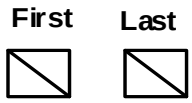
Modul 6 DOUBLE LINKED LIST (BAGIAN PERTAMA)

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

6.1 *Double Linked List*

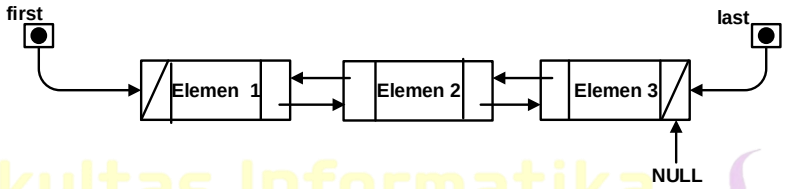
Double Linked list adalah *linked list* yang masing – masing elemen nya memiliki 2 *successor*, yaitu *successor* yang menunjuk pada elemen sebelumnya (*prev*) dan *successor* yang menunjuk pada elemen sesudahnya (*next*).

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



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

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



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

Double 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 *double 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 *double linked list*:

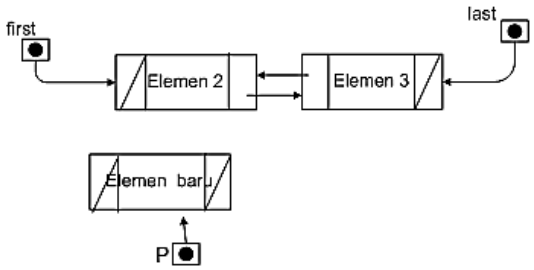
```
1   #ifndef doublelist_H
2   #define doublelist_H
3   #include "boolean.h"
4   #define Nil NULL
5   #define info(P) (P)->info
6   #define next(P) (P)->next
7   #define prev(P) (P)->prev
8   #define first(L) ((L).first)
9   #define last(L) ((L).last)
10
11  /*deklarasi record dan struktur data double linked list*/
12  typedef int infotype;
13  typedef struct elmlist *address;
14  struct elmlist {
15      infotype info;
16      address next;
17      address prev;
18  };
19
```

```
20  /* definisi list: */
21  /* list kosong jika First(L)=Nil */
22  struct list{
23      address first;
24      address last;
25  };
26  #endif
```

6.1.1 Insert

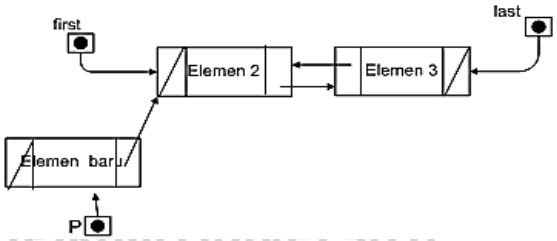
A. *Insert First*

Langkah-langkah dalam proses *insert first*:



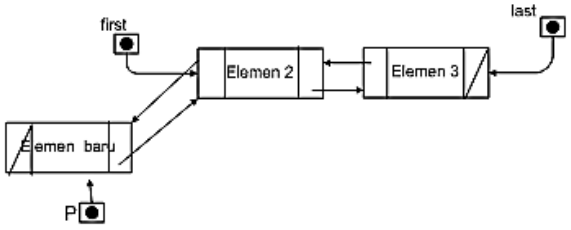

```
P = alokasi(X);
next(P) = Nil;
prev(P) = Nil;
```

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



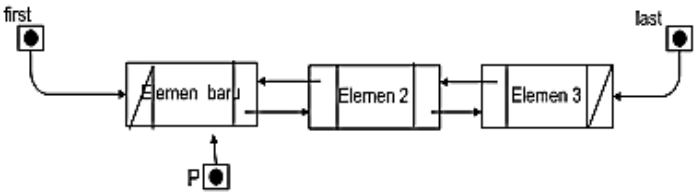

```
next(P) = first(L);
```

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




```
prev(first(L)) = P;
```

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



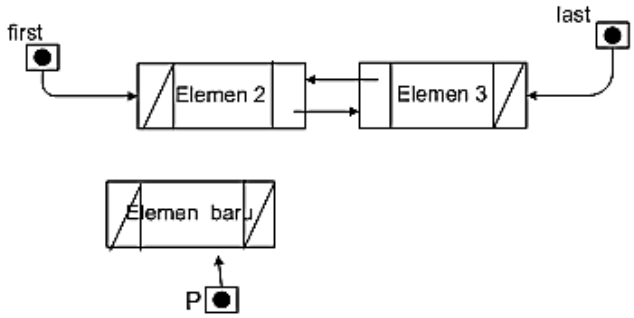

```
first(L) = P;
```

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

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

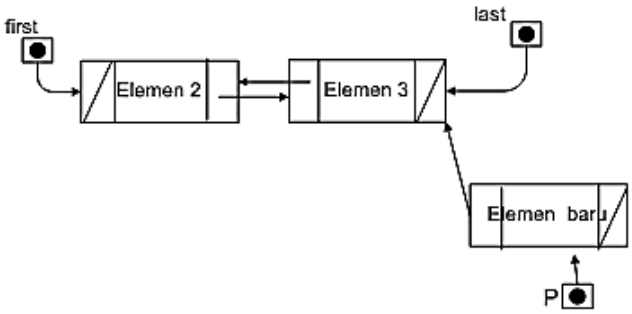
B. *Insert Last*

Langkah-langkah dalam proses *insert last*:



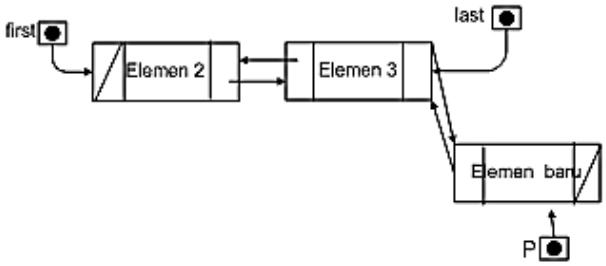

```
P = alokasi(X);
next(P) = Nil;
last(P) = Nil;
```

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



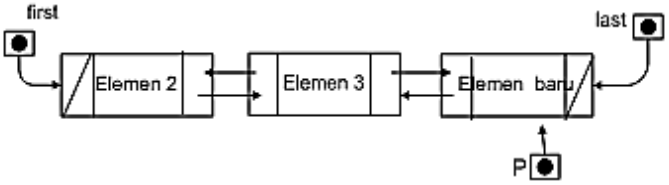

```
prev(P) = last(L);
```

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




```
next(last(L)) = P;
```

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

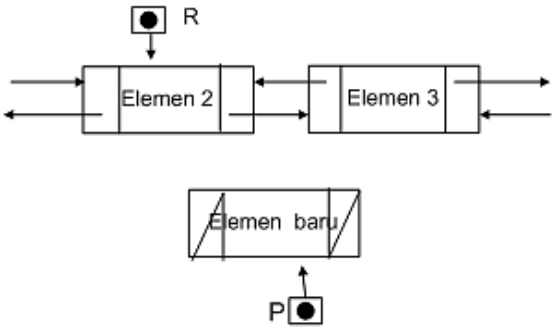



```
last(L) = P;
```

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

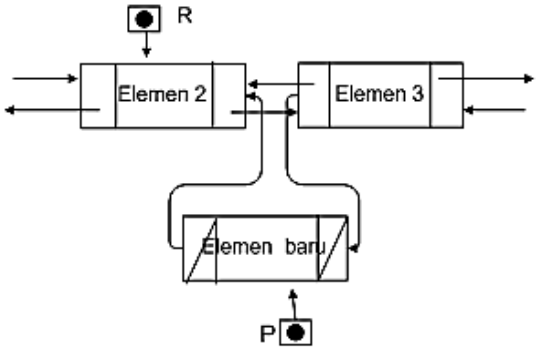
C. *Insert After*

Langkah-langkah dalam proses *insert after*:



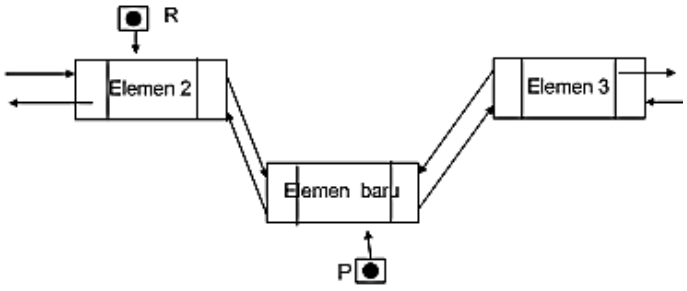

```
P = alokasi(X);
next(P) = Nil;
prev(P) = Nil;
```

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




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

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




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

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

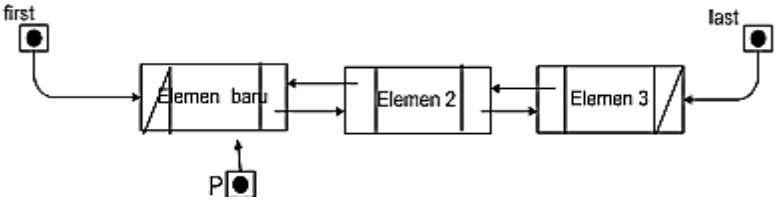
D. *Insert Before*

Diatas 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

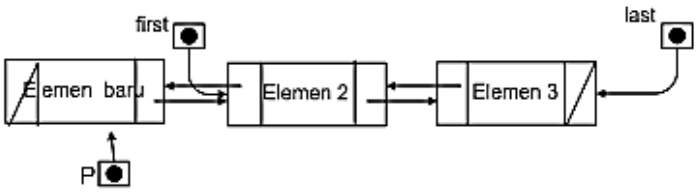
A. *Delete First*

Langkah-langkah dalam proses *delete first*:



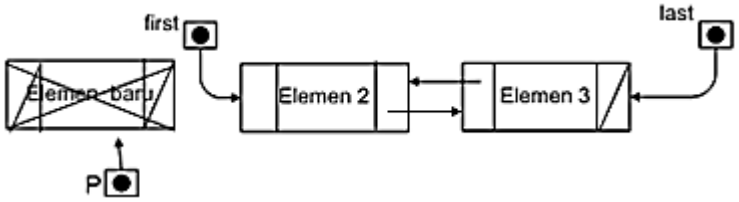

```
P = first(L);
```

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




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

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



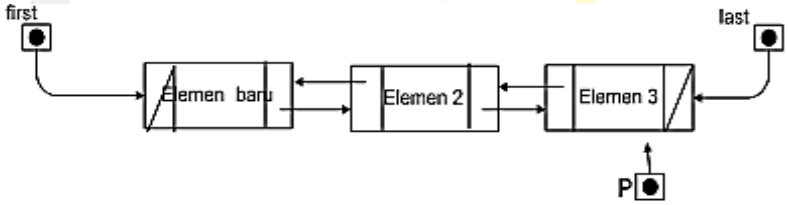

```
prev(first(L)) = Nil;
next(P) = Nil;
return P;
          atau
dealokasi(P);
```

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

```
/* contoh sintak delet first */
void deleteFirst(list &L, address &P){
    P = first(L);
    first(L) = next(first(L));
    prev (P) = null;
    prev(first(L)) = null;
    next(P) = null;
}
```

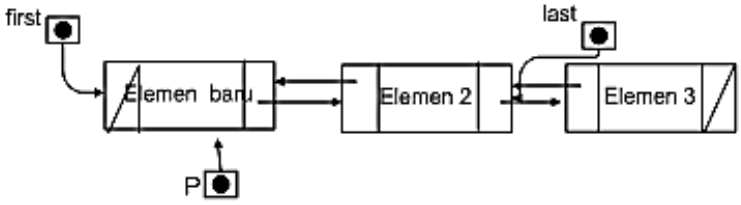
B. *Delete Last*

Langkah-langkah dalam proses *delete last*:



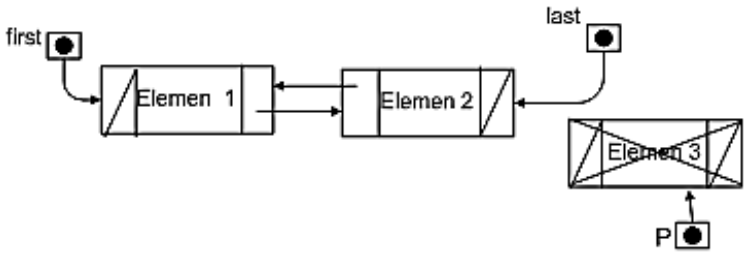

```
P = last(L);
```

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




```
last(L) = prev(last(L));
```

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

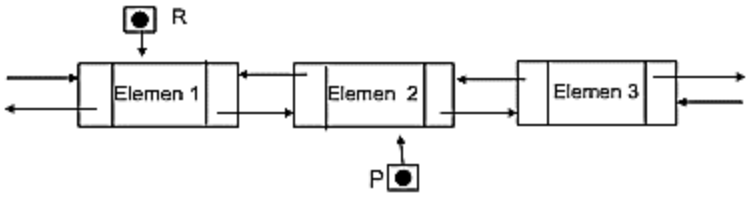



```
prev(P) = Nil;
next(last(L)) = Nil;
return P;
          atau
dealokasi(P);
```

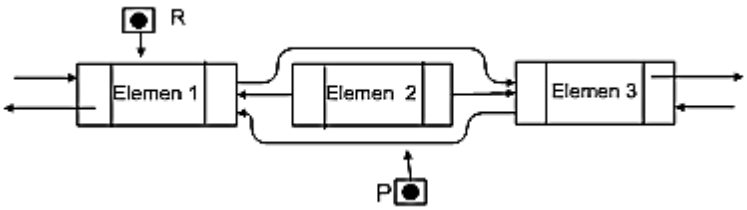
Gambar 6-19 *Double Linked list Delete Last* 3

C. *Delete After*

Langkah-langkah dalam proses *delete after*:

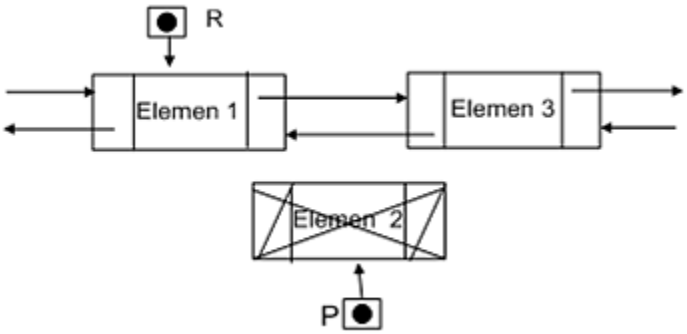


Gambar 6-20 *Double Linked list Delete After* 1




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

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




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

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

D. *Delete Before*

Diatas 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 *single linked list*. Hanya saja pada *double linked list* lebih mudah dalam melakukan proses akses elemen, karena bisa melakukan iterasi maju dan mundur.

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

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


6.2 *Latihan*

1. Buatlah ADT *Double Linked list* sebagai berikut di dalam *file* **"doublelist.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
>
prosedur CreateList( in/out L : List )
fungsi alokasi( x : infotype ) : address
prosedur dealokasi( in/out P : address )
prosedur printInfo( in L : List )
prosedur insertLast( in/out L : List, in P : address )
```

Buatlah implementasi ADT *Double Linked list* pada *file* "doublelist.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 prosedur *delete*.
   - prosedur `deleteFirst( in/out L : List, in/out P : address )`
   - prosedur `deleteLast( in/out L : List, in/out P : address )`
   - prosedur `deleteAfter( in Prec : address, in/out: 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