ALGORITMA DAN STRUKTUR DATA

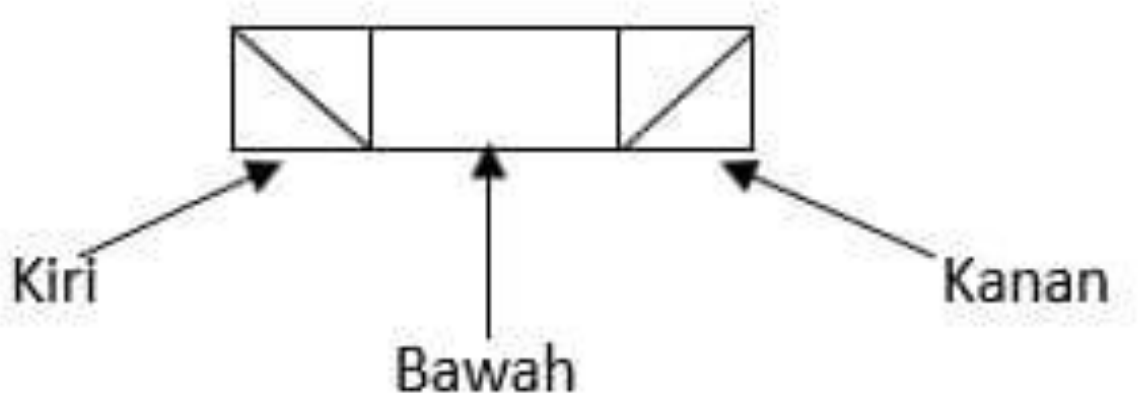
Nama : Rekaliana

NPM : G1F021006

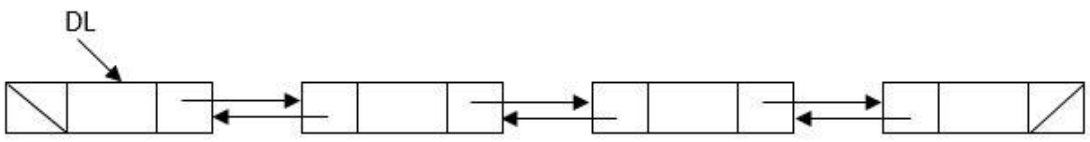
Ringkasan materi Double Link List

Pengertian Double Linked List pada C++. Salah satu kelemahan dari Singly 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 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 simpulan sebelumnya dan bagian kanan merupakan bagian yang berisi alamat dari simpul berikutnya. Gambar 1. merupakan simpul untuk Double Linked List.



Gambar 1. Simpul pada Double Link List



Gambar 2. Doubly Linked List DL dengan 4 Simpul

Dari gambar di atas dapat kita lihat bahwa pointer kanan suatu simpul menunjuk ke simpul berikutnya dan pointere kiri menunjuk simpul sebelumnya. Tetapi untuk pointer kiri simpul pertama tidak menunjuk ke simpul yang lain (NULL) dan pointer kanan dari simpul terakhir juga tidak menunjuk simpul yang lain (NULL).

1. Deklarasi Doubly Linked List

```
typedef struct node *simpul;

struct node

     {

            char Isi;

            simpul kanan;

            simpul kiri;

     };
```

2. Operasi Pada Doubly Linked List

Semua operasi pada Singly Linked List juga terdapat pada operasi Doubly Linked List, yaitu Penyisipan, Penghapusan, dan Pencetakan.

2.1. Penyisipan Simpul

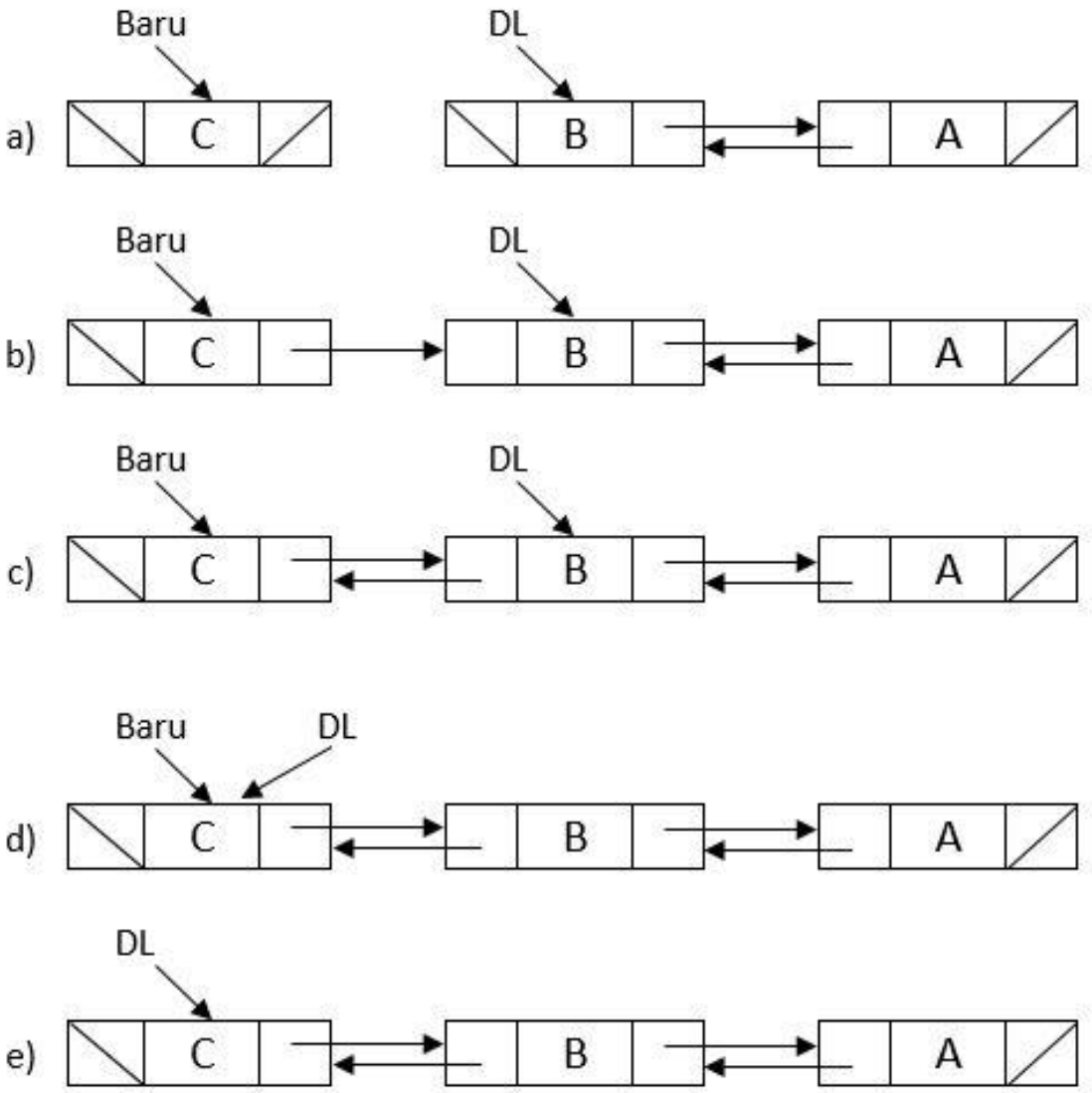
Penyisipa simpul adalah operasi penyisipan suatu simpul baru ke dalam suatu Doubly Linked List. Penyisipan dapat dilakukan di posisi depan, tengah, dan belakang.

1. Penyisipan Simpul Depan

Penyisipan simpul baru selalu berada di posisi paling depan. 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 (DL) 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).



Gambar 3. Penyisipan Simpul Depan dengan DL. Tidak Kosong

Fungsi penyisipan simpul depan dengan mengikuti langkah-langkah dan gambar 3 di atas dapat dilihat berikut ini.

```
void Sisip_Depan(simpul &DL, char elemen)
{
simpul baru;
baru = (simpul) malloc(sizeof(simpul));
```

```
baru->Isi = elemen;
baru->kanan = NULL;
baru->kiri = NULL;
if(DL == NULL)
DL=baru;
else
{
baru->kanan = DL;
DL->kiri = baru;
DL=baru;
}
}
```

2. Penyisipan Simpul Belakang

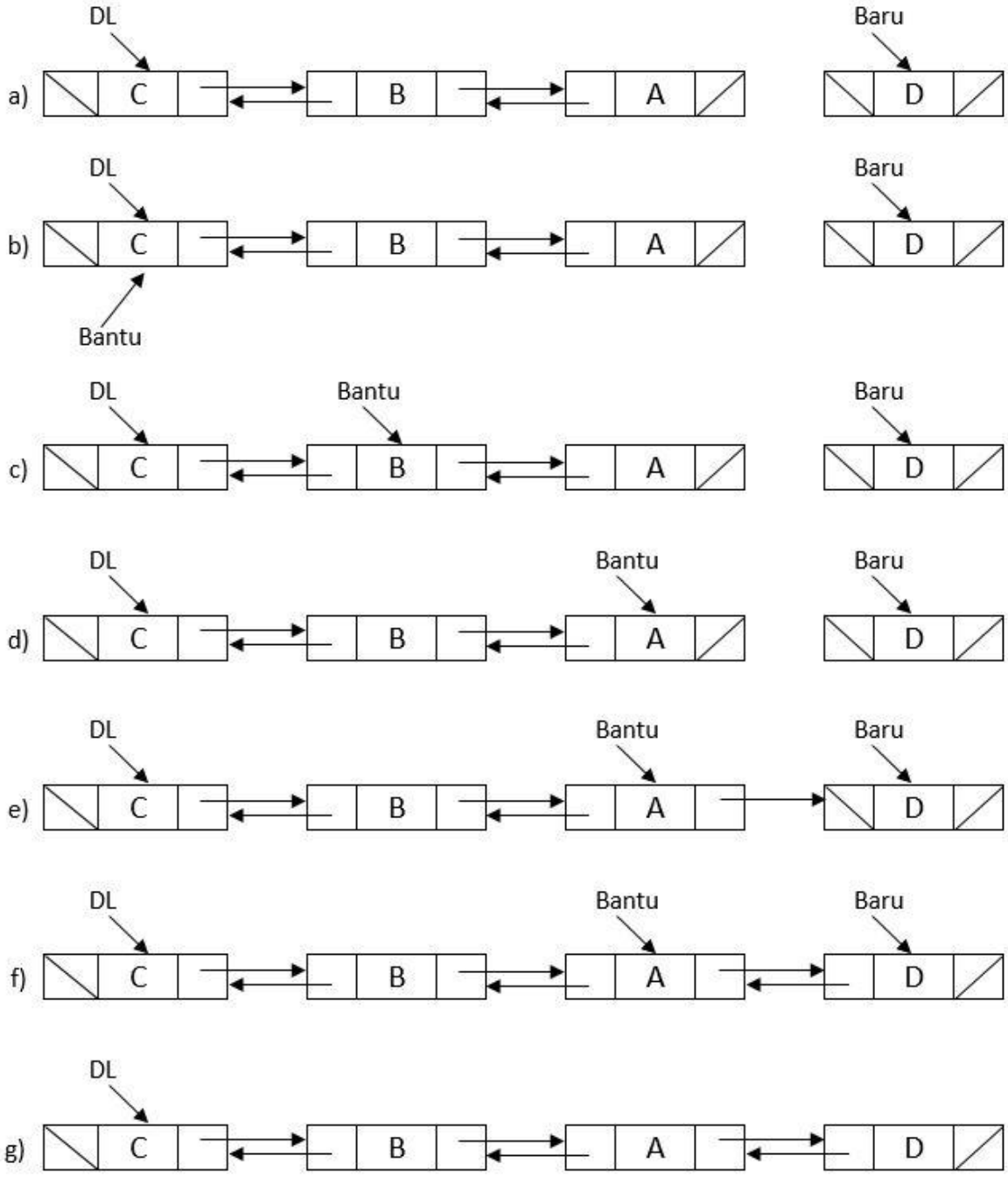
Penyisipan simpul baru selalu berada di posisi paling belakang. Penyisipan simpul belakang dapat dilakukan dengan dua cara, yaitu dengan menggunakan pointer bantu dan tanpa pointer bantu. Penyisipan simpul pada Doubly Linked List tidak mengharuskan menggunakan pointer bantu karena walaupun menggerakkan DL tidak mengakibatkan adanya simpul yang hilang. Hal ini dapat 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 Lis (DL) sudah ada maka penyisipan dilakukan dengan:
    - Buat pointer bantu yang dapat digerakkan (Bantu=DL).
    - Gerakkan pointer bantu hingga simpul terakhir (while(Bantu->Kanan != NULL) Bantu=Bantu->Kanan).
    - Pointer kanan simpul bantu menunjuk baru (Bantu->Kanan = Baru).
    - Pointer kiri baru menunjuk bantu (Baru->Kiri = Bantu).

Penyisipan simpul belakang dengan DL kosong sama seperti penyisipan simpul depan dengan DL kosong (Gambar 3), tetapi operasi penyisipan simpul

belakang untuk kasus DL tidak kosong dapat dilihat pada gambar 4.



Gambar 4. Operasi Penyisipan Simpul Belakang dengan Pointer Bantu

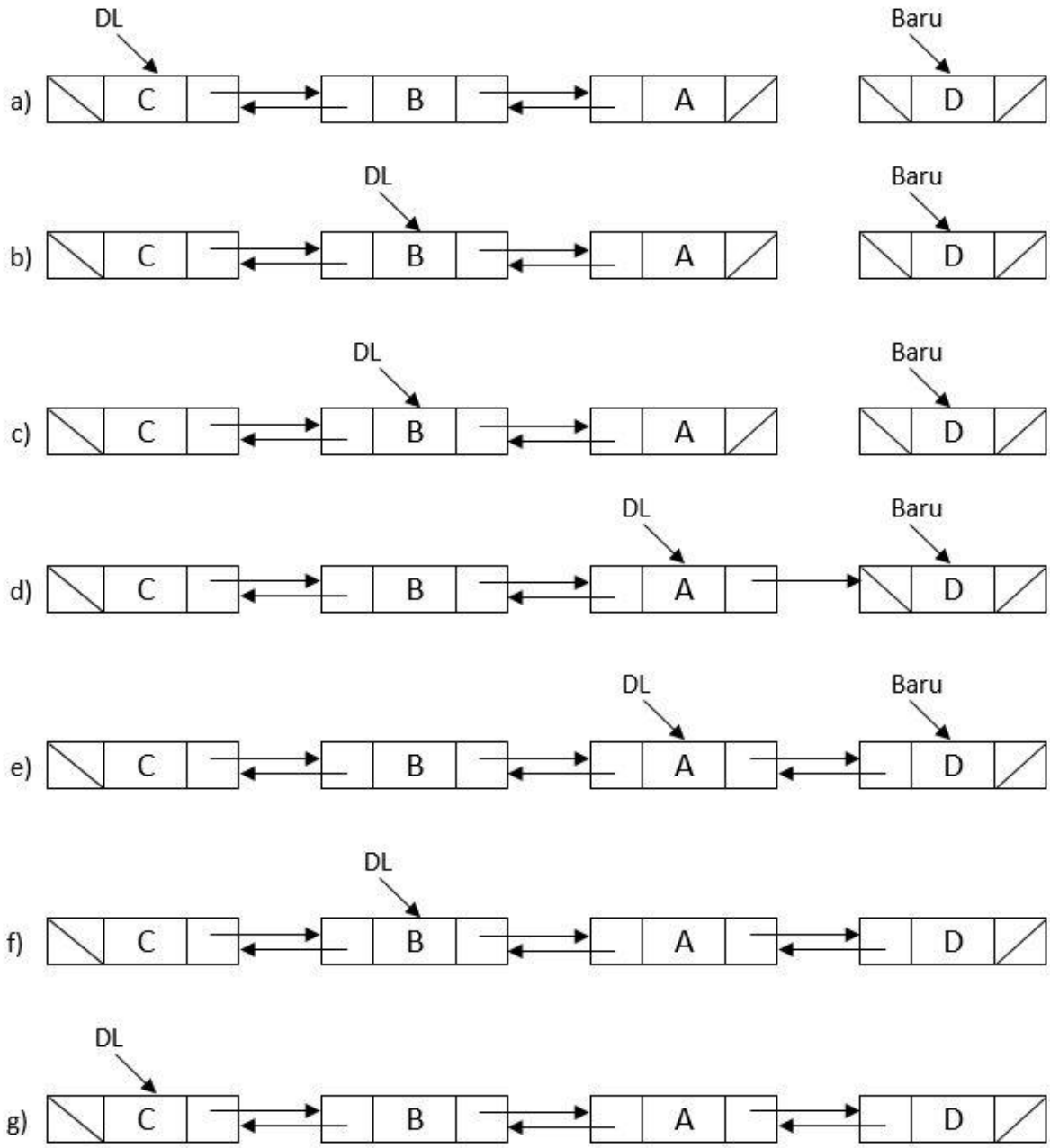
Fungsi penyisipan simpul belakang dengan mengikuti langkah-langkah dan gambar 4 di atas dapat dilihat berikut ini.

```
void Sisip_Belakang(simpul &DL, char elemen)

{
```

```
simpul baru, Bantu;

baru = (simpul) malloc(sizeof(simpul));

baru->Isi = elemen;

baru->kanan = NULL;

baru->kiri = NULL;

if(DL == NULL)

DL=baru;

else

{

Bantu = DL;

while (Bantu->kanan!=NULL) Bantu=Bantu->kanan

Bantu->kanan = baru;

baru->kiri = Bantu;

}

}
```

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 (while(DL->Kanan != NULL) DL=DL->Kanan).
    - Pointer kanan DL menunjuk baru (DL->Kanan = Baru).
    - Pointer kiri b aru menunjuk DL(Baru->Kiri = DL).
    - Gerakkan Pointer DL hingga simpul pertama (while(DL->Kiri != NULL) DL=DL->Kiri).



Gambar 5. Operasi Penyisipan Simpul Tanpa Pointer Bantu

Fungsi penyisipan simpul belakang dengan mengikuti langkah-langkah dan gambar 5 di atas dapat dilihat berikut ini.

```
void Sisip_Belakang(simpul &DL, char elemen)
{
simpul baru;
baru = (simpul) malloc(sizeof(simpul));
```

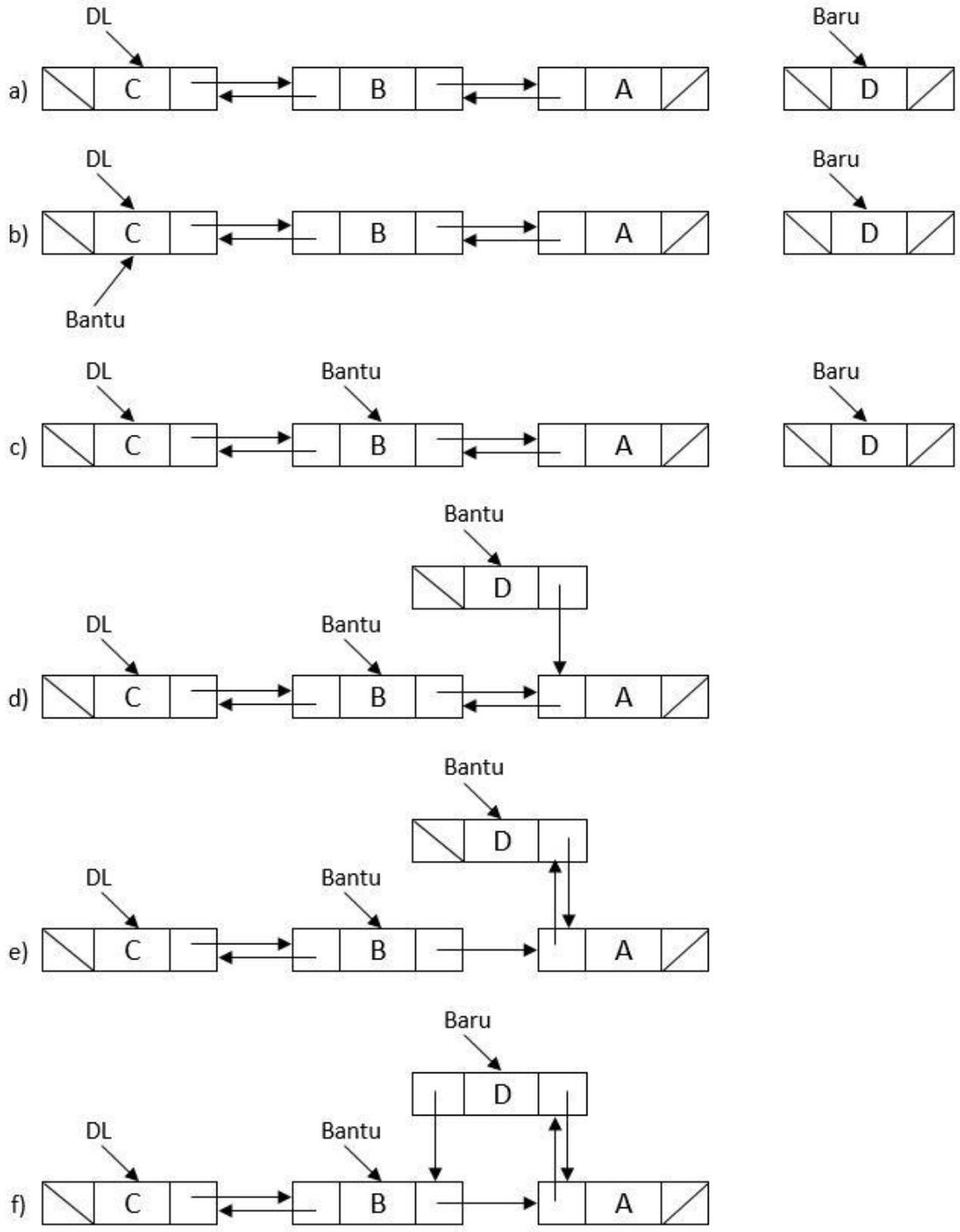
```
baru->Isi = elemen;

baru->kanan = NULL;

baru->kiri = NULL;

if(DL == NULL)

DL=baru;

else

{

while (DL->kanan!=NULL) DL=DL->kanan;

DL->kanan = baru;

baru->kiri = DL;

while (DL->kiri!=NULL) DL=DL->kiri;

}

}
```

3. Penyisipan Simpul Tengah

Menyisipkan suatu simpul Baru diantara dua simpul. Penyisipan simpul tengah hanya dapat dilakukan jika linked list (DL) tidak kosong. Misalkan kita mempunyai dobly 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 simpul yang berisi informasi B (elemen).

a. Penyisipan dengan pointer bantu

- Ciptakan simpul baru.
- Buat pointer bantu yang dapat digerakkan (Bantu=DL).
- Gerakkan pointer bantu hingga simpl 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->Kanan = Baru).
- Pointer kiri baru menunjuk bantu (Baru->Kiri = Bantu).



Gambar 6. Operasi Penyisipan Simpul Tengah dengan Pointer Bantu

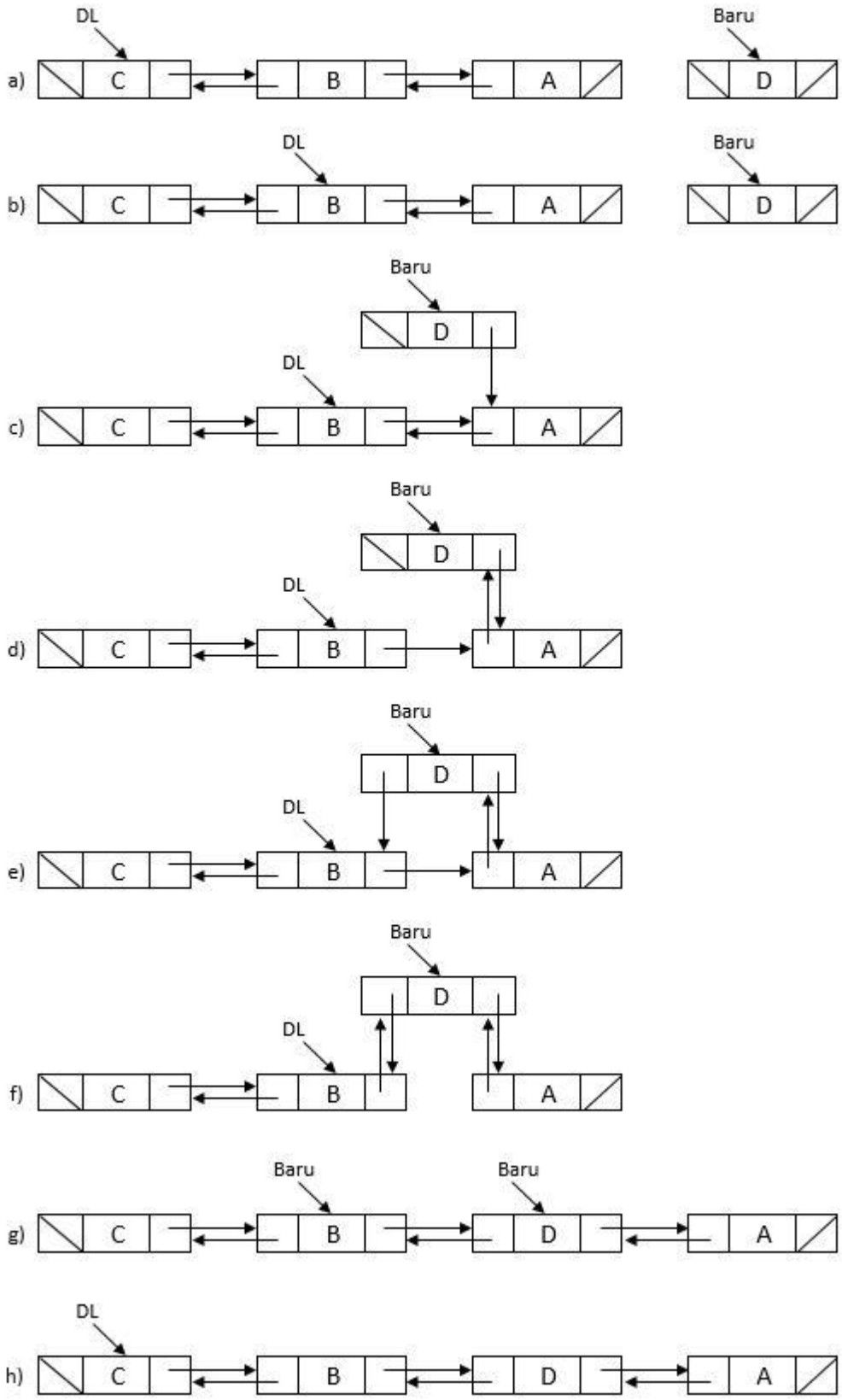
Fungsi penyisipan simpul tengah dengan mengikuti langkah-langkah dan gambar 6 di atas dapat dilihat berikut ini.

```
void Sisip_Tengah(simpul &DL, char elemen1, char elemen2)
```

```
{
simpul bantu, baru;
baru = (simpul) malloc(sizeof(simpul));
baru->Isi = elemen1;
baru->kanan = NULL;
baru->kiri = NULL;
if(DL == NULL)
cout<<"List Kosong ....."<<endl;
else
{
bantu = DL;
while(bantu->kanan->Isi != elemen2)
bantu=bantu->kanan;
baru->kanan = bantu->kanan;
baru->kiri = bantu;
bantu->kanan->kiri = baru;
bantu->kanan = baru;
}
}
```

b. Penyisipan simpul tengah tanpa pointer bantu

- Ciptakan simpul baru.
- Gerakkan pointer DL hingga simpul yang berisi informasi karakter.
- Kanan simpul baru menunjuk kanan DL (Baru->Kanan = DL->Kanan).
- Pointer kiri dari kanan DL menunjuk baru (DL->Kanan->Kiri = Baru).
- Pointer kiri baru menunjuk DL (Baru->Kiri = DL).
- Gerakkan pointer DL hingga simpul pertama.



Gambar 7. Operasi Penyisipan Simpul Tengah Tanpa Pointer Baru

Fungsi penyisipan simpul tengah dengan mengikuti langkah-langkah dan gambar 7 di atas dapat dilihat berikut ini.

```
void Sisip_Tengah2(simpul &DL, char elemen1, char elemen2)
{
simpul baru;
baru = (simpul) malloc(sizeof(simpul));
baru->Isi = elemen1;
baru->kanan = NULL;
baru->kiri = NULL;
if(DL == NULL)
cout<<"List Kosong ....."<<endl;
else
{
while(DL->kanan->Isi != elemen2) DL = DL->kanan;
baru->kanan = DL->kanan;
baru->kiri = DL;
DL->kanan->kiri = baru;
DL->kanan = baru;
}
while(DL->kiri != NULL) DL = DL->kiri;
}
```

2.2. Penghapusan Simpul

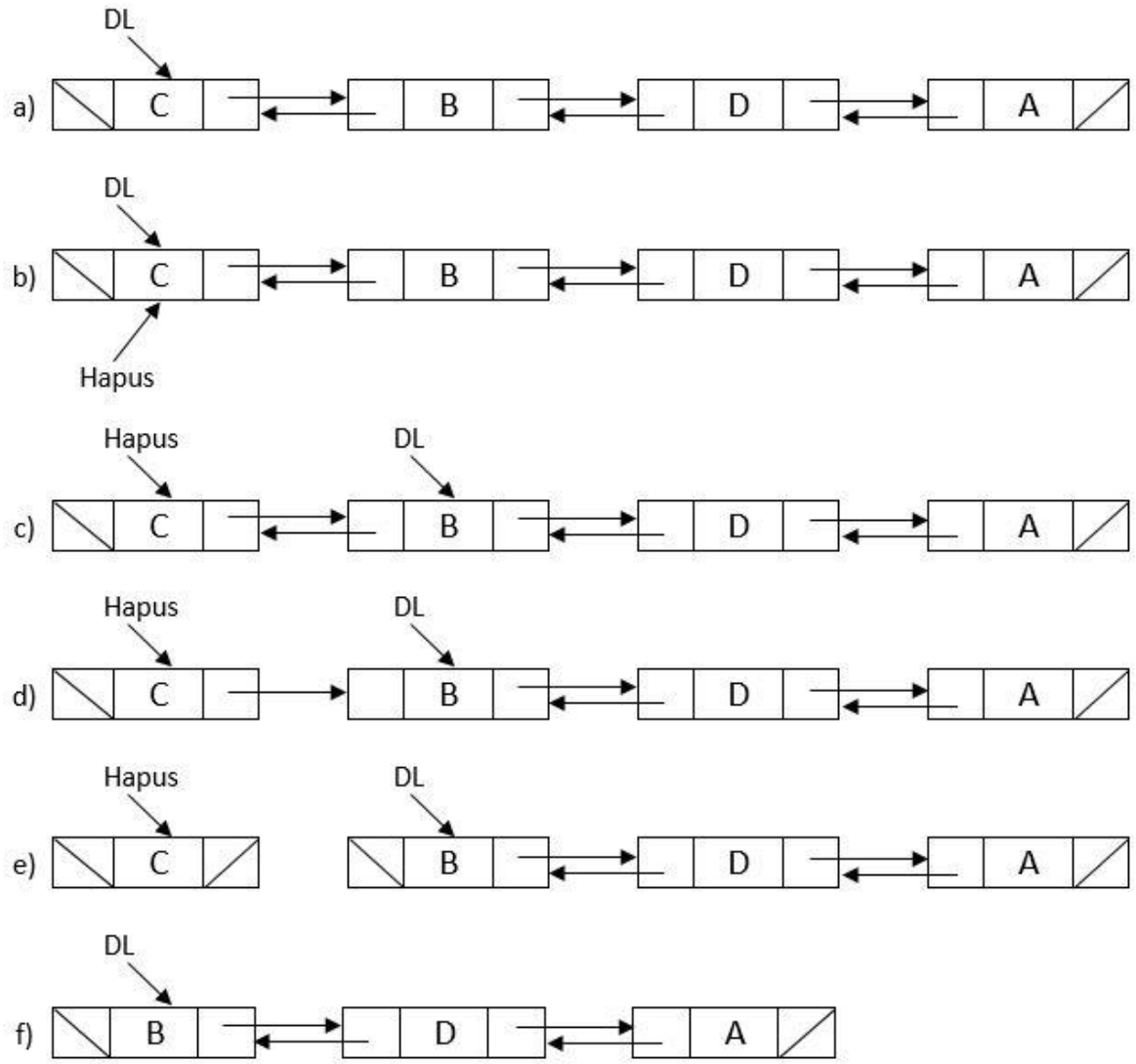
Operasi menghapus suatu simpul dari suatu Linked List pada Doubly Linked List hampir sama dengan penghapusan simpul pada Singly Linked List, yaitu Linked List (DL) tidak boleh dalam keadaan kosong. Penghapusan simpul juga dapat dilakukan terhadap simpul depan, simpul belakang, dan simpul tengah.

1. Penghapusan simpul depan

Simpul yang dihapus selalu yang 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 (Hapus = DL).
- Pointer DL digerakkan satu simpul ke kanan (DL = DL->Kanan).
- Putuskan pointer Kiri DL dari hapus (DL->Kiri = NULL).
- Putuskan pointer kanan hapus dari Linked List DL (Hapus->Kanan=NULL).



Gambar 8. Operasi Penghapusan Simpul Depan

Fungsi penghapusan simpul depan dengan mengikuti langkah-langkah dan gambar 8 di atas dapat dilihat berikut ini.

```
void Hapus_Depan(simpul &DL)
{
```

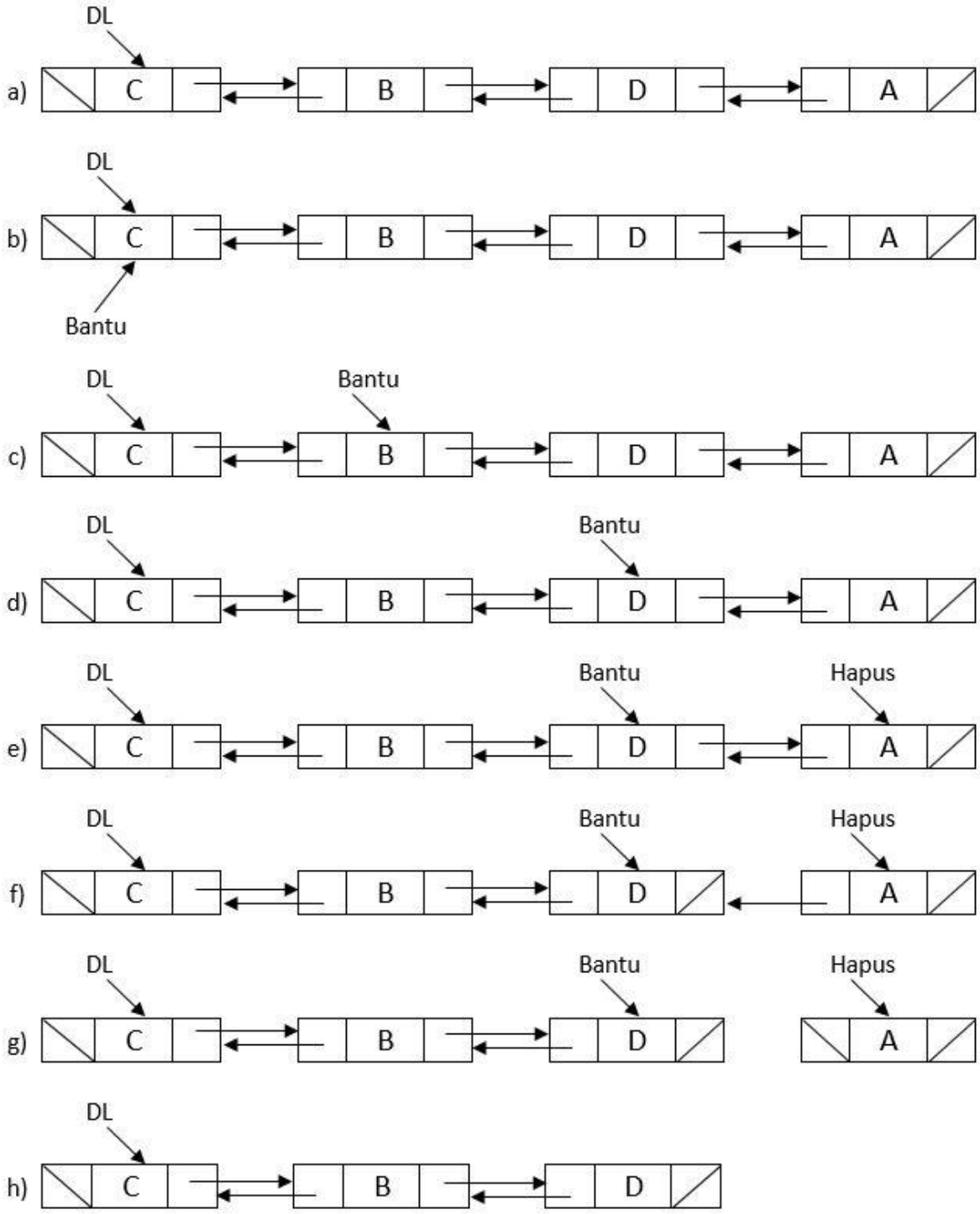
```
simpul Hapus;
if(DL==NULL)
cout<<"Linked List Kosong .....";
else
{
Hapus = DL;
DL = DL->kanan;
DL->kiri = NULL;
Hapus->kanan = NULL;
free(Hapus);
}
}
```

2. Penghapusan Simpul Belakang

Simpul yang dihapus selalu yang berada pada posisi belakang. Misalkan kita memiliki Linked List DL terdiri dari 4 simpul dengan masing-masing berisi informasi C, B, D, dan A, akan dilakukan penghapusan simpul belakang yaitu simpul yang berisi informasi A. Dalam menghapus simpul belakang, kita dapat lakukan dengan dua cara, yaitu dengan menggunakan pointer atau tanpa menggunakan pointer bantu. Langkah-langkah penghapusan simpul belakang dengan dan tanpa bantu dapat dilakukan seperti berikut.

a. Penghapusan Simpul Belakang dengan Pointer Bantu

- Buatlah pointer bantu yang menunjuk simpul pertama yang ditunjuk oleh pointer DL (Bantu = DL).
- Gerakkan pointer bantu hingga satu simpul sebelum simpul belakang (while(Bantu->Kanan->Kanan != NULL) Bantu = Bantu->Kanan).
- Simpul belakang yang ditunjuk pointer kanan bantuk ditunjuk juga oleh pointer hapus(Hapus = Bantu->Kanan).
- Putuskan pointer kanan bantu dari simpul yang ditunjuk pointer hapus (Bantu->Kanan=NULL).
- Putuskan pointer kiri hapus dari Linked List (Hapus->Kiri = NULL).
- Skema penghapusan simpul belakang dapat dilihat pada Gambar 9.



- Gambar 9. Operasi Penghapusan Simpul Belakang dengan Pointer Bantu

- Fungsi penghapusan simpul belakang dengan mengikuti langkah-langkah dan gambar 9 di atas dapat dilihat berikut ini.

```
void Hapus_Belakang(simpul &DL)
{
```

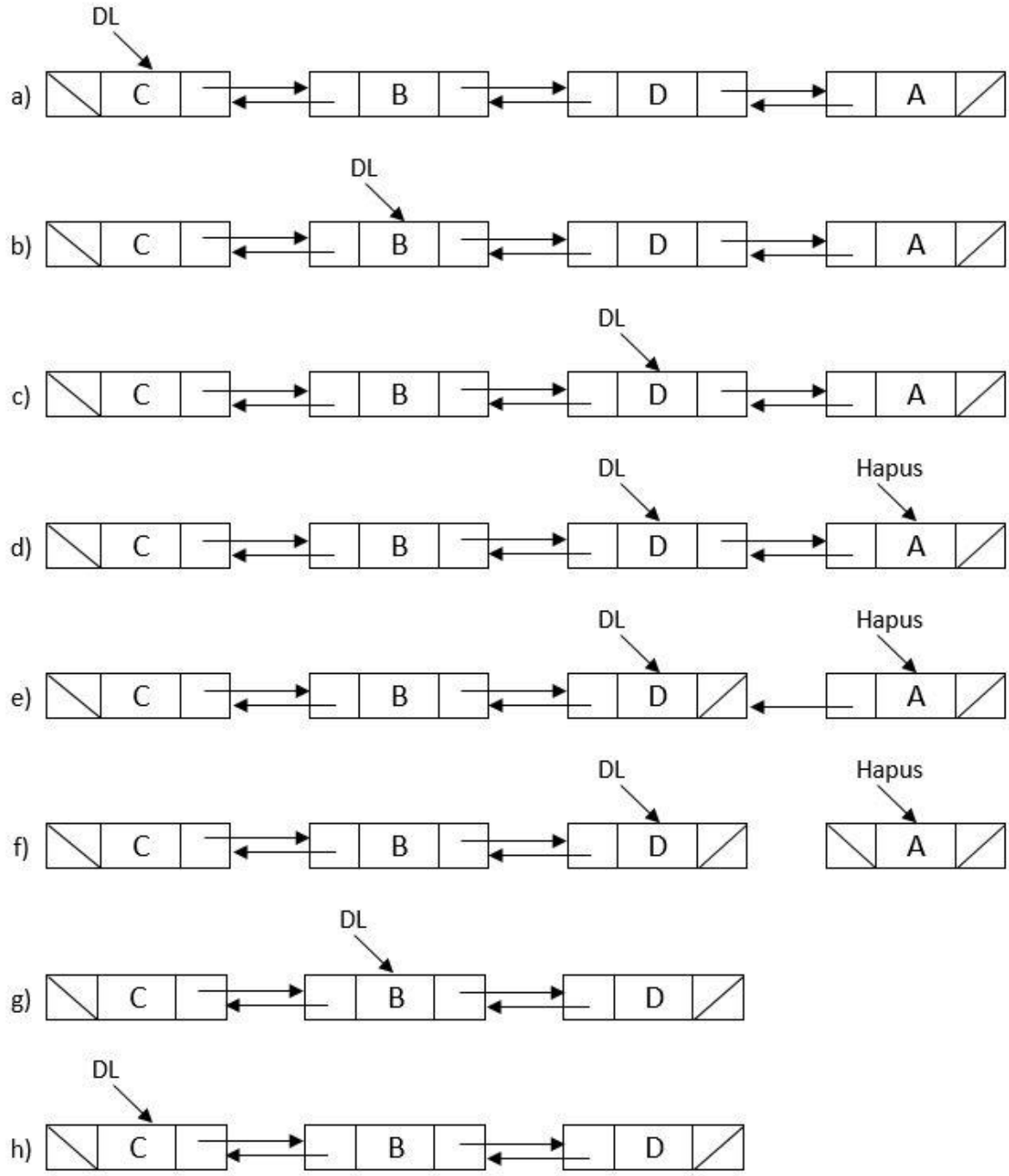
```
simpul bantu, hapus;
if(DL==NULL)
cout<<"Linked List Kosong .....";
else
{
bantu = DL;
while(bantu->kanan->kanan != NULL)
bantu=bantu->kanan;
hapus = bantu->kanan;
bantu->kanan = NULL;
hapus->kiri = NULL;
free(hapus);
}
}
```

- b. Penghapusan Simpul Belakang Tanpa Pointer Bantu (Versi 1)

Menghapus simpul belakang tanpa pointer bantu, dengan demikian kita cukup menggerakkan pointer DL hingga ke satu simpul sebelum simpul belakang. Setelah selesai menghapus simpul belakang kemudian pointer DL dipindahkan kembali hingga ke simpul depan.

- Gerakkan pointer DL hingga simpul sebelum simpul belakang (while(DL->Kanan->Kanan != NULL) DL = DL->Kanan).
- Simpul terakhir yang ditunjuk oleh pointer kanan DL ditunjuk juga oleh hapus (Hapus = DL->Kanan).
- Putuskan pointer kanan DL dari simpul yang ditunjuk hapus (DL->Kanan=NULL).
- Putuskan pointer kiri hapus dari simpul yang ditunjuk DL (Hapus->Kiri = NULL).
- Gerakkan pointer DL hingga simpul pertama (while(DL->Kiri != NULL) DL = DL->Kiri).

Skema penghapusan simpul belakang dapat dilihat pada Gambar 10.



Gambar 10. Operasi Penghapusan Simpul Belakang Tanpa Pointer Bantu

Fungsi penghapusan simpul belakang dengan mengikuti langkah-langkah dan gambar 10 di atas dapat dilihat berikut ini.

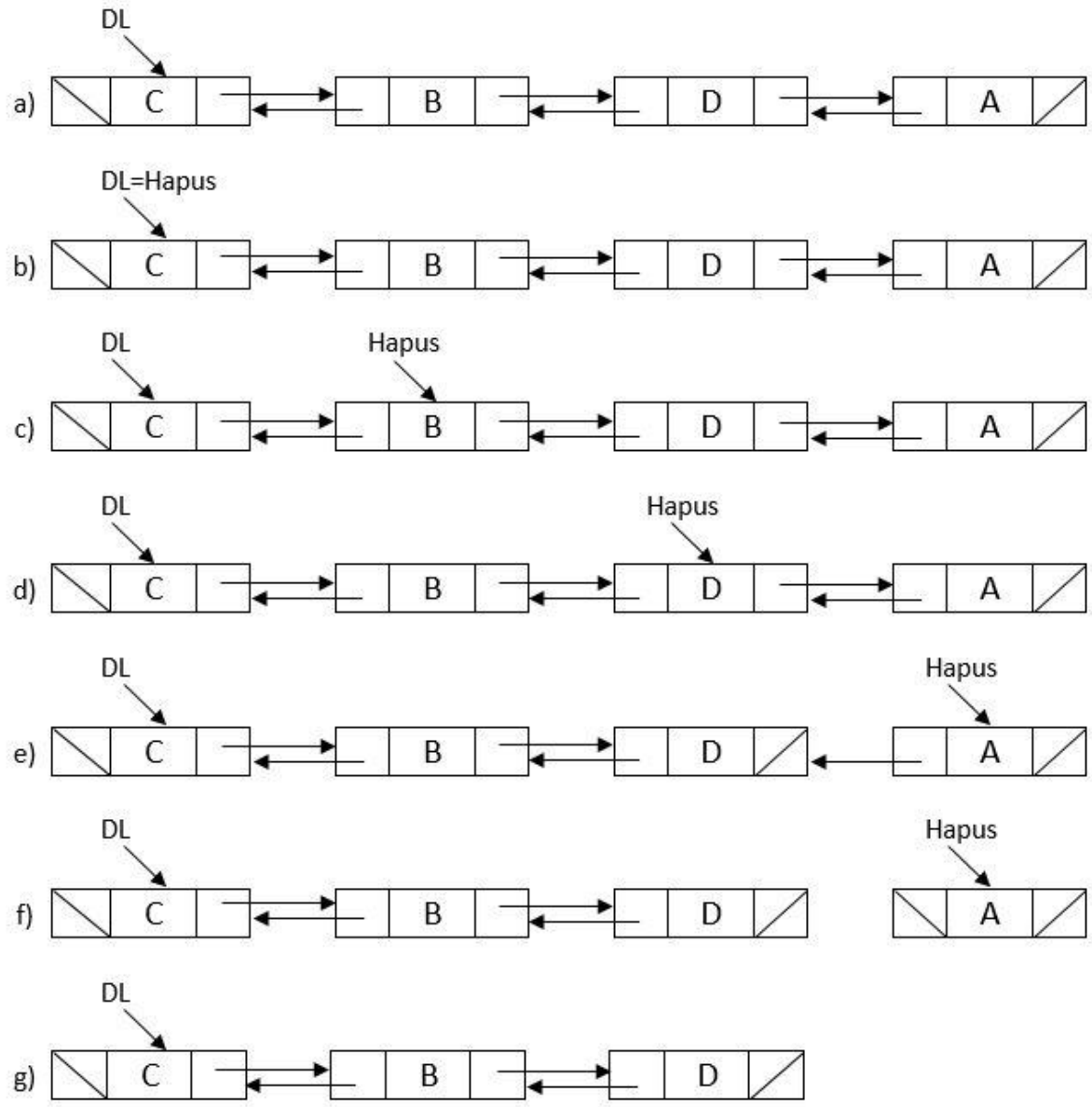
```
void Hapus_Belakang(simpul &DL)

{

simpul hapus;
```

```
if(DL==NULL)

cout<<"Linked List Kosong .....";

else

{

while(DL->kanan->kanan != NULL) DL=DL->kanan;

hapus = DL->kanan;

DL->kanan = NULL;

hapus->kiri = NULL;

free(hapus);

while(DL->kiri != NULL) DL=DL->kiri;

}

}
```

c. Penghapusan Simpul Belakang Tanpa Menggunakan Pointer Bantu (Versi 2)

Menghapus simpul belakang tanpa menggunakan pointer bantu dan tanpa menggerakkan pointer DL. Dalam versi 2 ini kita menggerakkan pointer hapus mulai simpul depan hingga simpul belakang.

- Pointer hapus menunjuk simpul depan (Hapus = DL).
- Gerakkan pointer hapus hingga simpul belaakng (while(Hapus->kanan != NULL) Hapus=Hapus->Kanan;).
- Putuskan pointer kanan dari kiri hapus dari hapus (Hapus->kiri->kanan = NULL).
- Putuskan pointer kiri hapus (Hapus->kiri = NULL).



Gambar 11. Operasi Penghapusan Simpul Belakang Tanpa Pointer Bantu (versi 1)

Fungsi penghapusan simpul belakang dengan mengikuti langkah-langkah dan gambar 11 di atas dapat dilihat berikut ini.

```
void Hapus_Belakang(simpul &DL)
{
simpul Hapus;
if(DL==NULL)
cout<<"Linked List Kosong .....";
```

```
else

{

Hapus = DL;

while(Hapus->Kanan != NULL) Hapus=Hapus->Isi;

Hapus->kiri->kanan = NULL;

Hapus->kiri = NULL;

free(Hapus);

}

}
```

3. Penghapusan Simpul Tengah

Berbeda dengan penghapusan simpul depan dan penghapusan simpul belakang, simpul yang akan dihapus adalah pasti yang ada di depan atau yang ada di belaakng dan hanya ada masing-masing satu. Tetapi karena simpul tengah mungkin lebih dari satu simpul maka harus diketahui simpul yang mana yang akan dihapus. Misalkan kita memiliki Linked List DL terdiri dari 4 simpul dengan masing-masing berisi informasi C, B, D, dan A, akan dilakukan penghapusan simpul yang ada di posisi tengah (simpul yang berisi informasi B atau D) yaitu simpul yang berisi informasi D (elemen). Dalam menghapus simpul belakang, kita dapat lakukan dengan dua cara juga, yaitu dengan menggunakan pointer bantu atau tanpa menggunakan pointer bantu.

https://pintarkom-com.cdn.ampproject.org/v/s/pintarkom.com/double-linked-list-pada-c-plus/amp/?amp_js_v=a6&amp_gsa=1&usqp=mq331AQKKAFQArABIIACAw%3D%3D#aoh=16465832578049&referrer=https%3A%2F%2Fwww.google.com&amp_tf=Dari%20%251%24s&ampshare=https%3A%2F%2Fpintarkom.com%2Fdouble-linked-list-pada-c-plus%2F