<<"\nAlamat      :\n";
  cout<<"\n\tJalan      : "<<Mhs.Alamat.Jalan;
  cout<<"\n\tKota       : "<<Mhs.Alamat.Kota;
  cout<<"\n\tKode Pos: "<<Mhs.Alamat.Pos;
  cout<<"\nTanggal Lahir       : "<<Mhs.Lahir.Tanggal<<"-";
  cout<<Mhs.Lahir.Bulan<<"-"<<Mhs.Lahir.Tahun;
  getch();
}
```

# Linked List

*Linked list* merupakan suatu cara untuk menyimpan data dengan struktur sehingga *programmer* dapat secara otomatis menciptakan suatu tempat baru untuk menyimpan data kapan saja. Struktur dinamis ini memiliki beberapa keuntungan disbanding struktur *array* yang bersifat statis. Struktur ini lebih dinamis karena banyaknya elemen dengan mudah ditambah atau dikurangi, berbeda dengan *array* yang ukurannya bersifat tetap.

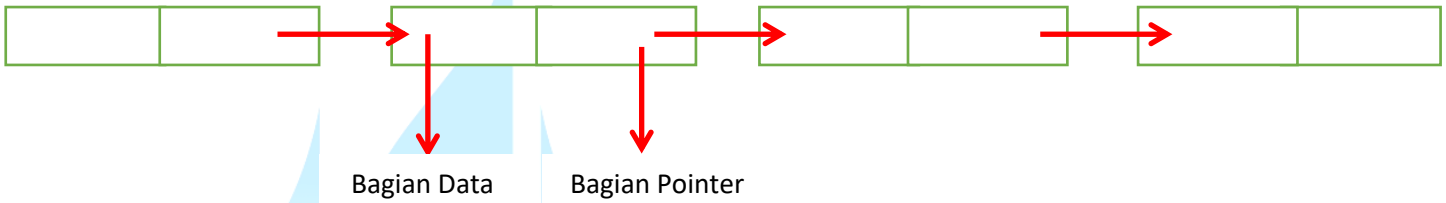
Penyimpanan dan pengolahan data dari sekelompok data yang telah terorganisir dalam sebuah pengurutan tertentu dapat dilakukan dengan menggunakan *array* seperti yang telah dibahas pada bab sebelumnya. Cara lain untuk menyimpan dan mengolah sekumpulan data seperti diatas juga dapat dilakukan dengan tipe data pointer. Penggunaan pointer sangat mendukung dalam pembentukan struktur data dinamis. Salah satu struktur data dinamis adalah *linked list.* Berarti *linked list* merupakan kumpulan komponen yang saling berkaitan satu dengan yang lain melalui *pointer.* Masing – masing komponen sering disebut dengan simpul atau *node atau verteks.*

*Linked list* dapat disajikan dengan 2 bagian besar yaitu :

1. *Single linked list*
2. *Double linked list*

# Linked List

Merupakan *linked list* paling sederhana dimana setiap simpul dibagi menjadi dua bagian yaitu bagian isi dan bagian *pointer.* Bagian isi merupakan bagian yang berisi data yang disimpan oleh simpul, sedangkan bagian pointer merupakan bagian yang berisi alamat dari simpul berikutnya. Sebagai ilustrasi seperti pada gambar berikut ini.



Dari gambar tersebut terlihat bahwa simpul pertama dihubungkan ke simpul kedua melalui bagian pointer simpul pertama. Bagian pointer simpul kedua dihubungkan kesimpul ketiga. Demikian seterusnya hingga simpul terakhir. Bagian pointer simpul terakhir tidak dihubungkan ke simpul lain yang disebut sebagai NULL.

# Operasi pada Linked List

Ada sejumlah operasi yang dapat dilakukan pada sebuah *single linked list,* seperti menambah simpul, menghapus simpul, membaca isi *linked list* atau pencarian informasi pada suatu *linked list.*

a. Menambah simpul

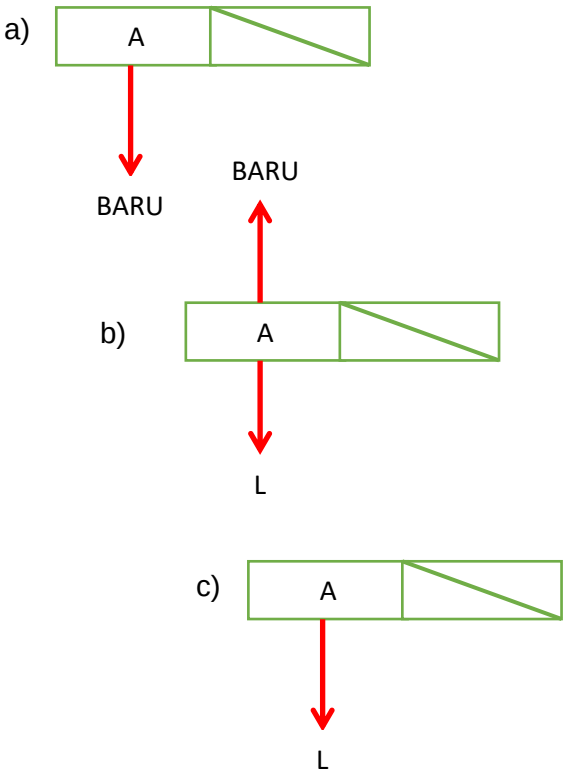
Operasi yang digunakan untuk menyisipkan simpul diposisi tertentu. Penyisipan simpul dapat dilakukan di posisi depan, penyisipan simpul dibelakang, penyisipan simpul diantara dua simpul (simpul tengah).

1. Menambah simpul depan

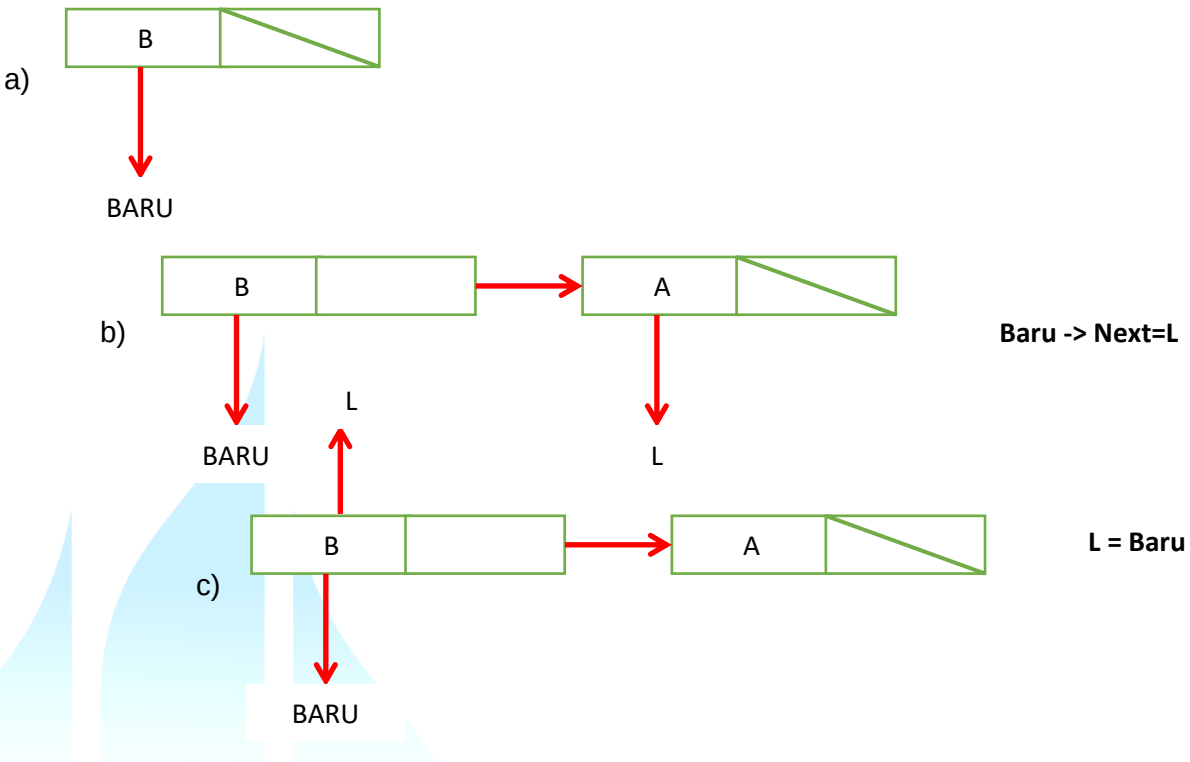
Operasi ini akan menyisipkan simpul baru selalu berada pada posisi pertama atau depan dari *linked list.* Langkah – langkah penyisipan simpul depan dapat dilakukan dengan:

- Ciptakan simpul baru yang akan disisipkan
- Jika *linked list* belum ada maka simpul baru menjadi *linked list* (L = Baru)
- Jika *linked list* sudah ada maka penyisipan dilakukan dengan cara:
  - *Pointer next* simpul baru menunjuk L (Baru ->Next=L)
  - *Pointer* L dipindahkan ke baru (L = Baru)

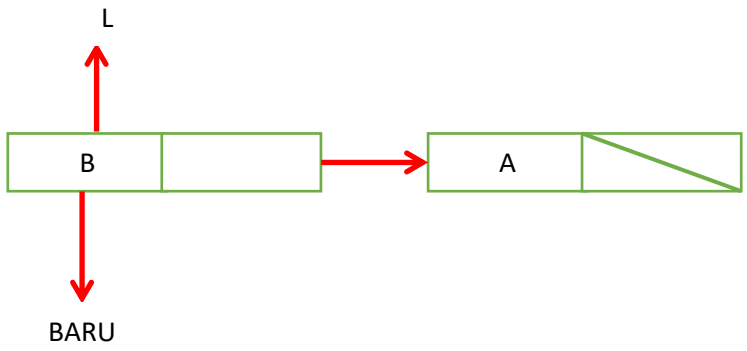
Skema penyisipan simpul depan seperti gambar berikut ini :




keterangan : Penyisipan simpul dengan *linked list* belum ada.

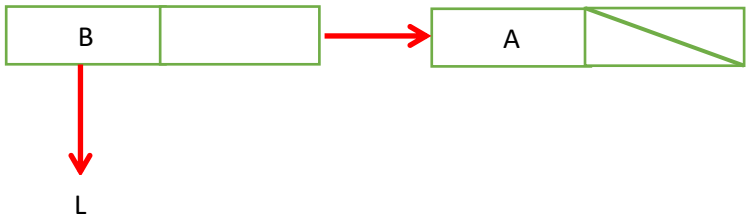


c)




**L = Baru**

d)

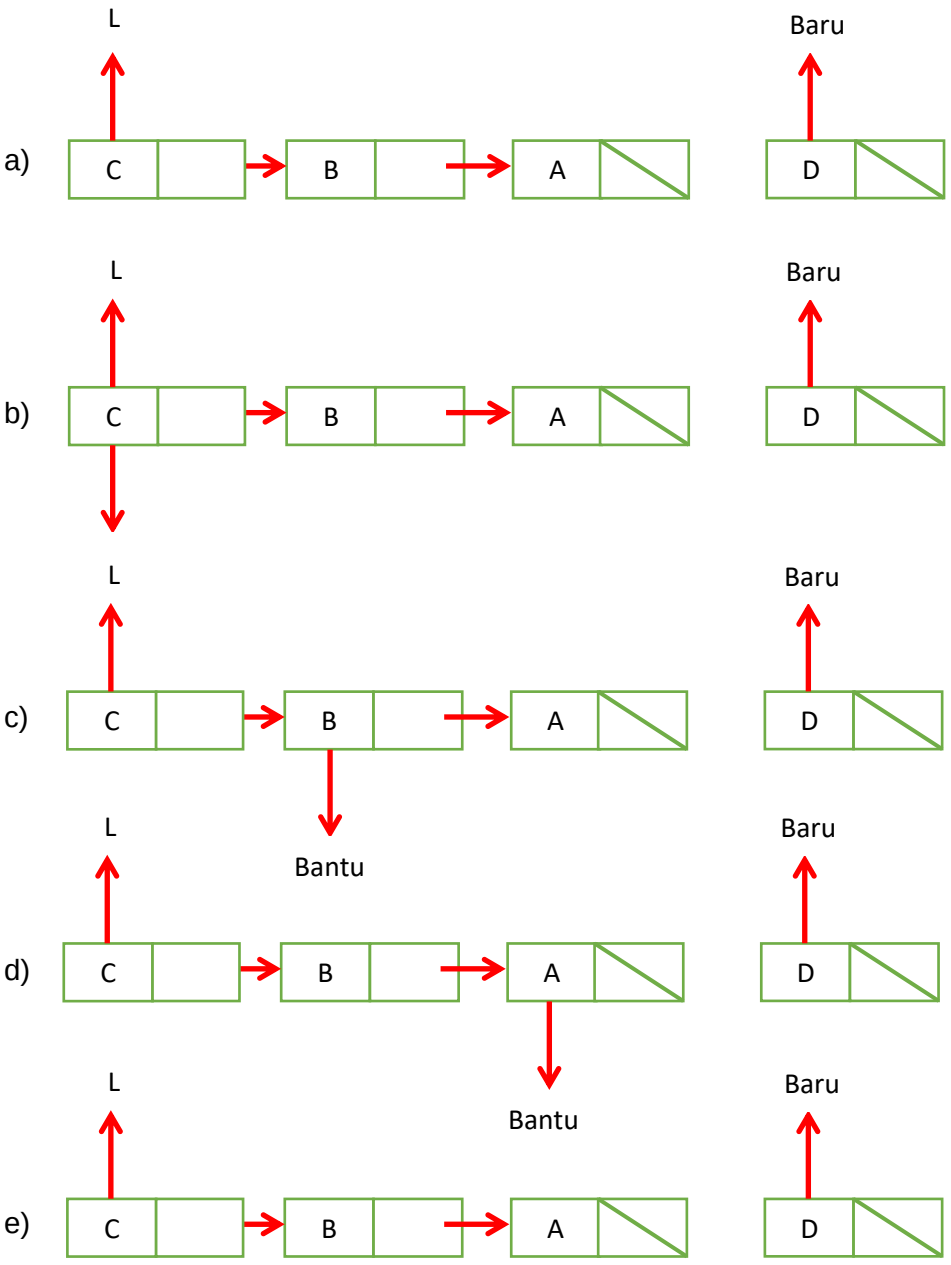



2. Menambah simpul belakang

Operasi ini akan menyisipkan simpul baru selalu berada pada posisi terakhir atau belakang dari *linked list.* Langkah – langkah penyisipan simpul belakang dapat dilakukan dengan:

- Ciptakan *linked list* belum ada maka simpul baru menjadi *linked list* (L = Baru)
- Jika *linked list* sudah ada maka penyisipan dilakukan dengan cara:
  - Buat suatu *pointer* yang dapat digerakkan, misalnya *pointer* bantu yang menunjuk simpul pertama dari *linked list* (Bantu = L). hal ini dilakukan karena pointer L tidak boleh digerakkan dari simpul depan. Karena jika pointer L digerakkan maka informasidari simpul yang ditinggalkan oleh pointer L tidak lagi dapat diakses.
  - Gerakkan pointer bantu hingga ke simpul paling belakang dari *linked list* (*while(*Bantu -> Next!=NULL) Bantu=Bantu -> Next;).
  - Sambungkan *linked list* dengan simpul baru (Bantu ->Next=Baru)

Skema penyisipan simpul belakang dapat dilihat pada gambar berikut ini:



3. Menambah simpul tengah

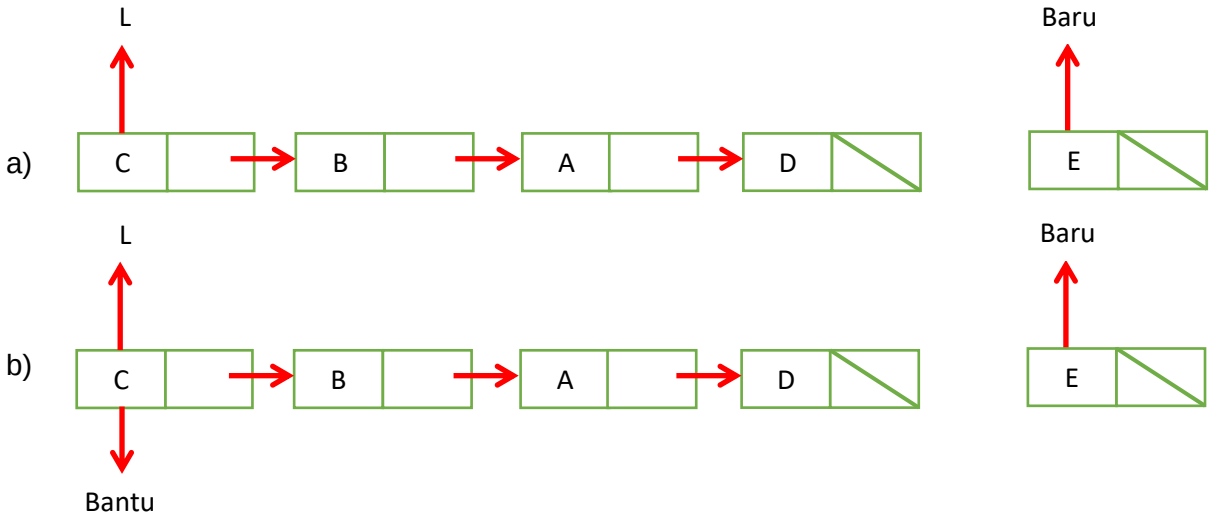
Operasi ini akan menyisipkan simpul baru selalu berada diantara dua simpul dari *linked list*. Simpul dapat diletakkan sebelum simpul tertentu atau setelah simpul tertentu. Penyisipan simpul tengah hanya dapat dilakukan jika *linked list* tidak kosong.

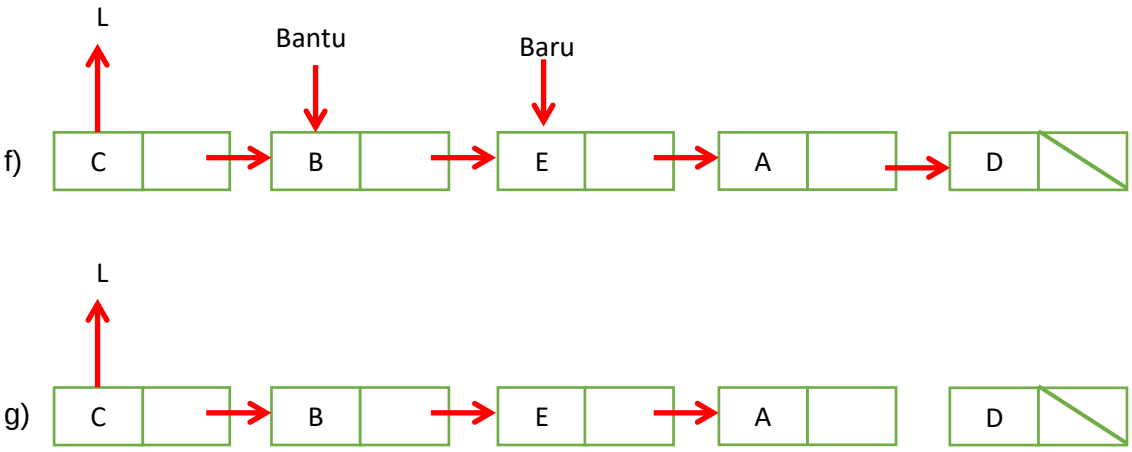
a. Menambah simpul sebelum simpul tertentu

Maksudnya adalah operasi penyisipan simpul sebelum simpul tertentu. Sebelum disisipkan simpul maka terlebih dahulu kita ketahui simpul yang dimaksud, yaitu simpul yang berisi informasi dimana simpul akan disisipkan. Misalkan *linked list* dengan 4 simpul yang masing – masing berisi informasi C, B, A dan D kemudian memilikimsimpul baru yang berisi informasi E. simpul baru akan disisipkan sebelum simpul yang berisi informasi A. adapun langkah – langkah tang dapat dilakukan adalah sebagai berikut:

- Ciptakan simpul baru yang akan disisipkan
- Buat suatu *pointer* yang dapat digerakkan, misalnya *pointer* bantu yang menunjuk simpul pertama dari *linked list* (Bantu = L). Hal ini dilakukan karena *pointer* L tidak boleh digerakkan dari simpul depan. Karena jika *pointer* L digerakkan maka informasi dari simpul yang ditinggalkan oleh pointer L tidak dapat lagi diakses.
- Gerakkan pointer bantu hingga satu simpul sebelum simpul yang berisi informasi A. (While (Bantu -> Next -> Isi!='A') Bantu=Bantu ->Next
- Sambung simpul baru dengan simpul berikutnya (Baru -> Next = Bantu -> Next)
- Sambungkan simpul bantu dengan simpul baru (Bantu -> Next = Baru)

Skema penyisipan simpul sebelum simpul tertentu seperti gambar berikut ini:



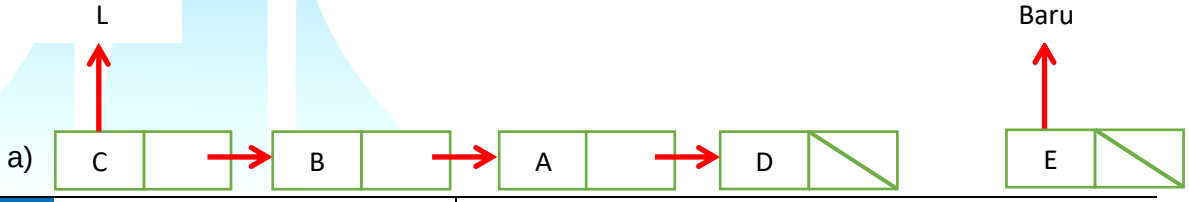



b. Menambah simpul setelah simpul tertentu

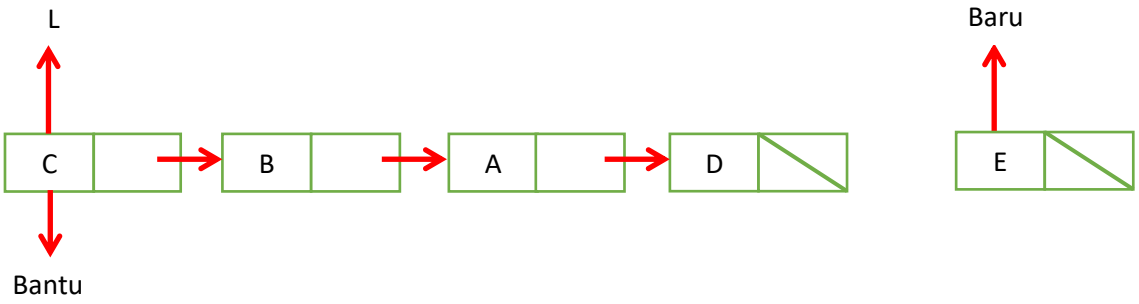
Maksudnya adalah operasi penyisipan simpul setelah simpul tertentu. Sebelum disisipkan simpul maka terlebih dahulu diketahui simpul yang dimaksud yaitu simpul yang berisi informasi dimana simpul akan disisipkan.misalkan *linked list* dengan 4 simpul yang masing – masing berisi informasi C, B, A dan D kemudian memiliki simpultambahan yang berisi informasi E. simpul baru akan disisipkan setelah simpul yang berisi informasi A. adapun langkah – langkah yang dapat dilakukan adalah sebagai berikut:

- Ciptakan simpul baru yang akan disisipkan
- Buat suatu *pointer* yang dapat digerakkan, misalnya *pointer* bantu yang menunjuk simpul pertama dari *linked list* (Bantu = L). hal ini dilakukan karena *pointer* L tidak boleh digerakkan dari simpul depan. Karena jika *pointer* L digerakkan maka informasi dari simpul yang ditinggalkan oleh *pointer* L tidak dapat diakses lagi
- Gerakkan *pointer* bantu hingga simpul yang berisi informasi A. (While(Bantu->Isi !='A')Bantu=Bantu->Next
- Sambung simpul baru dengan simpul setelah simpul yang ditunjuk oleh Bantu (Baru->Next = Bantu->Next)
- Sambungkan simpul bantu dengan simpul baru (Bantu -> Next = Baru)

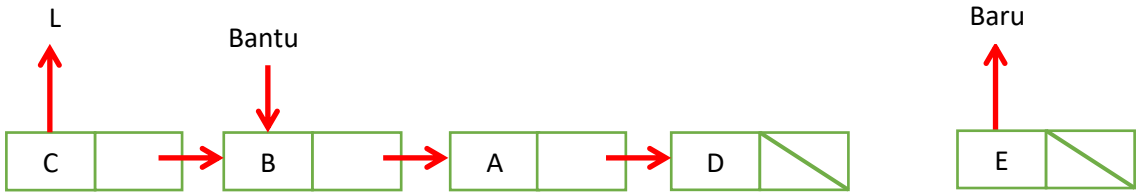
Skema penyisipan simpul sebelum simpul tertentu seperti gambar berikut ini:



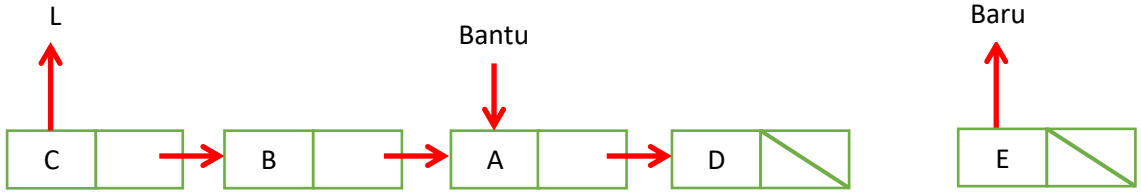
b)


L
C
B
A
D
Bantu
Baru
E

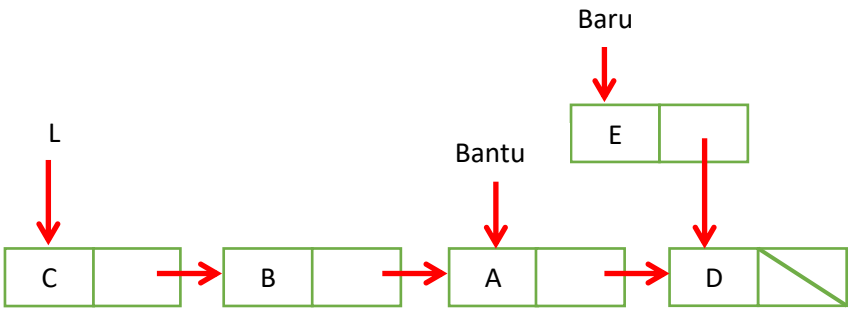
c)


L
Bantu
C
B
A
D
Baru
E

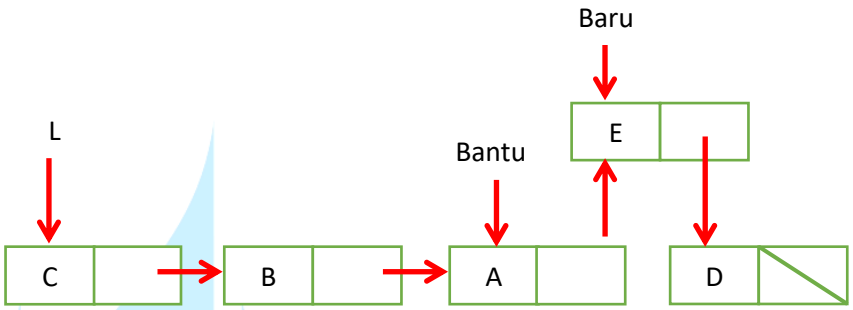
d)

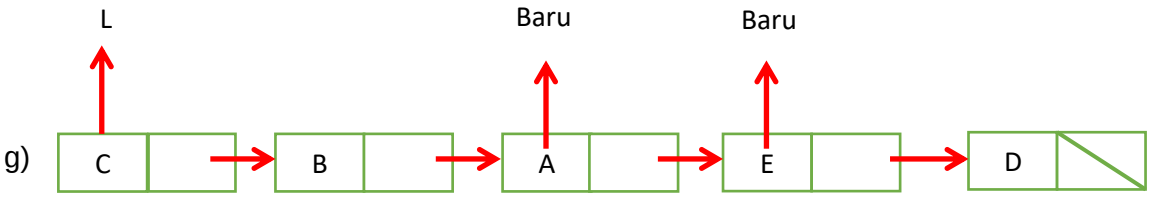

L
Bantu
C
B
A
D
Baru
E

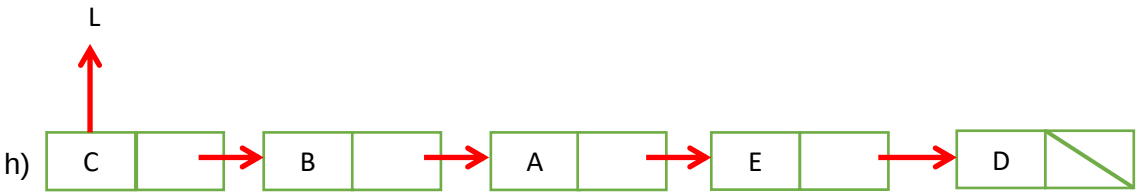
e)


Baru
E
L
Bantu
C
B
A
D

f)


Baru
E
L
Bantu
C
B
A
D




b. Menghapus simpul

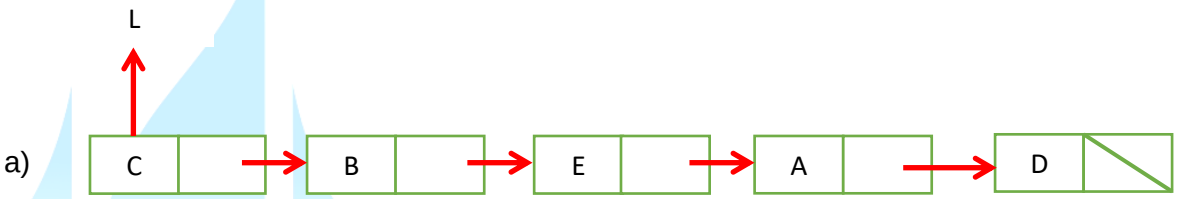
maksudnya adalah operasi menghapus suatu simpul dari *linked list*. Dalam melakukan penghapusan simpul, ada yang perlu diperhatikan, nahwa *linekd list* tidak boleh kosong dan *linked list* tidak boleh terputus. Sama halnya dengan penyisipan, penghapusan simpul juga dapat dialkukan terhadap simpul depan, simpul belakang dan simpul tengah.

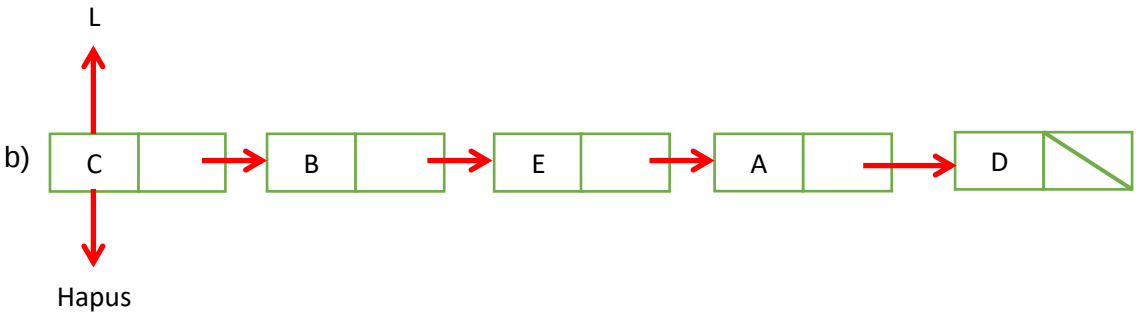
1. Menghapus simpul depan

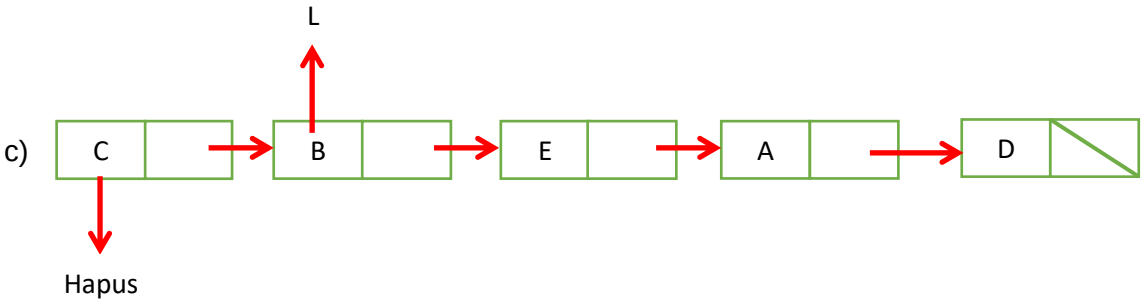
Selalu menghapus simpul depan dari *linked list. Linked list* tidak boeh kosong. Langkah – langkah penghapusan simpul depan dapat dilakukan dengan cara sebagai berikut:

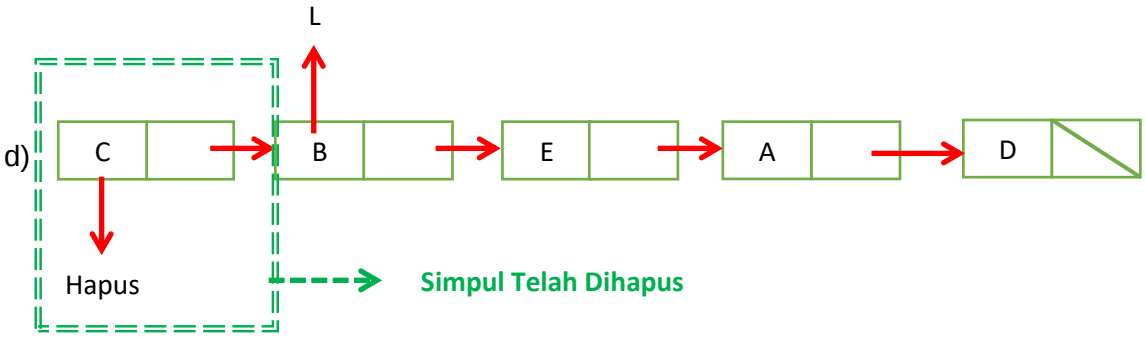
- Buat suatu *pointer* misalanya *pointer untuk* menunjuk simpul yang akan dihapus dan L untuk menujuk *linked list.*
- Simpul pertama ditunjuk oleh *pointer hapus (Hapus = L)*
- *Pointer* L digerakkan satu simpul berikutnya (L = L -> Next)
- Putuskan simpul pertama dari L (Hapus -> Next = NULL)

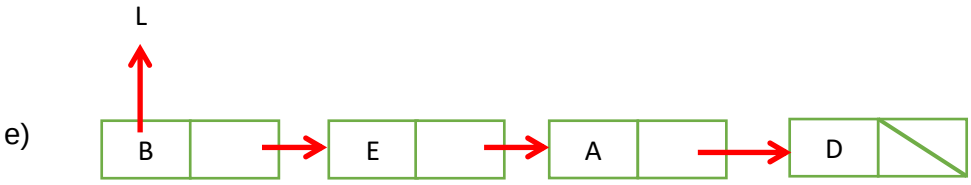
Skema peghapusan simpul depan seperti gambar berikut ini:





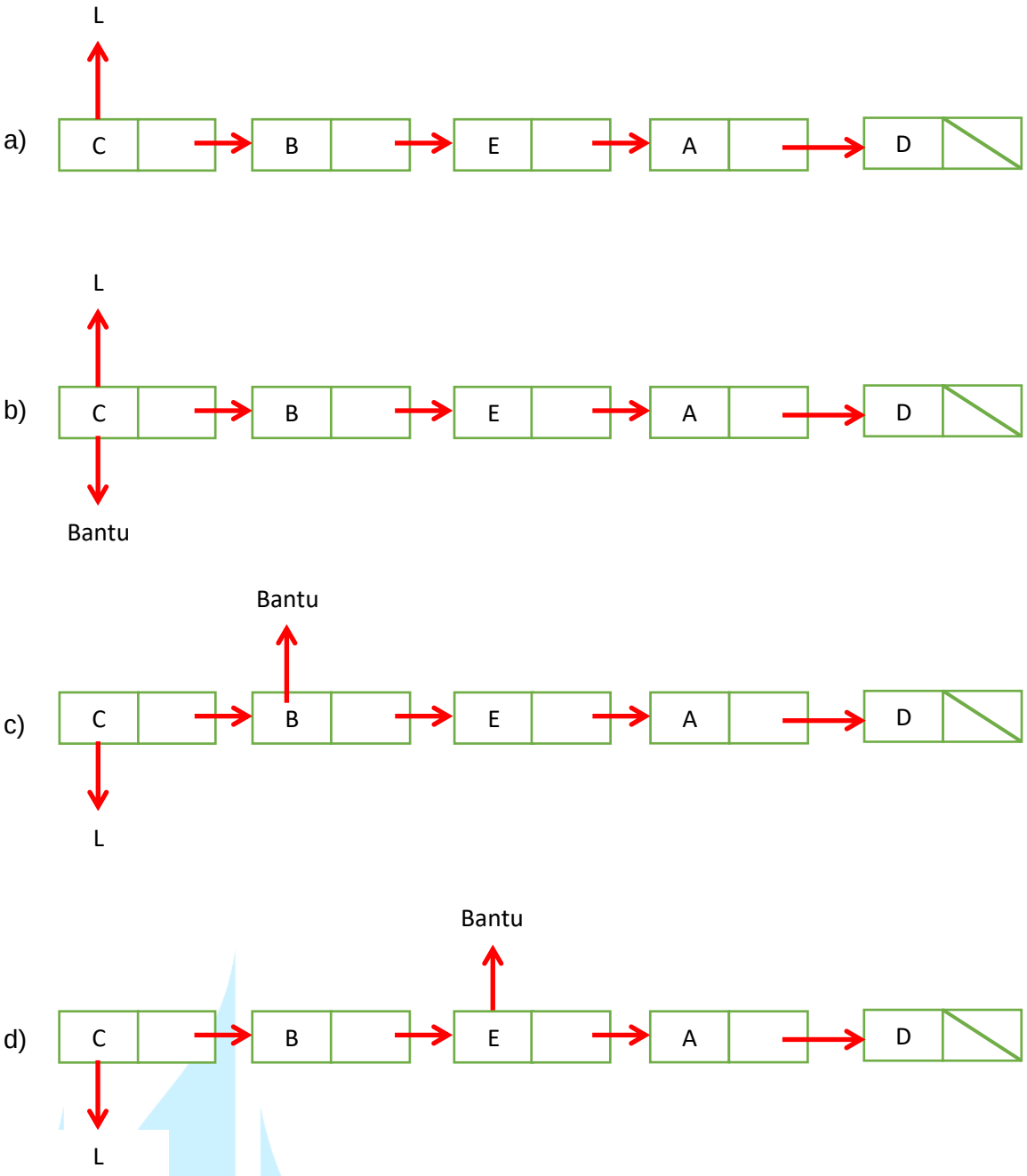




2. Menghapus simpul belakang

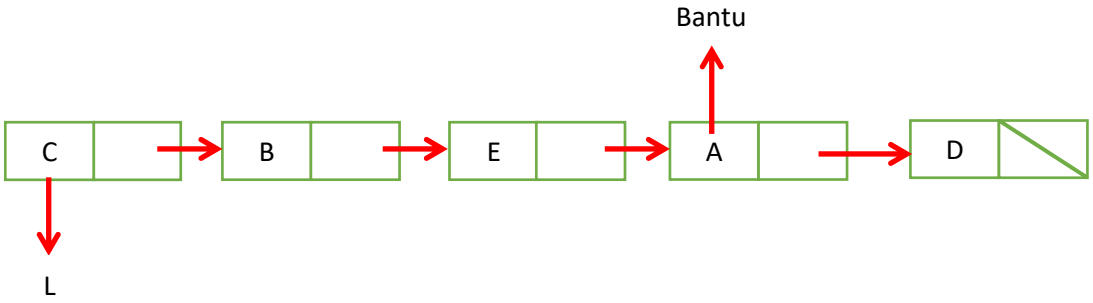
Selalu menghapussimpul terakhir atau belakang dari *linked list. Linked list* tidak boleh kosong. *Pointer* tidak boleh digerakkan, maka buat suatu *pointer* mislanya *pointer* bantu yang dapat digerakkan dalam *linked list*. Dengan menggunakan *pointer* bantu maka memeungkinkan tidak adanya simpul yang hilang atau terputus. Disamping itu juga buatlah *pointer* hapus yang digunakan untuk menunjuk simpul yang akan dihapus. Langkah – langkah penghapusan simpul belakagndapat dilakukan sebagai berikut :

- Letakkan *pointer* bantu pada simpul pertama (Bantu = L)
- Gerakkan *pointer* bantu hingga pada satu simpul sebelum seimpul terakhir
- Simpul terakhir ditunjuk oleh pointer hapus
- Putuskan simpul terakhir dari L

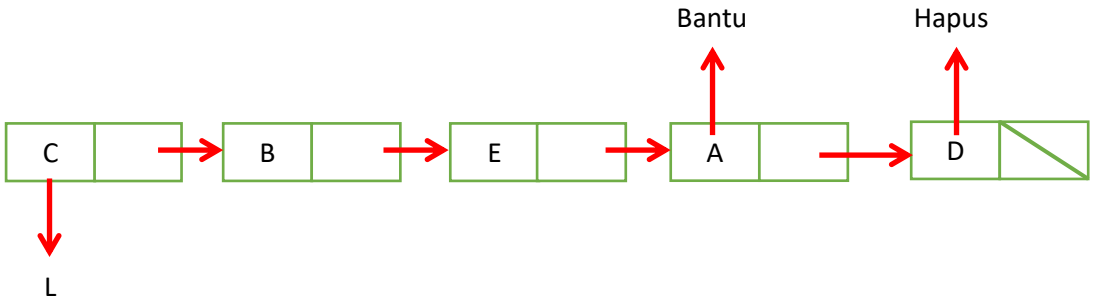
Skema peghapusan simpul belakang seperti gambar berikut ini:



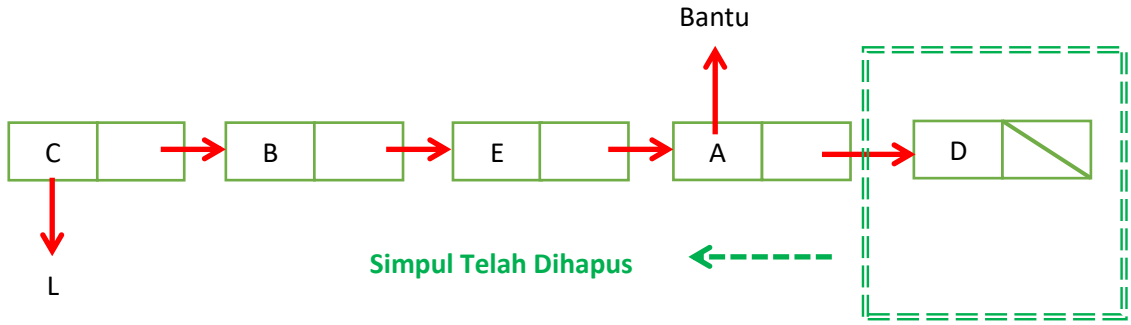
e)


Bantu
C
B
E
A
D
L

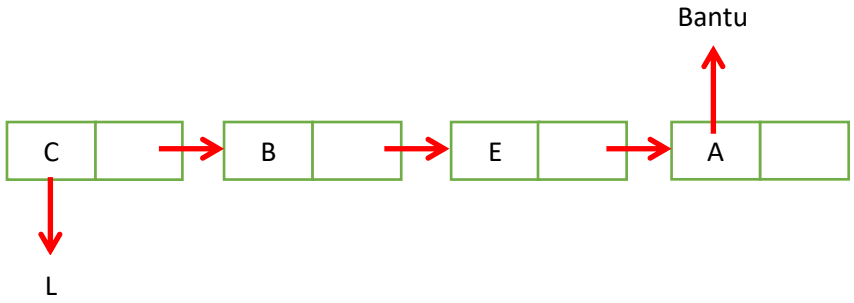
f)


Bantu
Hapus
C
B
E
A
D
L

g)


Bantu
C
B
E
A
D
L
Simpul Telah Dihapus

h)

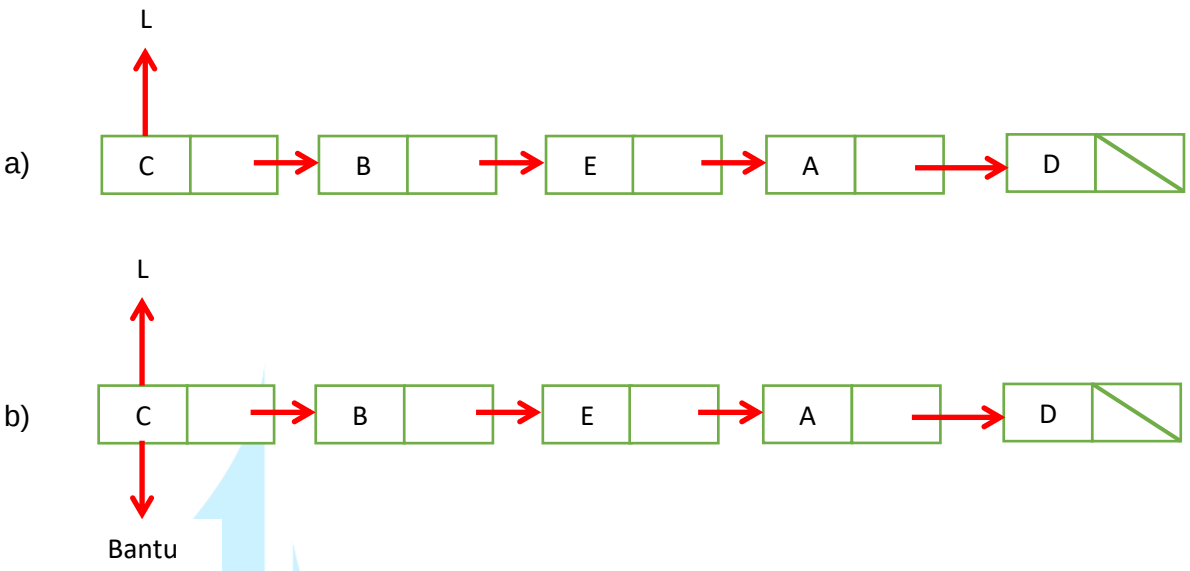

Bantu
C
B
E
A
L

3. Menghapus simpul tengah

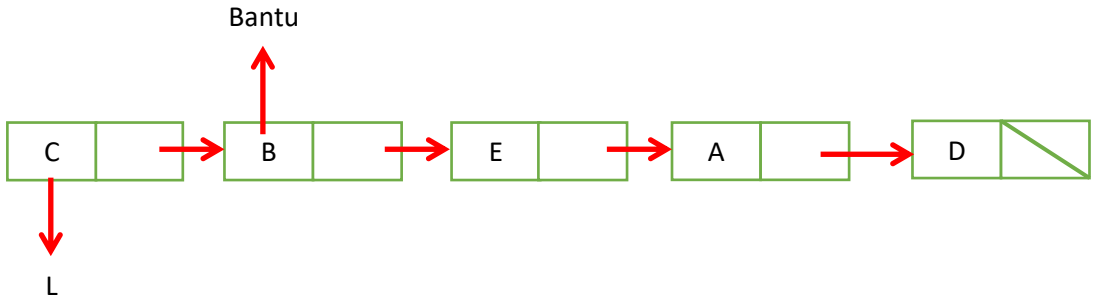
Menghapus simpul tertentu yang ada di posisi tengah dari *linked list. Linked list* tidak boleh kosong. *Pointer L* tidak boleh digerakkan, maka baut suatu *pointer* misalnya *pointer* bantu yang dapat digerakkan dalam *linked list.* Dengan menggunakan *pointer* bantu maka memungkinkan tidak adanya simpul yang hilang atau terputus. Disamping itu juga buatlah *pointer* hapus yang digunakan untuk menunjuk simpul yang akan dihapus. Langkah – langkah penghapusan simpul belakang dapat dilakukan sebagai berikut:

- Letakkan *pointer* bantu pada simpul pertama
- Gerakkan *pointer* bantu hingga satu simpul sebelum seimpul yang akan dihapus
- Letakkan *pointer* hapus pada simpul yang akan dihapus
- *Pointer* bantu menunjuk simpul setelah simpul yang ditunjuk oleh hapus
- Putuskan simpul yang ditunjuk hapus dari *linked list*

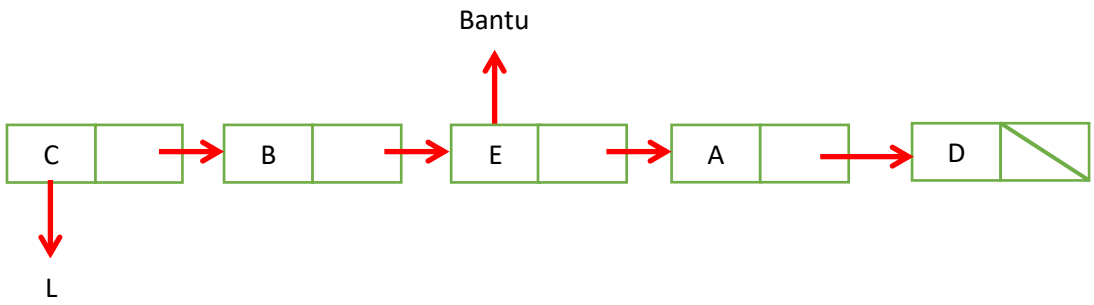
Skema peghapusan simpul tengah seperti gambar berikut ini:



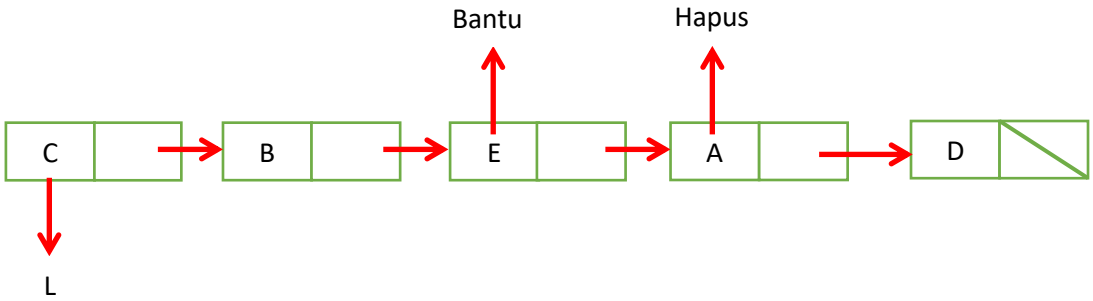
c)


Bantu
C
B
E
A
D
L

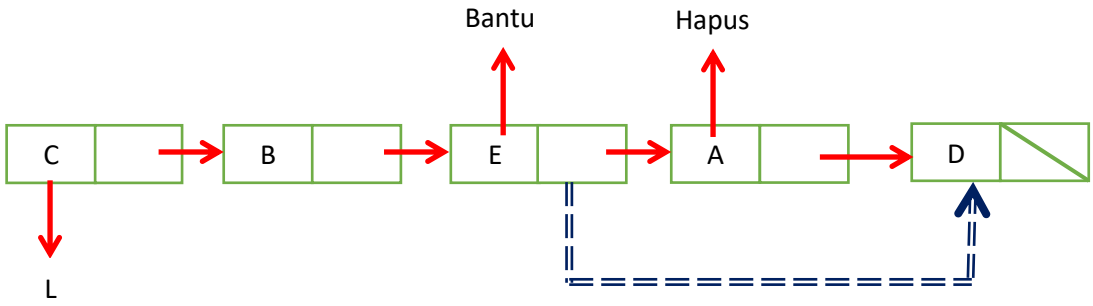
d)


Bantu
C
B
E
A
D
L

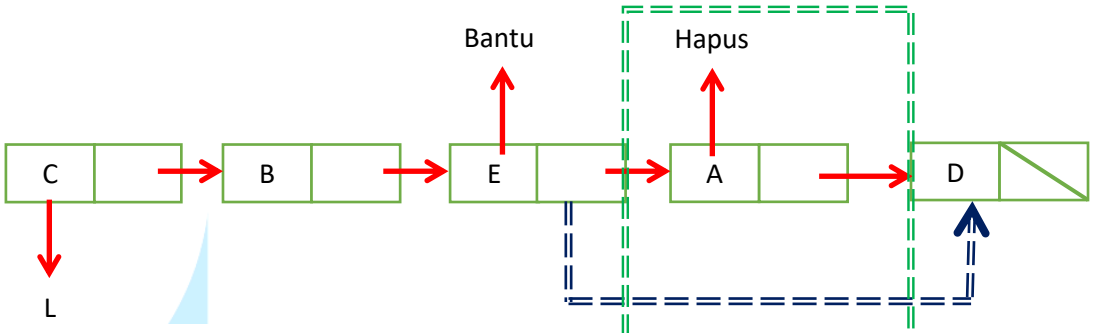
e)


Bantu
Hapus
C
B
E
A
D
L

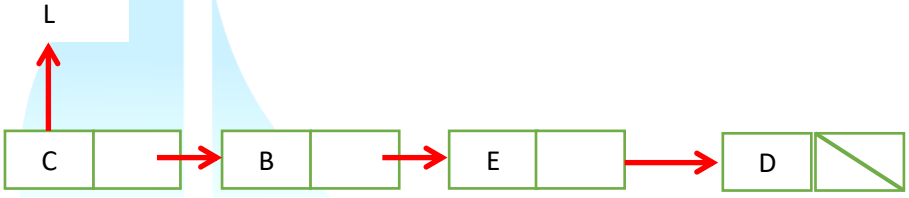
f)


Bantu
Hapus
C
B
E
A
D
L

g)


Bantu
Hapus
C
B
E
A
D
L
Simpul Telah Dihapus

h)


L
C
B
E
D

Contoh Penggunaan *Linked List*

```
#include<iostream.h>
#include<conio.h>

struct TNode{
  int data;
  TNode *next;
};
TNode *head, *tail;

int isEmpty(){
  if(head == NULL) return 1;
  else return 0;
}

void insertDepan(){
  clrscr();
  int jumlah;
  TNode *baru;
  cout<<"Berapa jumlah data yang ingin diinputkan dari depan?";
cin>>jumlah;
  for(int i=1;i<=jumlah;i++){
                baru = new TNode;
                cout<<"Data ke-"<<i<<" = "; cin>>baru->data;
                baru->next = NULL;
                if(isEmpty()==1){
                                head=tail=baru;
                                tail->next=NULL;
                }
                else {
                                baru->next = head;
                                head = baru;
                }
        }
}

void insertBelakang (){
  clrscr();
  int jumlah;
  TNode *baru;
  cout<<"Berapa jumlah data yang ingin diinputkan dari belakang?";
cin>>jumlah;
  for(int i=1;i<=jumlah;i++){
```

```
                        baru = new TNode;
                        cout<<"Data ke-"<<i<<" = "; cin>>baru->data;
                        baru->next = NULL;
        if(isEmpty()==1){
                                        head=baru;
            tail=baru;
                                        tail->next = NULL;
        }
                        else {
            tail->next = baru;
                                        tail=baru;
        }
         }
}

void tampil(){
  clrscr();
  TNode *bantu;
  bantu = head;
  if(isEmpty()==0){
                while(bantu!=NULL){
          cout<<bantu->data<<" ";
                                bantu=bantu->next;
       }
                cout<<endl;
  }
  else cout<<"Data masih kosong\n";
  getch();
}

void hapusDepan (){
  TNode *hapus;
  int d;
  int jumlah;
  clrscr();
  if (isEmpty()==0){
                        cout<<"Berapa jumlah data yang ingin anda hapus dari
depan ? "; cin>>jumlah;
        for(int i=1; i<=jumlah; i++){
                                        if(head!=tail){
                hapus = head;
                                                d = hapus->data;
                head = head->next;
```

```
                                        delete hapus;
            }
                                    else {
                d = tail->data;
                                        head=tail=NULL;
            }
                                    cout<<d<<" terhapus\n";
            if(isEmpty()==1){
                                        cout<<"Data                    sudah
kosong"<<endl;
                break;
                                     }
        }
        }
   else cout<<"Masih kosong\n";
        getch();
}

void hapusBelakang(){
  TNode *hapus,*bantu;
  int d;
  int jumlah;
  clrscr();
  if (isEmpty()==0){
     cout<<"Berapa jumlah data yang ingin anda hapus dari belakang? ";
cin>>jumlah;
                for(int i=1;i<=jumlah; i++){
          bantu = head;
                            if(head!=tail){
                                    while(bantu->next!=tail){
                                            bantu = bantu->next;
                }
                                    hapus = tail;
                tail=bantu;
                                    d = hapus->data;
                                    delete hapus;
                                    tail->next = NULL;
          }
                            else {
                d = tail->data;
                                    head=tail=NULL;
                            }
                            cout<<d<<" terhapus\n";
```

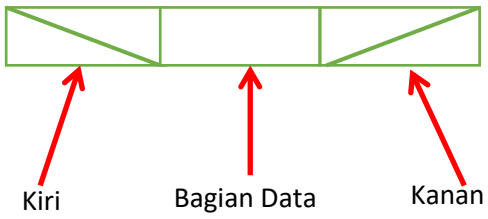
```
            if(isEmpty()==1){
                                                cout<<"Data sudah kosong"<<endl;
                break;
                                    }
                    }
  }

  else cout<<"Masih kosong\n";
  getch();
}

void clear(){
  TNode *bantu,*hapus;
  bantu = head;
  char jawab='y';
  clrscr();
  cout<<"Hapus semua data? [y/t]"; cin>>jawab;
  if(jawab=='y'||jawab=='Y'){
                while(bantu!=NULL){
                                hapus = bantu;
                                bantu = bantu->next;
                                delete hapus;
                }
                head = NULL;
                tail = NULL;
                cout<<"Semua data sudah terhapus"<<endl;
  }
  getch();
}

void main(){
  head = NULL;
  tail = NULL;
  int pilih;
  char jawab='y';
  menu:
  clrscr();
  cout<<"Menu : "<<endl;
  cout<<"1. Tampilkan data "<<endl;
  cout<<"2. Masukkan data dari depan"<<endl;
  cout<<"3. Masukkan data dari belakang "<<endl;
  cout<<"4. Hapus data paling depan "<<endl;
  cout<<"5. Hapus data paling belakang "<<endl;
```

```
        cout<<"6. Hapus semua data"<<endl;
        cout<<"7. Exit"<<endl<<endl;
        cout<<"Silahkan pilih menu : [1-7] "; cin>>pilih;

        switch(pilih){
                case 1 : tampil();
                                goto menu;
                case 2 : insertDepan();
                                goto menu;
                case 3 : insertBelakang();
                                goto menu;
                case 4 : hapusDepan();
                                goto menu;
                case 5 : hapusBelakang();
                                goto menu;
                case 6 : clear();
                                goto menu;
                case 7 : cout<<"Ingin keluar aplikasi? [y/t]"; cin>>jawab;
                                if(jawab=='y'||jawab=='Y') break;
                                else goto menu;
                default : cout<<"Pilihan Salah ! Silahkan ulangi kembali!";
                                getch();
                goto menu;
        }

}
```

# Pengenalan Double Linked List

Salah satu kelemahan dari *sinle linked list* adalah masing – masing simpul hanya memiliki satu *pointer* saja sehingga hanya dapat bergerak satu arah saja, yaitu maju atau ke kanan. Untuk mengatasi kelemahan ini maka dibuat dengan menggunakan *double linked list.*

*Double linked list* merupakan *linked list* dimana setiap simpul dibagi menjadi tiga bagian yaitu bagian isi, bagian *pointer* kiri dan bagian *pointer* kanan. Bagian isi merupakan bagian yang berisi data yang disimpan oleh simpul, sedangkan bagian *pointer* kiri merupakan bagian yang berisi alamat dari simpul sebelumnya (pada penjeleasan ini digambarkan bahwa digunakan istilah kiri dan bagian *pointer* kanan merupakan bagian yang berisi alamat dari simpul berikutnya).




# Operasi Double Linked List

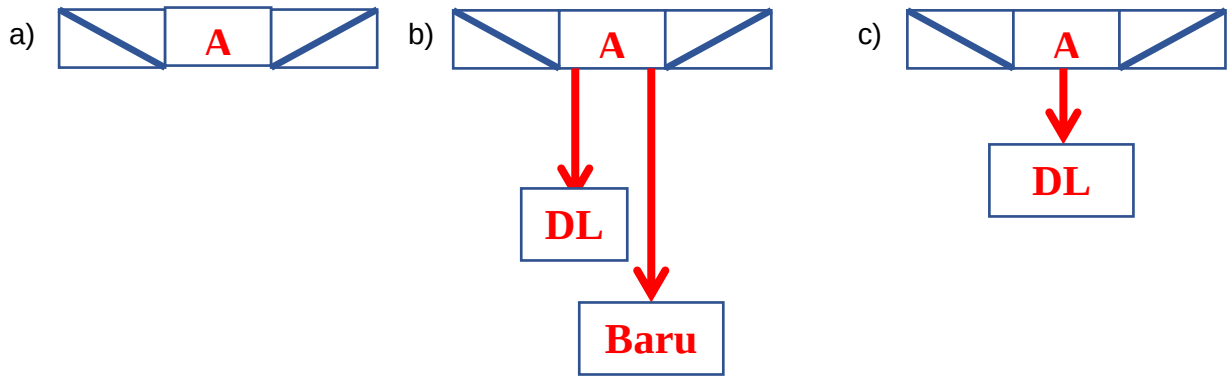
Semua operasi yang terdapat pada *double linked list* yaitu: penyisipan, penghapusan dan pencetakan.

A. Penyisipan simpul

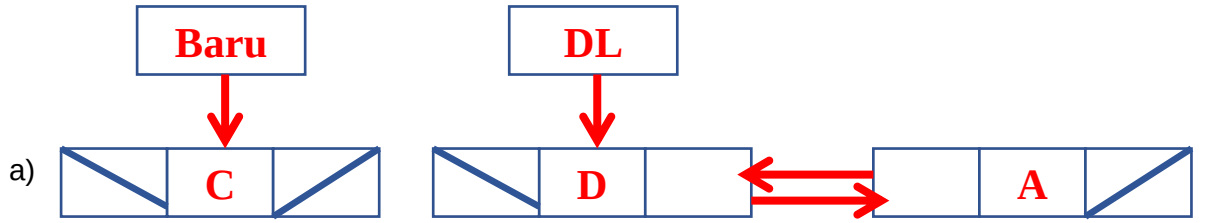
Penyisipan simpul baru selalu berada di posisi paling depan. 1.

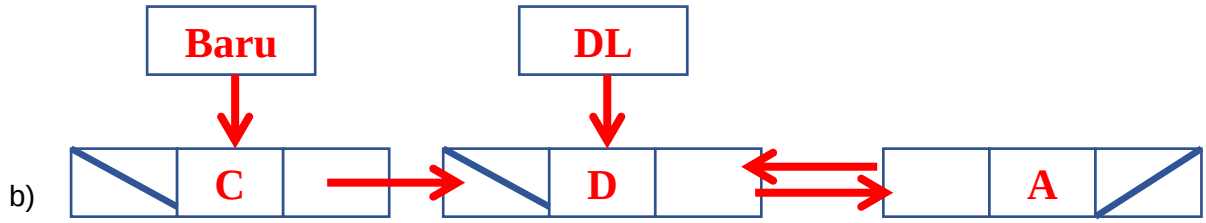
Penyisipan simpul depan dapat dilakukan dengan langkah berikut :

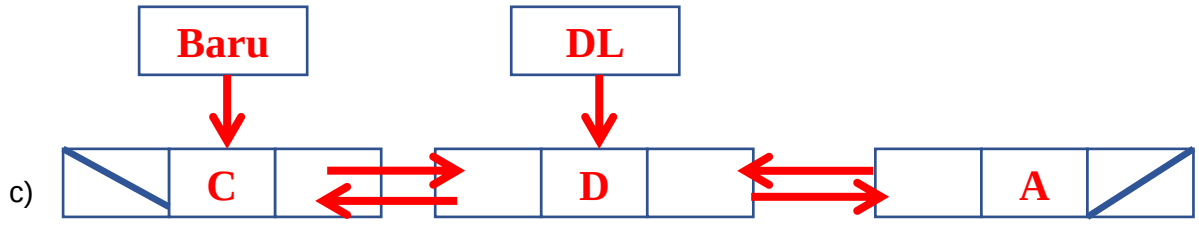
- Ciptakan simpul baru
- Jika *linked list (*DL) belum ada maka simpul baru menjadi *linked list* (DL = Baru)
- Tetapi jika *linked list* sudah ada maka penyisipan dilakukan dengan:
  - *Pointer* kanan dari baru menunjuk DL (Baru -> Kanan = DL)
  - *Pointer* kiri dari DL menunjuk baru (DL -> Kiri = Baru)
  - *Pointer* DL dipindahkan menunjuk simpul baru (DL = Baru)

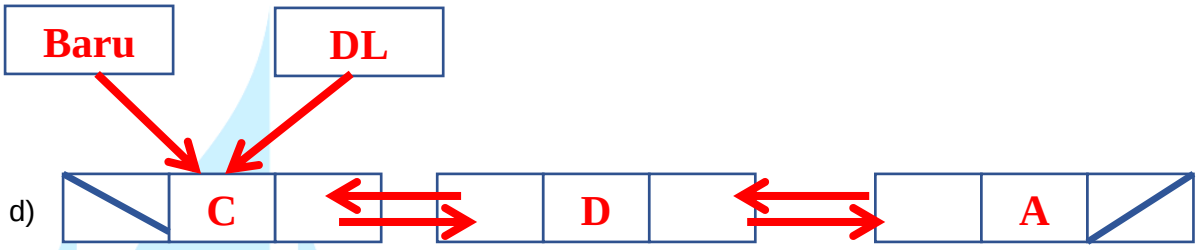



Gambar10.1 Penyisipan dengan simpul kosong

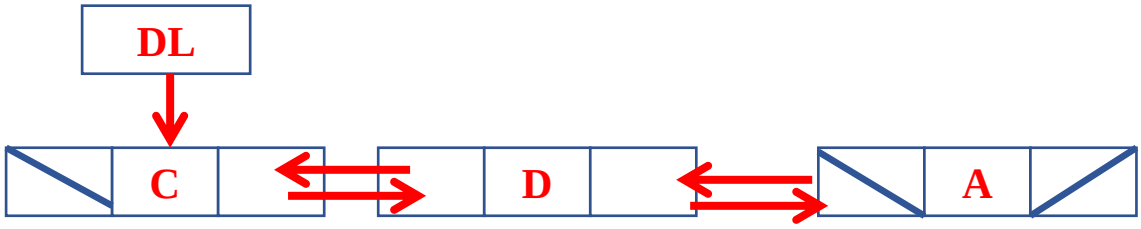


e)



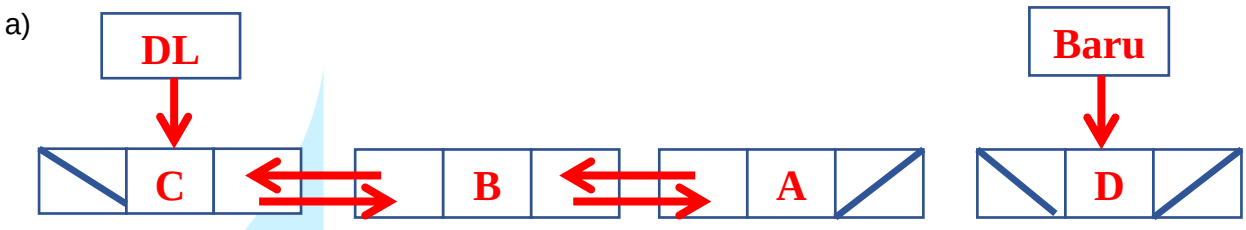

2. Penyisipan simpul belakang

Penyisipan simpul baru selalu berada diposisi paling belakang. Penyisipan simpul belakang dapat dilakukan dengan dua cara, yaitu simpul dengan menggunakan *pointer* bantu dan tanpa *pointer* bantu. Penyisipan simpul pada *double linked list* tidak harus menggunakan *pointer* bantu karena walaupun menggerakkan DL tidak mengakibatkan adanya simpul yang hilang. Hal ini terjadi karena semua simpul saling menunjuk atau saling berhubungan.

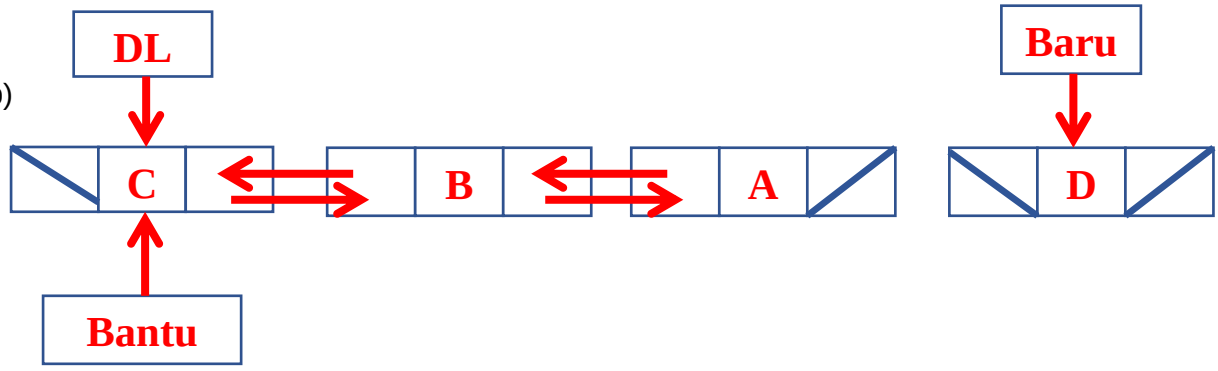
a. Penyisipan dengan pointer bantu

- Ciptakan simpul baru
- Jika *linked list* (DL) belum ada maka simpul baru menjadi *linked list* (DL = Baru)
- Tetapi jika *linked list* sudah ada maka penyisipan dilakukan dengan:
  - Buat *pointer* bantu yang dapat digerakkan (Bantu = DL)
  - Gerakkan *pointer* bantu hingga simpul terakhir
  - *Pointer* kanan simpul bantu menunjuk baru (Bantu -> Kanan = Baru)
  - Pointer kiri baru menunjuk bantu (Baru -> Kiri = Bantu)

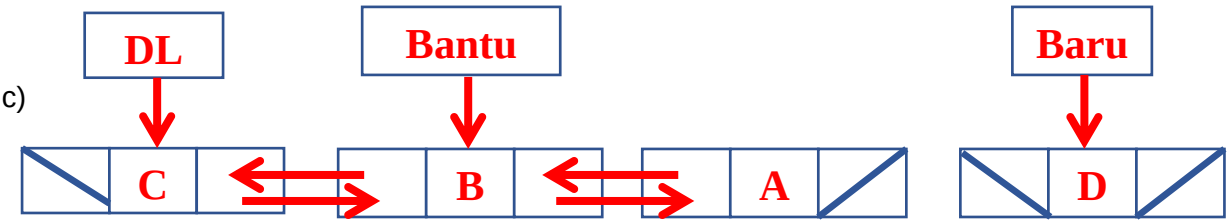
a)



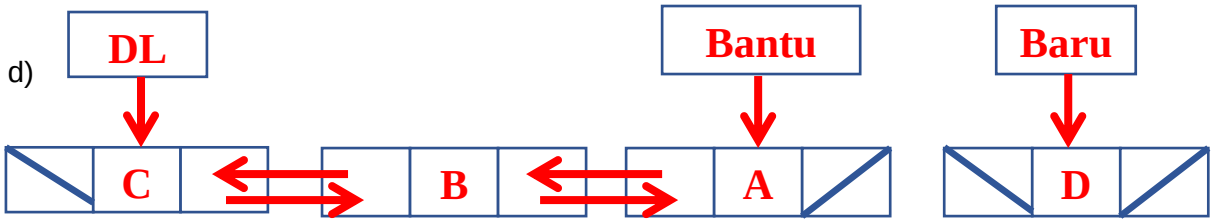
b)


DL
Baru
C
B
A
D
Bantu

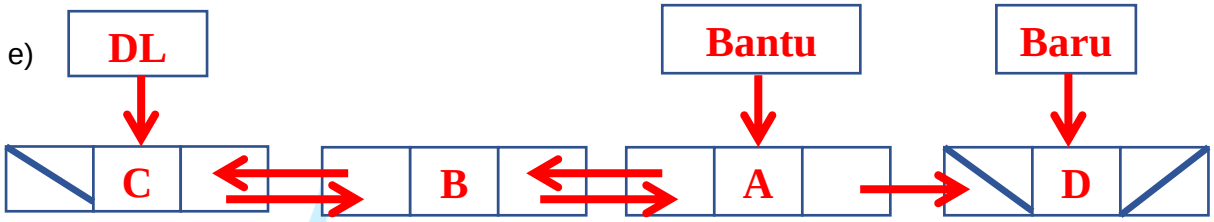
c)


DL
Bantu
Baru
C
B
A
D

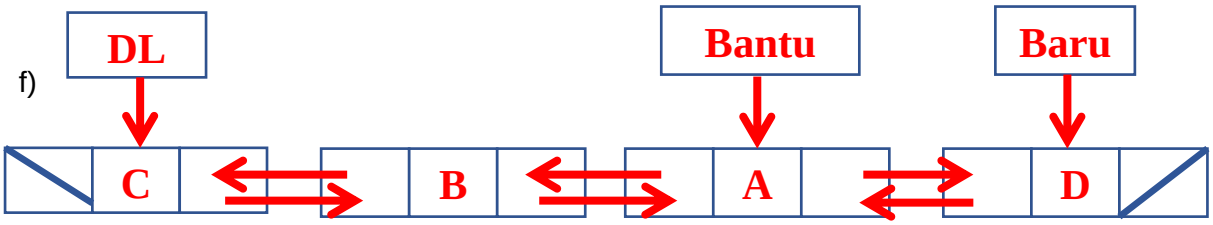
d)


DL
Bantu
Baru
C
B
A
D

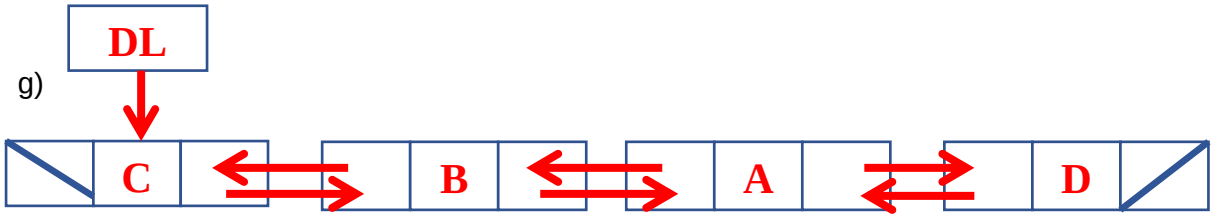
e)


DL
Bantu
Baru
C
B
A
D

f)



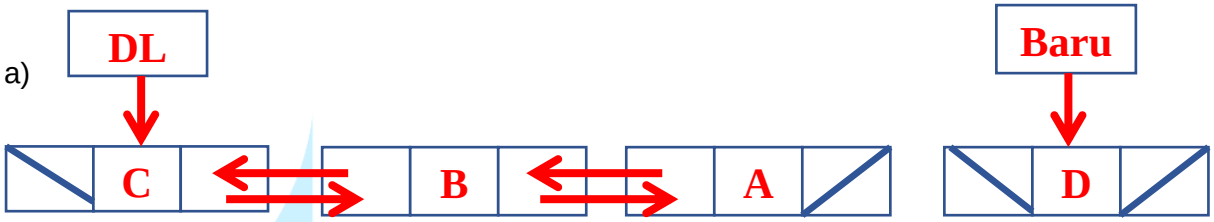

g)



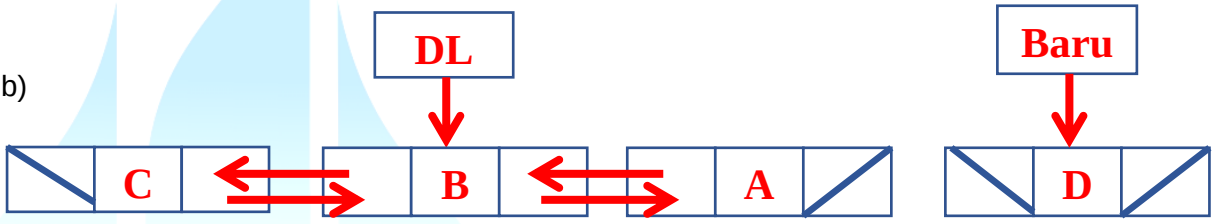

b. Penyisipan tanpa pointer bantu

- Ciptakan simpul baru
- Jika *linked list* (DL) belum ada maka simpul baru menjadi *linked list* (DL = Baru)
- Tetapi jika *linked list* (DL) sudah ada maka penyisipan dilakukan dengan:
  - Gerakkan *pointer* DL hingga simpul terakhir
  - *Pointer* kanan DL menunjuk baru
  - *Pointer* kiri baru menunjuk DL (Baru -> Kiri = DL)
  - Gerakkan *pointer* DL hingga simpul pertama

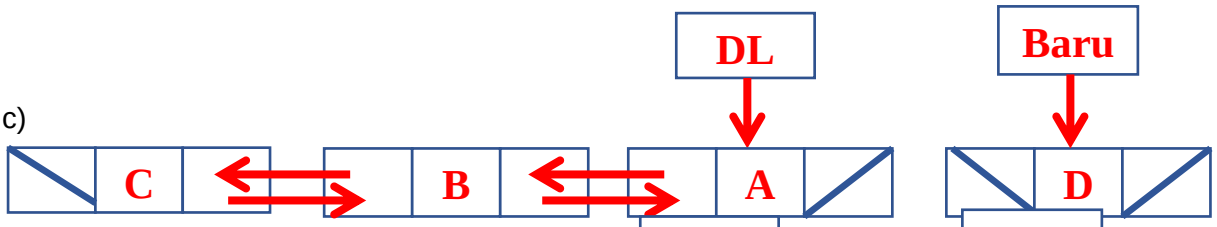
a)



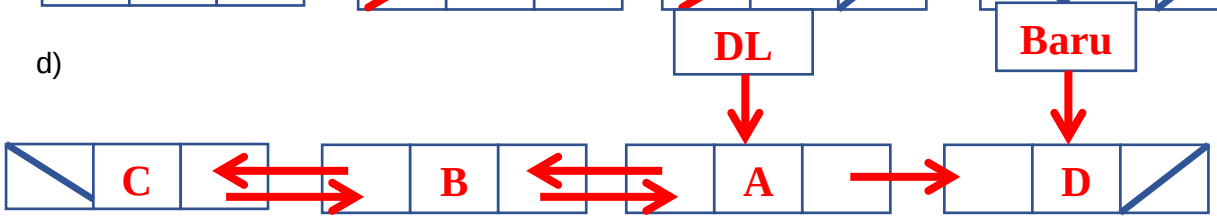

b)



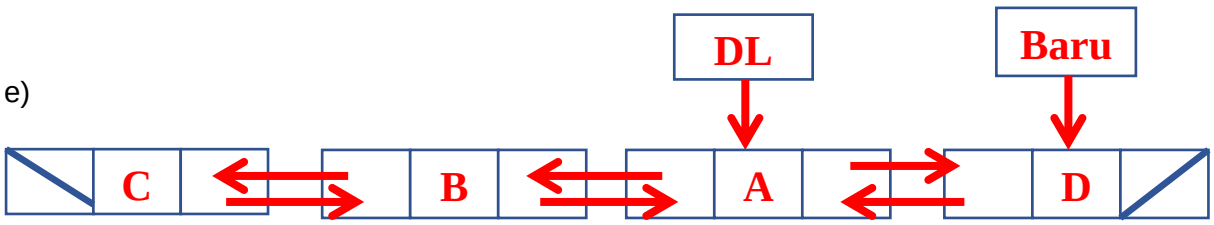
c)



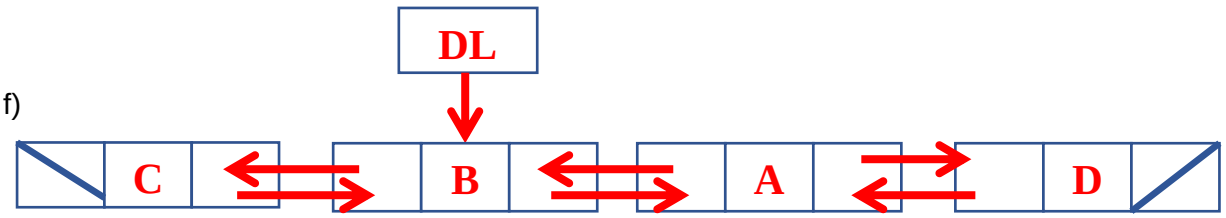

d)



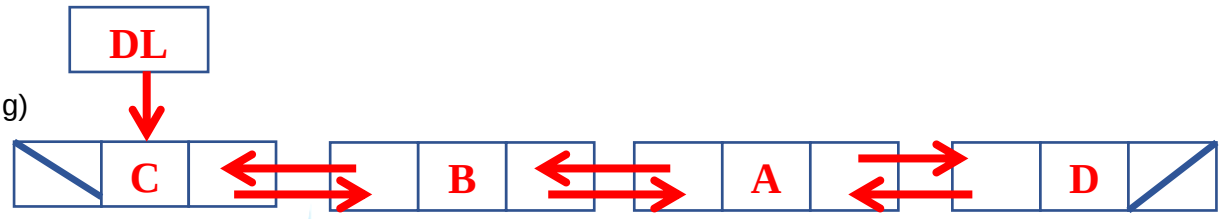

e)




f)




g)



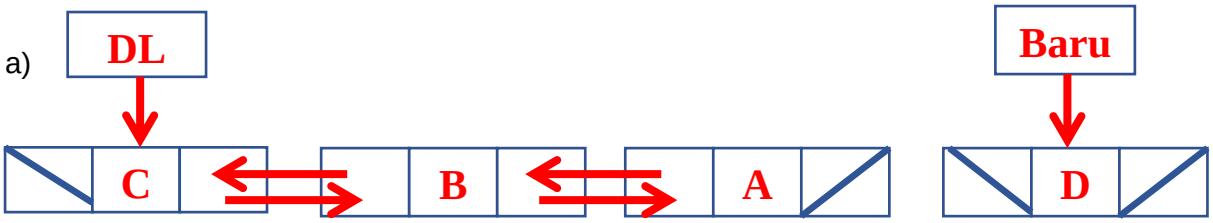

3. Penyisipan simpul tengah

Menyisipkan suatu simpul baru diantara dua simpul. Penyisipan tengah hanya dapat dilakukan jika *linked list* (DL) tidak kosong. Misalkan kita mempunyai *double linked list* (DL) dengan 3 simpul yang masing – masing berisi informasi C, B dan A serta simpul baru yang akan disisipkan ditengah yang berisi informasi D (karakter). Simpul baru akan disisipkan setelah yang berisi informasi B (elemen).

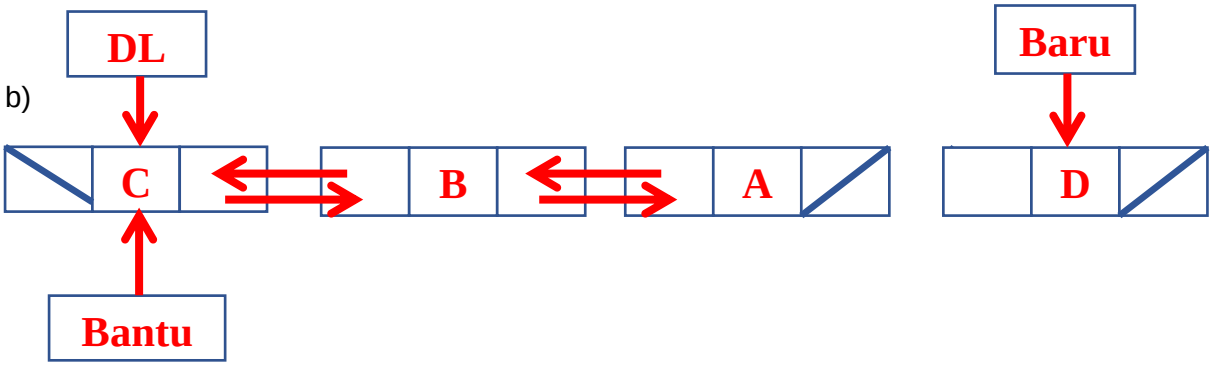
a. Penyisipan dengan pointer bantu

- Ciptakan simpul baru
- Bat *pointer* yang dapat digerakkan (Bantu = DL)
- Gerakkan *pointer* bantu hingga simpul yang berisi informasi karakter
- Kanan simpul baru menunjuk kanan bantu (Baru - > kanan = Bantu - >kanan)
- *Pointer* kiri dari kanan bantu menunjuk baru (Bantu -> Kanan -> Kiri = Baru)
- *Pointer* kanan bantu menunjuk baru (Bantu -> Kanana = Baru)
- *Pointer* baru menunjuk bantu (Baru -> Kiri = Bantu)

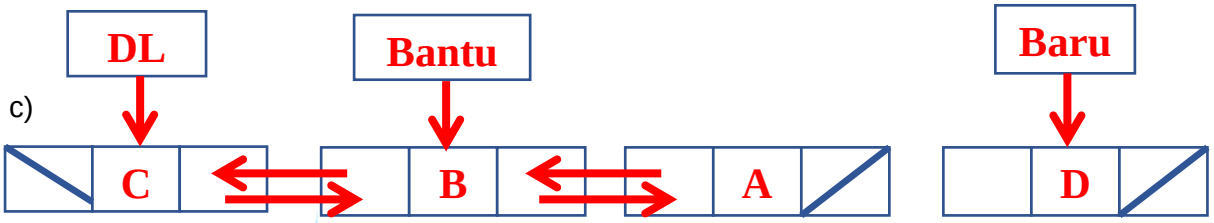
a)



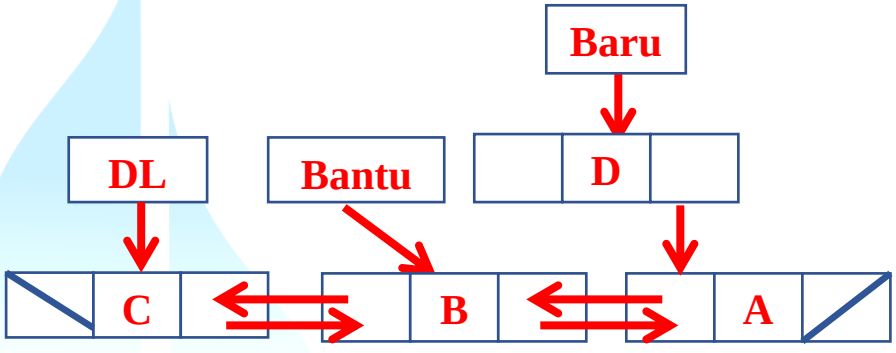

b)



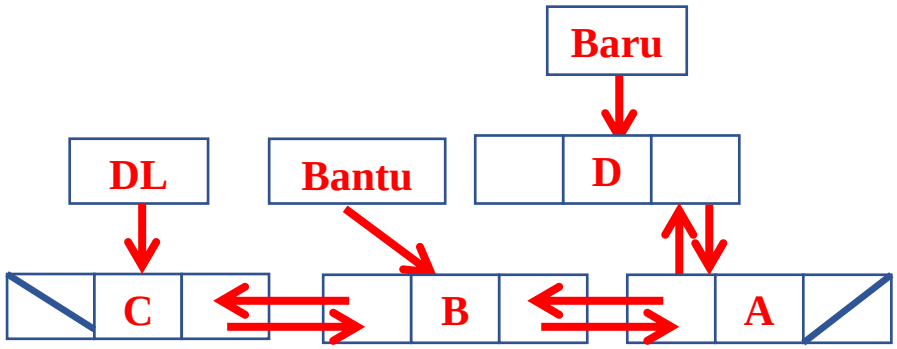

c)



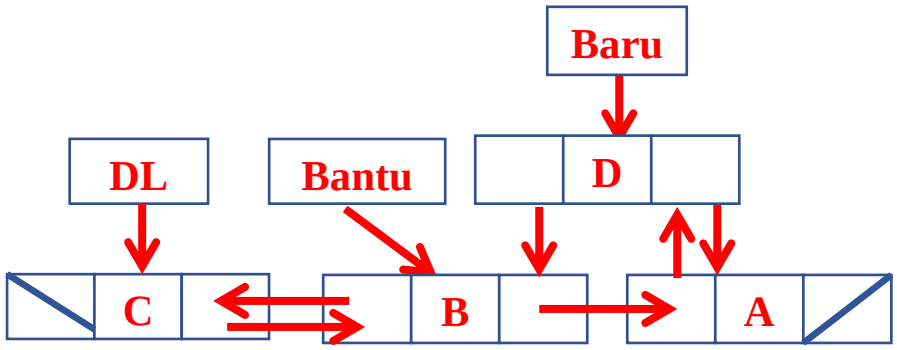

d)



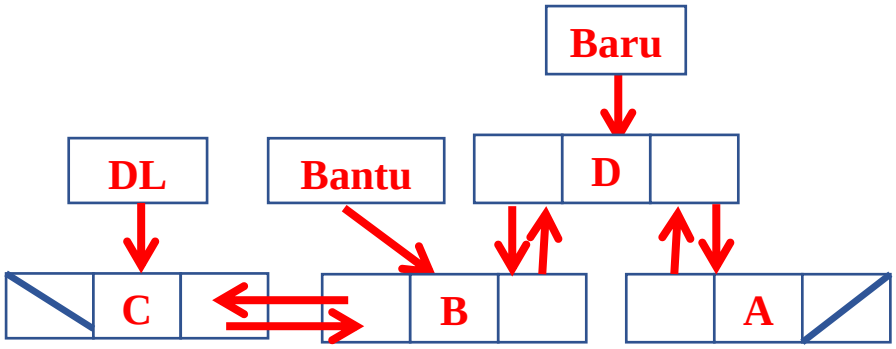
e)


Baru
DL
Bantu
D
C
B
A

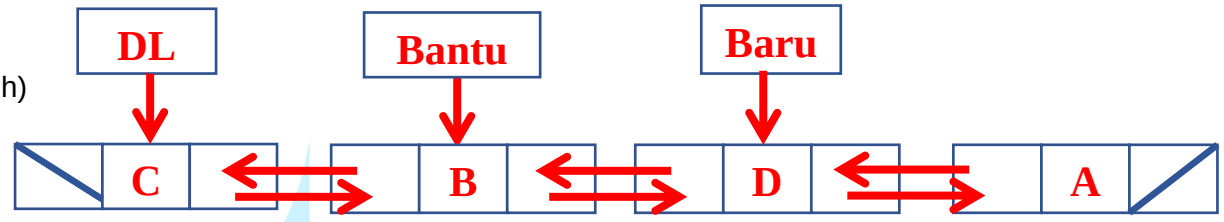
f)


Baru
DL
Bantu
D
C
B
A

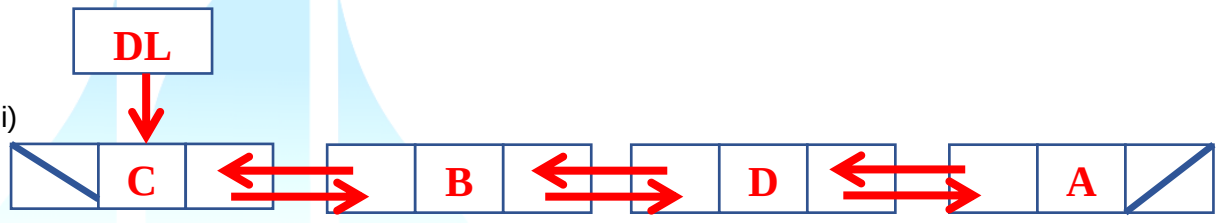
g)


Baru
DL
Bantu
D
C
B
A

h)

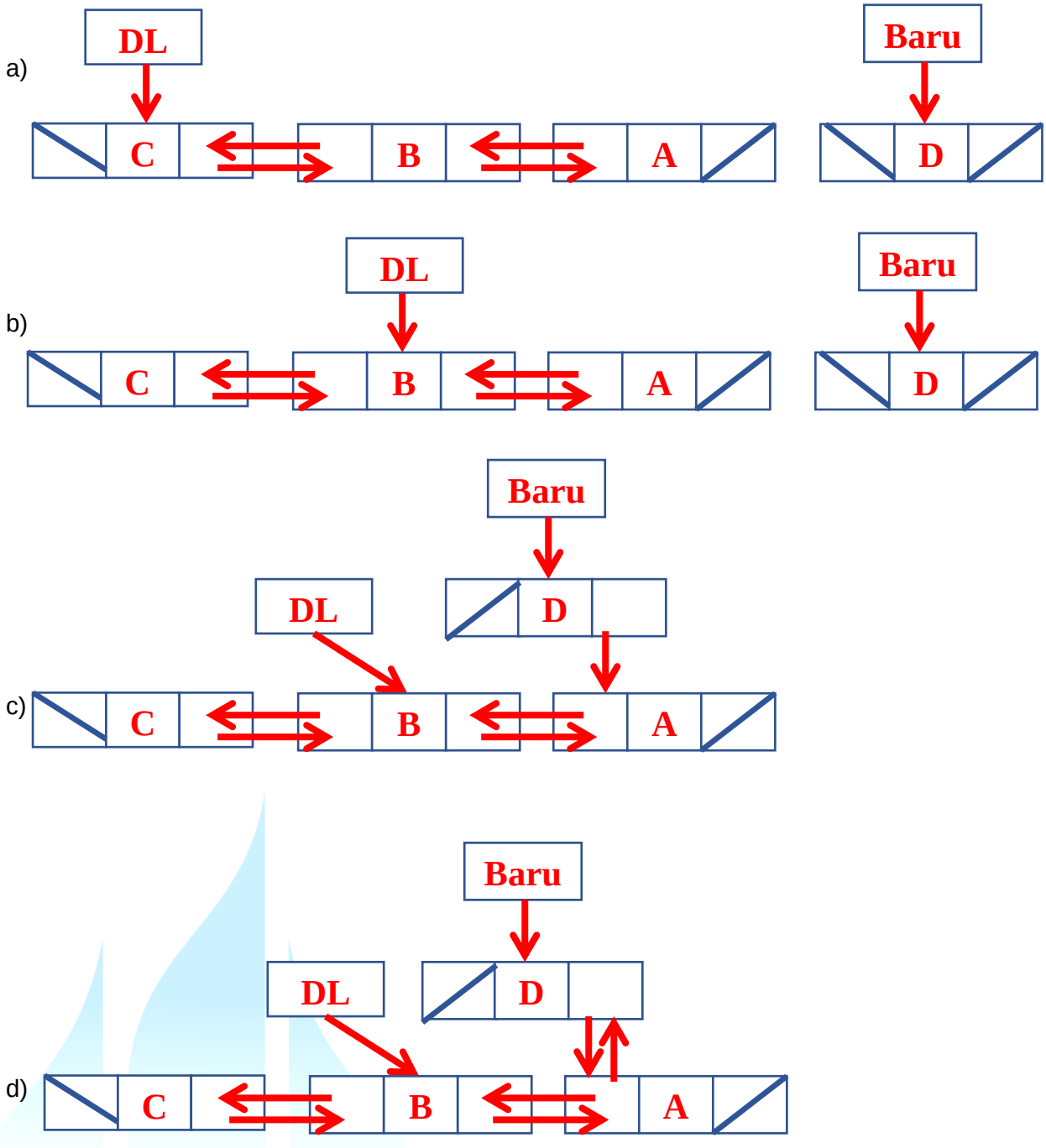

DL
Bantu
Baru
C
B
D
A

i)

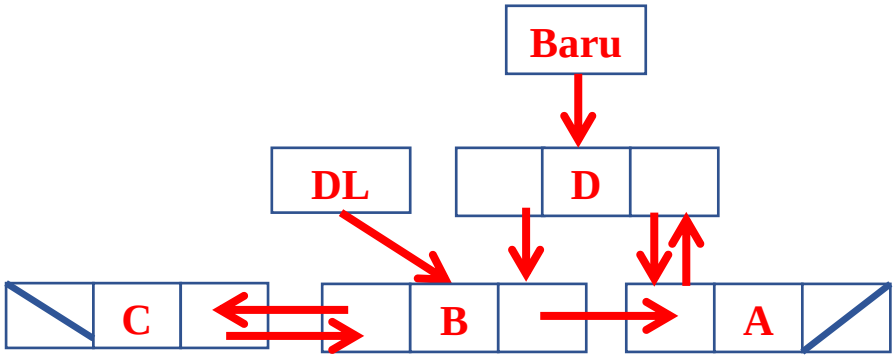

DL
C
B
D
A

b. Penyisipan tanpa pointer bantu

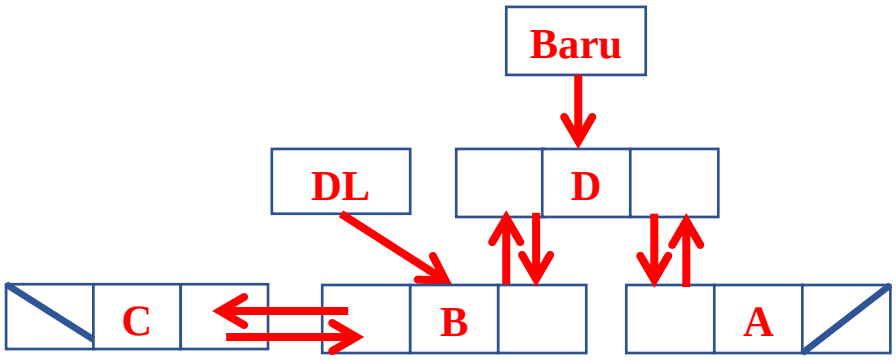
- Ciptakan simpul baru
- Gerakkan *pointer* DL hingga simpul yang berisi informasi karakter
- Kanan simpul baru menunjuk kanan DL
- *Pointer* kiri dari kanan DL menunjuk baru
- *Pointer* kanan DL menunjuk baru
- *Pointer* kiri baru menunjuk DL
- Gerakkan *pointer* DL hingga simpul pertama



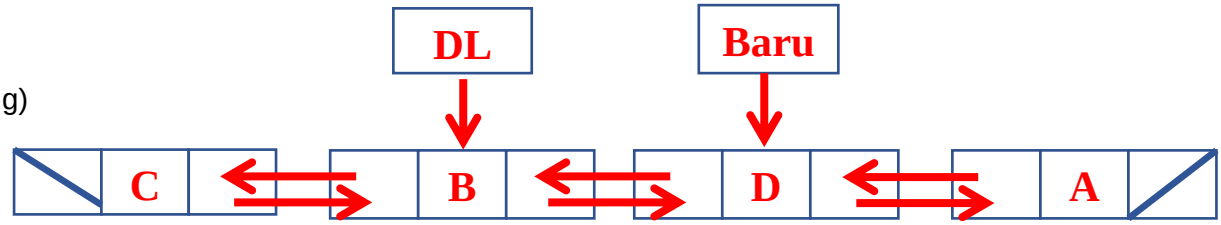
e)



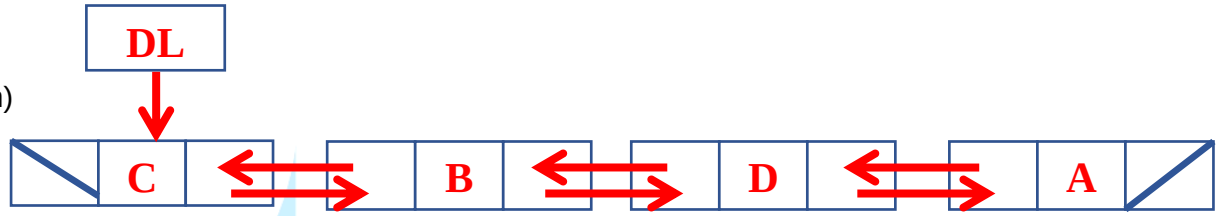

f)




g)




h)



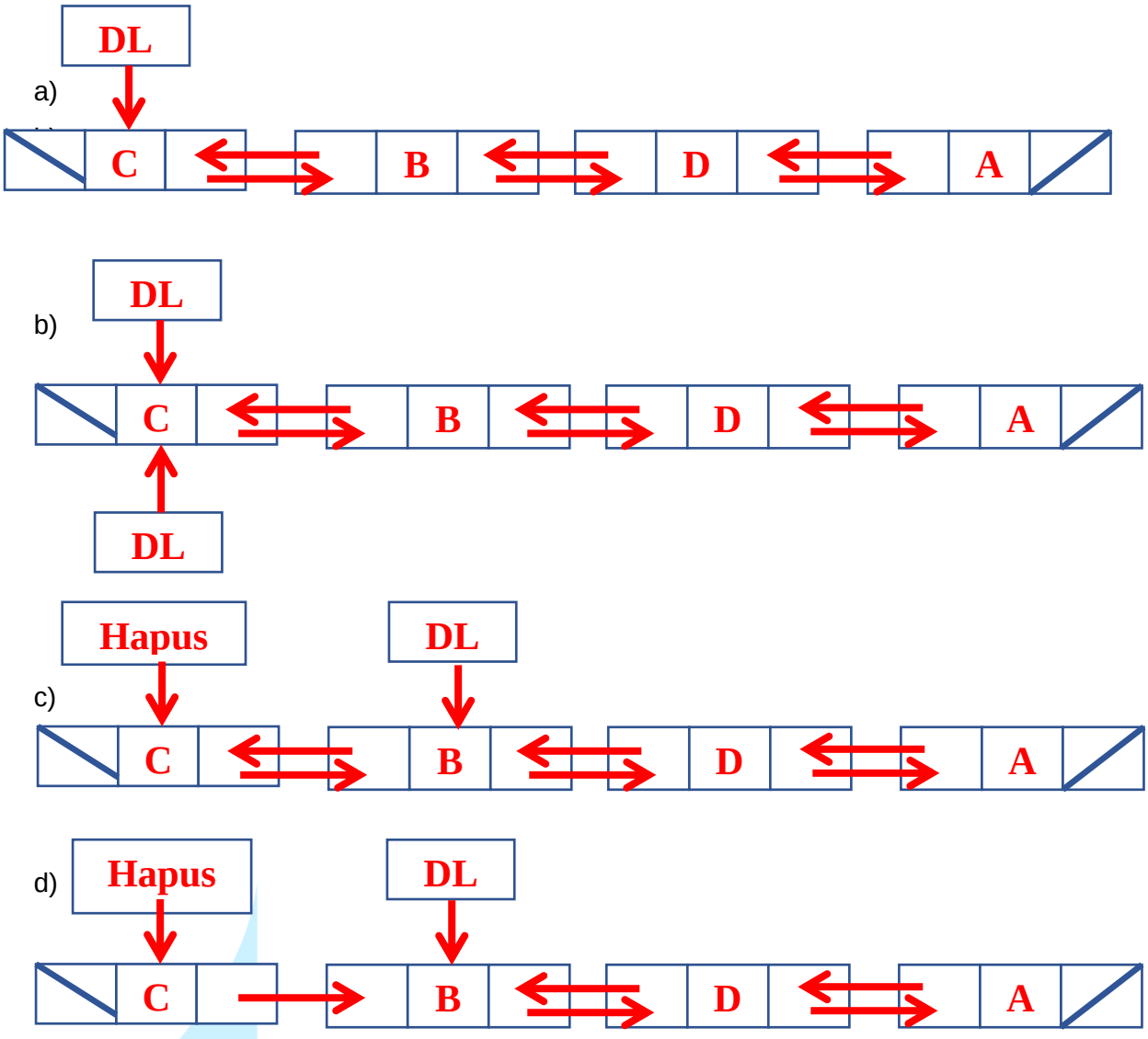

B. Penghapusan Simpul

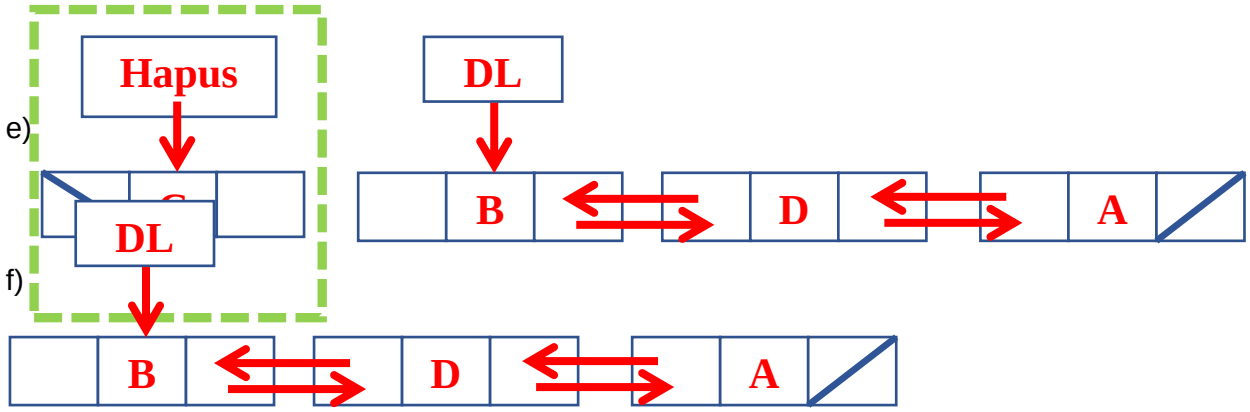
Operasi penghapusan simpul dari *linked list* pada *double linked list* hampir sama dengan penghapusan simpul pada *linked list,* yaitu *linked list* (DL) tidak boleh dalam keadaan kosong. Penghapusan simpul juga dapat dilakukan terhadap simpul depan, simpul belakang dan simpul tengah.

1. Penghapusan simpul depan

Simpul yang dihapus selalu berada pada posisi depan. Misalkan kita memiliki *linked list* DL, akan dilakukan penghapusan simpul depan yaitu simpul yang berisi informasi C. langkah – langkah penghapusan simpul depan adalah sebagai berikut:

- Simpul depan ditunjuk oleh *pointer* hapus
- *Pointer* DL digerakkan satu simpul ke kanan
- Putuskan *pointer* kiri (DL) dari hapus
- Putuskan *pointer* kanan hapus dari *linked list.*






Contoh Program

```
#include <iostream.h>
#include <conio.h>
#include <stdio.h>
#include <alloc.h>

int pil;
void pilih();
void buat_baru();
void tambah_belakang();
void tambah_depan();
void hapus_belakang();
void hapus_depan();
void tampil();



struct simpul
{
    char nim[8], nama [20];
    int umur;
    struct simpul *kiri, *kanan;
} mhs, *baru, *awal=NULL, *akhir=NULL,*hapus,*bantu;

int main()
{
    do
```

```
    {
        clrscr();
        cout<<"MENU DOUBLE LINKEDLIST"<<endl;
        cout<<"1. Tambah Depan"<<endl;
        cout<<"2. Tambah Belakang"<<endl;
        cout<<"3. Hapus Depan"<<endl;
        cout<<"4. Hapus Belakang"<<endl;
        cout<<"5. Tampilkan"<<endl;
        cout<<"6. Selesai"<<endl;
        cout<<"Pilihan Anda : ";
        cin>>pil;
        pilih();
    } while(pil!=6);
    return 0;
}

void pilih()
{
    if(pil==1)
        tambah_depan();
    else if(pil==2)
        tambah_belakang();
    else if(pil==3)
        hapus_depan();
     else if(pil==4)
        hapus_belakang();
      else if(pil==5)
        tampil();
    else
        cout<<"selesai";
}

void buat_baru()
{
    baru=(simpul*)malloc(sizeof(struct simpul));
    cout<<"input nim   : ";cin>>baru->nim;
```

```
    cout<<"input nama  : ";cin>>baru->nama;
    cout<<"input umur  : ";cin>>baru->umur;
    baru->kiri=NULL;
    baru->kanan=NULL;
}

void tambah_belakang()
{
    buat_baru();
    if(awal==NULL)
    {
        awal=baru;
        akhir=baru;
    }
    else
    {
        akhir->kanan=baru;
        baru->kiri=akhir;
        akhir=baru;
    }
    cout<<endl<<endl;
    tampil();
}

void tambah_depan()
{
    buat_baru();
    if(awal==NULL)
    {
        awal=baru;
        akhir=baru;
    }
    else
    {
        baru->kanan=awal;
        awal->kiri=baru;
```

```
        awal=baru;
    }
    cout<<endl<<endl;
    tampil();
}

void hapus_depan()
{
    if (awal==NULL)
        cout<<"Kosong";
    else if (awal->kanan==NULL)
    {
      hapus=awal;
      awal=NULL;
      akhir=NULL;
      free(hapus);
    }
    else
    {
        hapus=awal;
        awal=hapus->kanan;
        awal->kiri=NULL;
        free(hapus);
    }
    cout<<endl<<endl;
    tampil();
}

void hapus_belakang()
{
    if (awal==NULL)
        cout<<"Kosong";
    else if (awal->kanan==NULL)
    {
      hapus=awal;
      awal=NULL;
```

```
      akhir=NULL;
      free(hapus);
    }
    else
    {
     hapus=akhir;
     akhir=hapus->kiri;
     akhir->kanan=NULL;
     free(hapus);
    }
    cout<<endl<<endl;
    tampil();
}

void tampil()
{
     if (awal==NULL)
          cout<<"Kosong";
     else
     {
         bantu=awal;
         while(bantu!=NULL)
         {
           cout<<"nim : "<<bantu->nim;
           cout<<"  nama : "<<bantu->nama;
           cout<<"  umur : "<<bantu->umur<<endl;
           bantu=bantu->kanan;
         }
     }
     getch();
}
```

Tugas:

1. Buatlah sebuah program untuk menghitung nilai mahasiswa dengan menggunakan **structure** dan menambahkan Grade dengan kondisi:

   A : 80 – 100

   B+ : 70 – 79

B : 60 – 69

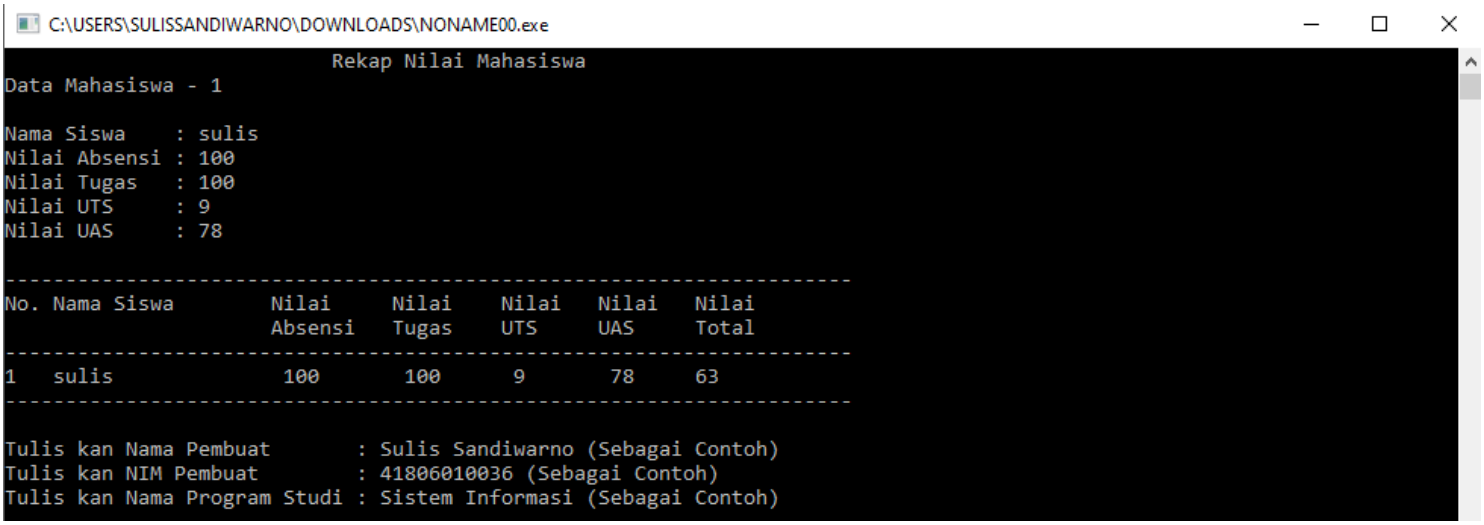
C+ : 50 – 59

C : 40 – 49

D : 30 – 39

E : 0 – 29

Dengan output seperti berikut:




2. Buatlah sebuah program **linked list** dengan menambahkan fungsi **input dan hapus data tengah**
3. Buatlah sebuah program **double linked list** dengan menambahkan fungsi **input dan hapus data tengah**

Tugas ini dikumpulkan pada pertemuan minggu depan pada link POST, dikerjakan secara kelompok

# Daftar Pustaka


1. Reema Thareja, Data Structures Using C, Oxford University Press, 2017
2. R.M.Z. Trigo, Data Structures, Automation & Problem Solving, Algorithms Series, 2017
3. Hemant Jain, Problem Solving in Data Structures & Algorithms Using C, Hemant Jain Publisher, 2017
4. Narasimha Karumanchi, Data Structures and Algorithms Made Easy, CareerMonk Publications, 2017
5. Stephen R. Davis, C++ For Dummies®, 7th Edition, John Wiley & Sons, Inc, 2017
6. S.K Srivasta and Deepali Srivasta, Data Structures Through C in Depth, BPB Publications, 2018
7. Timothy Masters, Data Mining Algorithms in C++ Data Patterns and Algorithms for Modern Applications, Timothy Masters, 2018
8. Michael T. Goodrich, Roberto Tamassia, and David M. Mount, Data Structures and Algorithms in C++, John Wiley & Sons, Inc, 2018