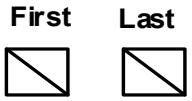
Modul 6 DOUBLY LINKED LIST (BAGIAN PERTAMA)

TUJUAN PRAKTIKUM
1. Memahami konsep modul *linked list*. 2. Mengaplikasikan konsep *Doubly linked list* dengan menggunakan *pointer* dan dengan bahasa C++

6.1 *Doubly Linked List*

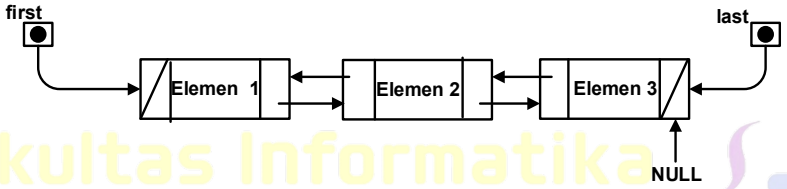
Doubly Linked list adalah *linked list* yang masing – masing elemen nya memiliki 2 *successor*, yaitu *successor* yang menunjuk pada elemen sebelumnya (*prev*) dan *successor* yang menunjuk pada elemen sesudahnya (*next*).

Gambar berikut menunjukan bentuk *Doubly Linked list* dengan elemen kosong:



Gambar 6-1 *Doubly Linked list* dengan Elemen Kosong

Gambar berikut menunjukan bentuk *Doubly Linked list* dengan 3 elemen:



Gambar 6-2 *Doubly Linked list* dengan 3 Elemen

Doubly linked list juga menggunakan dua buah *successor* utama yang terdapat pada *list*, yaitu *first* (*successor* yang menunjuk elemen pertama) dan *last* (susesor yang menunjuk elemen terakhir *list*).

Komponen-komponen dalam *Doubly linked list*:

1. *First* : *pointer* pada *list* yang menunjuk pada elemen pertama *list*.
2. *Last* : *pointer* pada *list* yang menunjuk pada elemen terakhir *list*.
3. *Next* : *pointer* pada elemen sebagai *successor* yang menunjuk pada elemen didepannya.
4. *Prev* : *pointer* pada elemen sebagai *successor* yang menunjuk pada elemen dibelakangnya.

Contoh pendeklarasian struktur data untuk *Doubly linked list*:

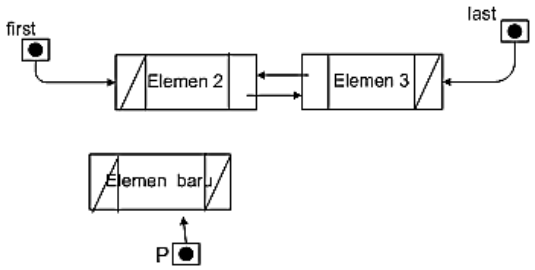
```
1   #ifndef Doublylist_H
2   #define Doublylist_H
3   #include "boolean.h"
4   #define Nil NULL
5
6   /*deklarasi record dan struktur data Doubly linked list*/
7   typedef int infotype;
8   typedef struct elmlist *address;
9   struct elmlist {
10      infotype info;
11      address next;
12      address prev;
13  };
14
15  /* definisi list: */
16  /* list kosong jika L.first=Nil */
17  struct list{
18      address first;
```

```
    address last;
};
#endif
```

6.1.1 Insert

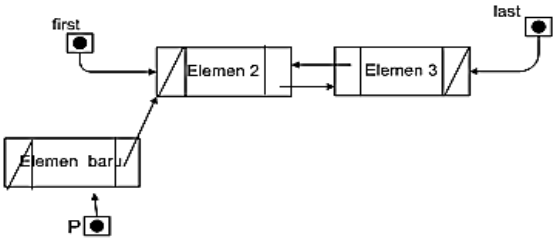
A. *Insert First*

Langkah-langkah dalam proses *insert first*:



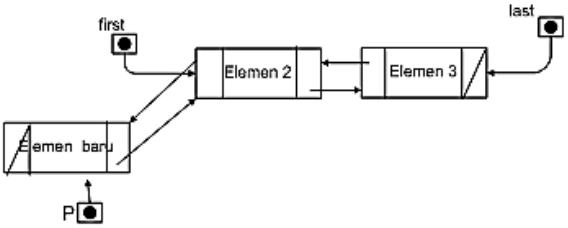

```
P = alokasi(X);
P→next = Nil;
P→prev = Nil;
```

Gambar 6-3 *Doubly Linked list Insert First* 1



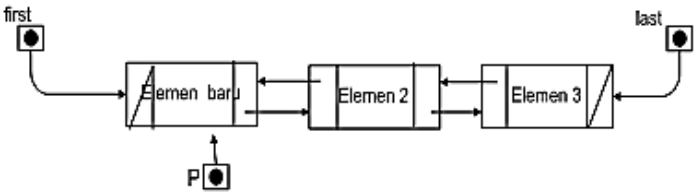

```
P→next = L.first;
```

Gambar 6-4 *Doubly Linked list Insert First* 2




```
(L.first)→prev = P;
```

Gambar 6-5 *Doubly Linked list Insert First* 3



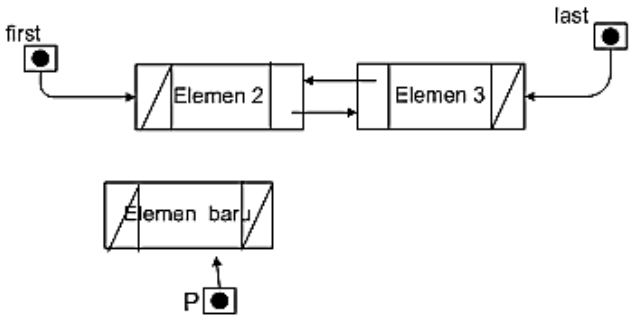

```
L.first = P;
```

Gambar 6-6 *Doubly Linked list Insert First* 4

```
void insertFirst(list &L, address &P){
    P→next = L.first
    (L.first)→prev = P;
    L.first = P;
}
```

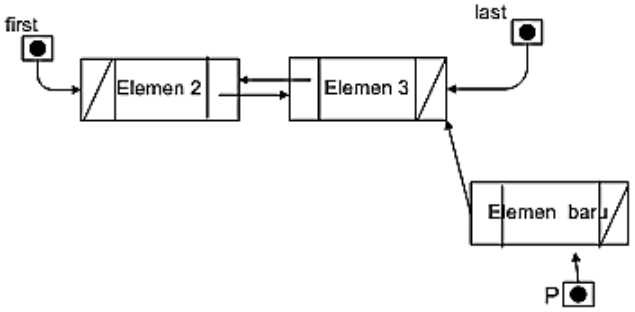
B. *Insert Last*

Langkah-langkah dalam proses *insert last*:



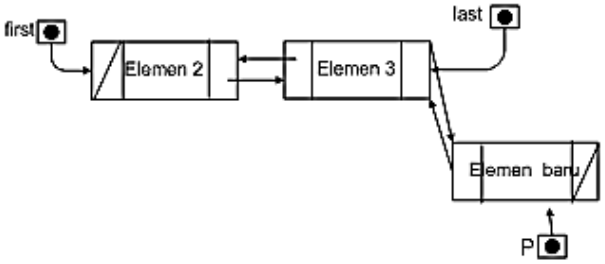

```
P = alokasi(X);
P→ next = Nil;
P→prev = Nil;
```

Gambar 6-7 *Doubly Linked list Insert Last* 1



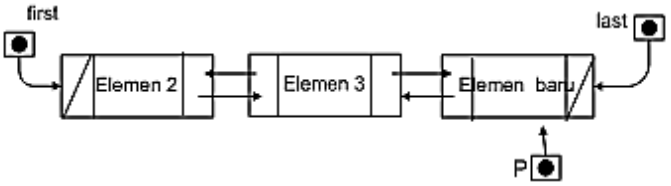

```
P→prev = L.last;
```

Gambar 6-8 *Doubly Linked list Insert Last* 2




```
(L.last)→next = P;
```

Gambar 6-9 *Doubly Linked list Insert Last* 3

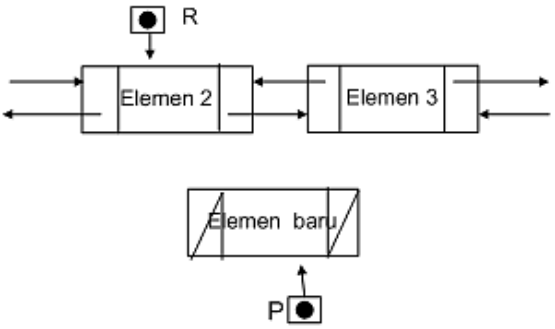



```
L.last = P;
```

Gambar 6-10 *Doubly Linked list Insert Last* 4

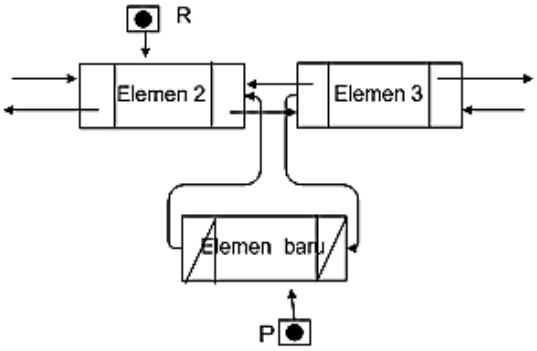
C. *Insert After*

Langkah-langkah dalam proses *insert after*:



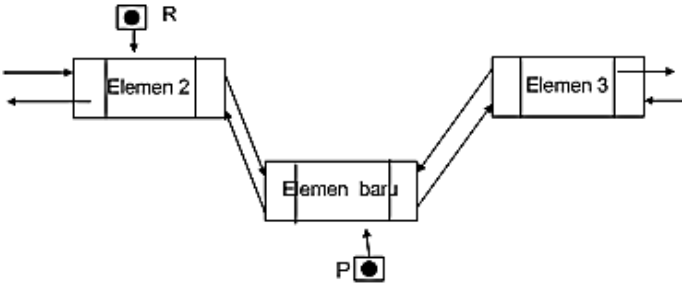

```
P = alokasi(X);
P→next = Nil;
P→prev = Nil;
```

Gambar 6-11 *Doubly Linked list Insert After* 1




```
P→next = R→next;
P→prev = R;
```

Gambar 6-12 *Doubly Linked list Insert After* 2




```
(R→next)→prev = P;
R→prev = P;
```

Gambar 6-13 *Doubly Linked list Insert After* 3

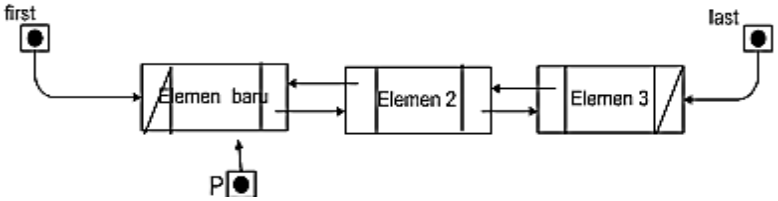
D. *Insert Before*

Di atas hanya dijelaskan tentang *insert after*. *Insert before* hanya kebalikan dari *insert after*. Perbedaan *Insert After* dan *Insert Before* terletak pada pencarian elemennya.

6.1.2 Delete

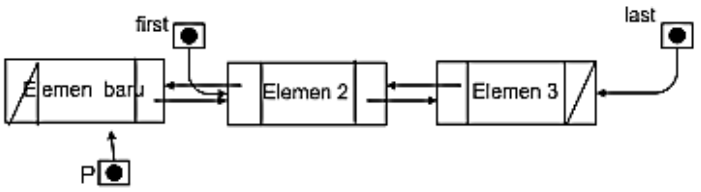
A. *Delete First*

Langkah-langkah dalam proses *delete first*:



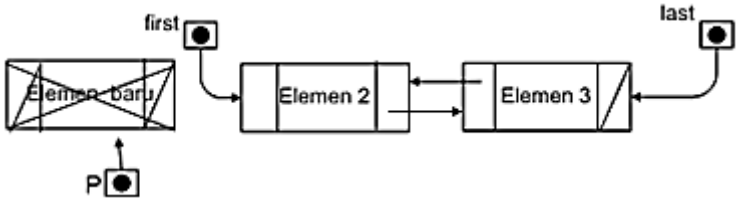

```
P = L.first;
```

Gambar 6-14 *Doubly Linked list Delete First* 1




```
L.first = (L.first)→next;
P→next = Nil;
```

Gambar 6-15 *Doubly Linked list Delete First* 2



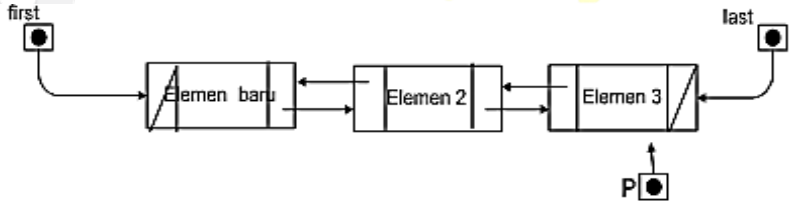

```
(L.first)→prev = Nil;
P→next = Nil;
return P;
          atau
dealokasi(P);
```

Gambar 6-16 *Doubly Linked list Delete First* 3

```
/* contoh syntax delete first */
void deleteFirst(list &L, address &P){
    P = L.first;
    L.first = (L.first)→next;
    P→prev = Nil;
    (L.first)→prev = Nil;
    P→next = Nil;
}
```

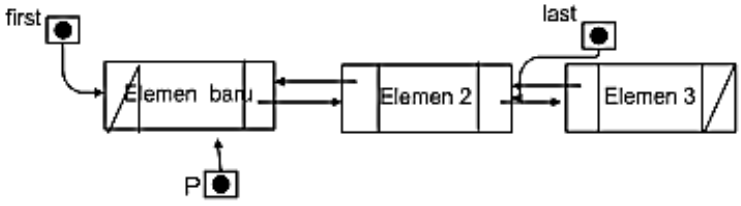
B. *Delete Last*

Langkah-langkah dalam proses *delete last*:



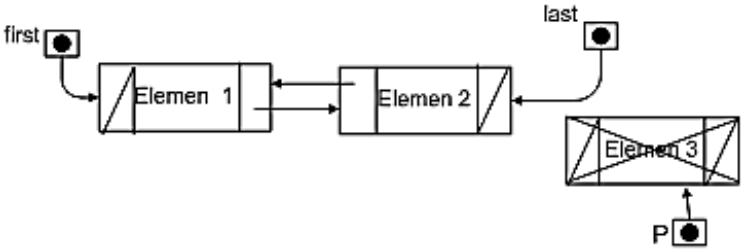

```
P = L.last;
```

Gambar 6-17 *Doubly Linked list Delete Last* 1




```
L.last = (L.last)→prev;
```

Gambar 6-18 *Doubly Linked list Delete Last* 2

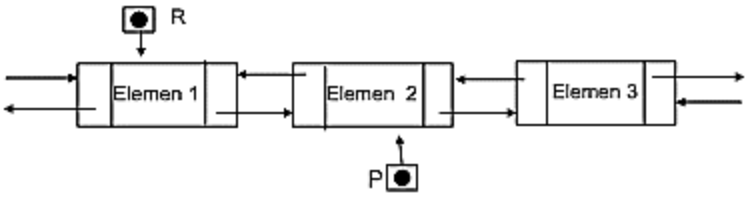



```
P→prev = Nil;
(L.last)→next = Nil;
return P;
          atau
dealokasi(P);
```

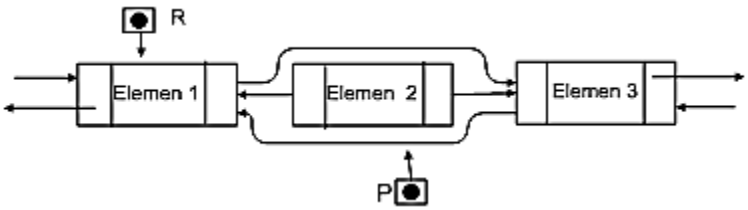
Gambar 6-19 *Doubly Linked list Delete Last* 3

C. *Delete After*

Langkah-langkah dalam proses *delete after*:

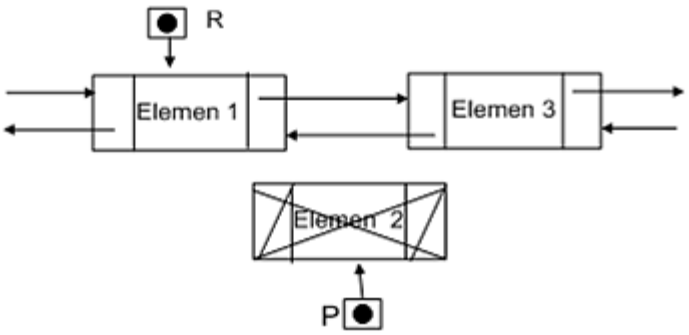


Gambar 6-20 *Doubly Linked list Delete After* 1




```
R→next = P→next;
(P→next)→prev = R;
```

Gambar 6-21 *Doubly Linked list Delete After* 2




```
P→prev = Nil;
P→next = Nil;
return P;
        atau
dealokasi(P);
```

Gambar 6-22 *Doubly Linked list Delete After* 3

D. *Delete Before*

Di atas hanya dijelaskan tentang *delete after*. *Delete before* hanya kebalikan dari *delete after*. Perbedaan *Delete After* dan *Delete Before* terletak pada pencarian elemennya.

E. *Update*, *View*, dan *Searching*

Proses pencarian, *update* data dan *view* data pada dasarnya sama dengan proses pada *Singly linked list*. Hanya saja pada *Doubly linked list* lebih mudah dalam melakukan proses akses elemen, karena bisa melakukan iterasi maju dan mundur.

Seperti halnya *Singly linked list*, *Doubly linked list* juga mempunyai ADT yang pada dasarnya sama dengan ADT yang ada pada *Singly linked list*.

```
1   /*file : Doublylist .h*/
2   /* contoh ADT list berkait dengan representasi fisik pointer*/
3   /* representasi address    dengan pointer*/
4   /* info tipe adalah integer */
5   #ifndef Doublylist_H
6   #define Doublylist_H
7
8   #define Nil NULL
9
10  typedef int infotype;
11  typedef struct elmlist *address;
12  /* pendefinisian tipe data bentukan elemen list
13     dengan dua successor, yaitu next dan prev */
14  struct elmlist{
15      infotype info;
16      address prev;
17      address next;
18  };
19
20  /* definisi Doubly linked list : list kosong jika L.first=Nil
21     setiap elemen address P dapat diacu P→info atau P→next
22     elemen terakhir adalah last
23     nama tipe list yang dipakai adalah 'list', sama dengan pada singly list*/
24  struct list {
25      address first,last;
26  };
27
28  /** Deklarasi fungsi primitif lain **/
29  /** Sama dengan Singly Linked list **/
30
31
32
33
34
35
```


6.2 *Latihan*

1. Buatlah ADT *Doubly Linked list* sebagai berikut di dalam *file* **"Doublylist.h"**:

```
Type infotype : kendaraan <
    nopol : string
    warna : string
    thnBuat : integer
>
Type address : pointer to ElmList
Type ElmList <
    info : infotype
    next :address
    prev : address
>

Type List <
    First : address
    Last : address
>
procedure CreateList( input/output L : List )
function alokasi( x : infotype ) → address
procedure dealokasi(input/output P : address )
procedure printInfo( input L : List )
procedure insertLast(input/output L : List,
   input P : address )
```

Buatlah implementasi ADT *Doubly Linked list* pada *file* "Doublylist.cpp" dan coba hasil implementasi ADT pada *file* "main.cpp".

Contoh *Output* :

```
masukkan nomor polisi: D001
masukkan warna kendaraan: hitam
masukkan tahun kendaraan: 90

masukkan nomor polisi: D003
masukkan warna kendaraan: putih
masukkan tahun kendaraan: 70

masukkan nomor polisi: D001
masukkan warna kendaraan: merah
masukkan tahun kendaraan: 80
nomor polisi sudah terdaftar

masukkan nomor polisi: D004
masukkan warna kendaraan: kuning
masukkan tahun kendaraan: 90

DATA LIST 1

no polisi : D004
warna     : kuning
tahun     : 90
no polisi : D003
warna     : putih
tahun     : 70
no polisi : D001
warna     : hitam
tahun     : 90
```

Gambar 6-23 *Output* kasus kendaraan

2. Carilah elemen dengan nomor polisi D001 dengan membuat fungsi baru. fungsi findElm(L : *List*, x : infotype) : *address*

```
Masukkan Nomor Polisi yang dicari   : D001

Nomor Polisi   : D001
Warna          : hitam
Tahun          : 90
```

Gambar 6-24 *Output* mencari nomor polisi

3. Hapus elemen dengan nomor polisi D003 dengan *procedure delete*.
   - *procedure* `deleteFirst( input/output L : List, P : address )`
   - *procedure* `deleteLast( input/output L : List, P : address )`
   - *procedure* `deleteAfter( input Prec : address, input/output P : address )`

```
Masukkan Nomor Polisi yang akan dihapus   : D003
Data dengan nomor polisi D003 berhasil dihapus.

DATA LIST 1

Nomor Polisi   : D004
Warna          : kuning
Tahun          : 90
Nomor Polisi   : D001
Warna          : hitam
Tahun          : 90
```

Gambar 6-25 *Output* menghapus data nomor polisi