Modul 4 SINGLE LINKED LIST (BAGIAN PERTAMA)

TUJUAN PRAKTIKUM
2. Memahami penggunaan *linked list* dengan *pointer* operator- operator dalam program. 3. Memahami operasi-operasi dasar dalam *linked list*. 4. Membuat program dengan menggunakan *linked list* dengan *prototype* yang ada

4.1 *Linked List dengan Pointer*

Linked list (biasa disebut *list* saja) adalah salah satu bentuk struktur data (representasi penyimpanan) berupa serangkaian elemen data yang saling berkait (berhubungan) dan bersifat fleksibel karena dapat tumbuh dan mengerut sesuai kebutuhan. Data yang disimpan dalam *Linked list* bisa berupa data tunggal atau data majemuk. Data tunggal merupakan data yang hanya terdiri dari satu data (variabel), misalnya: nama bertipe *string*. Sedangkan data majemuk merupakan sekumpulan data (*record*) yang di dalamnya terdiri dari berbagai tipe data, misalnya: Data Mahasiswa, terdiri dari Nama bertipe *string*, NIM bertipe *long integer*, dan Alamat bertipe *string*.

Linked list dapat diimplementasikan menggunakan *Array* dan *Pointer* (*Linked list*).

Yang akan kita gunakan adalah *pointer*, karena beberapa alasan, yaitu :

1. *Array* bersifat statis, sedangkan *pointer* dinamis.
2. Pada *linked list* bentuk datanya saling bergandengan (berhubungan) sehingga lebih mudah memakai *pointer*.
3. Sifat *linked list* yang fleksibel lebih cocok dengan sifat *pointer* yang dapat diatur sesuai kebutuhan.
4. Karena *array* lebih susah dalam menangani *linked list*, sedangkan *pointer* lebih mudah.
5. *Array* lebih cocok pada kumpulan data yang jumlah elemen maksimumnya sudah diketahui dari awal.

Fakultas Informatika
School of Computing
Telkom University

Dalam implementasinya, pengaksesan elemen pada *Linked list* dengan *pointer* bisa menggunakan (->) atau tanda titik (.).

Model-model dari ADT *Linked list* yang kita pelajari adalah :

1. *Single Linked list*
2. *Double Linked list*
3. *Circular Linked list*
4. *Multi Linked list*
5. *Stack* (Tumpukan)
6. *Queue* (Antrian)
7. *Tree*
8. *Graph*

Setiap model ADT *Linked list* di atas memiliki karakteristik tertentu dan dalam penggunaannya disesuaikan dengan kebutuhan.

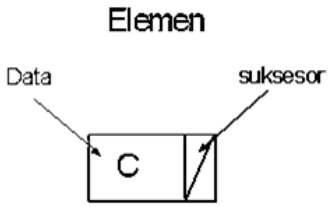
Secara umum operasi-operasi ADT pada *Linked list*, yaitu :

1. Penciptaan dan inisialisasi *list* (*Create List*).
2. Penyisipan elemen *list* (*Insert*).
3. Penghapusan elemen *list* (*Delete*).
4. Penelusuran elemen *list* dan menampilkannya (*View*).
5. Pencarian elemen *list* (*Searching*).
6. Pengubahan isi elemen *list* (*Update*).

4.2 *Single Linked List*

Single Linked list merupakan model ADT *Linked list* yang hanya memiliki satu arah *pointer*.

Komponen elemen dalam *single linked list*:



Gambar 4-1 Elemen *Single Linked list*

Keterangan:
Elemen: segmen-segmen data yang terdapat dalam suatu *list*.
Data: informasi utama yang tersimpan dalam sebuah elemen.
Suksesor: bagian elemen yang berfungsi sebagai penghubung antar elemen.

Sifat dari *Single Linked list*:

1. Hanya memerlukan satu buah *pointer*.
2. *Node* akhir menunjuk ke Nil kecuali untuk *list circular*.
3. Hanya dapat melakukan pembacaan maju.
4. Pencarian sequensial dilakukan jika data tidak terurut.
5. Lebih mudah ketika melakukan penyisipan atau penghapusan di tengah *list*.

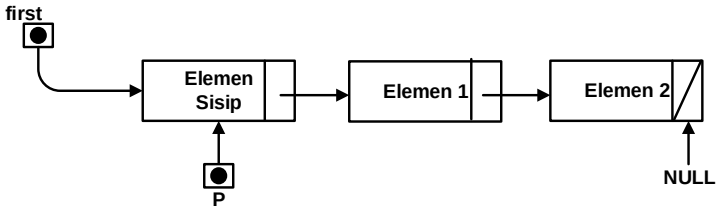
Istilah-istilah dalam *Single Linked list* :

1. *first*/*head*: *pointer* pada *list* yang menunjuk alamat elemen pertama *list*.
2. *next*: *pointer* pada elemen yang berfungsi sebagai *successor* (penunjuk) alamat elemen di depannya.
3. Null/Nil: artinya tidak memiliki nilai, atau tidak mengacu ke mana pun, atau kosong.
4. *Node*/simpul/elemen: merupakan tempat penyimpanan data pada suatu memori tertentu.

Gambaran sederhana *single linked list* dengan elemen kosong:

Gambar 4-2 *Single Linked list* dengan Elemen Kosong

Gambaran sederhana *single linked list* dengan 3 elemen:



Gambar 4-3 *Single Linked list* dengan 3 Elemen

Contoh deklarasi struktur data *single linked list*:

```
1   /*file : list.h*/
2   #ifndef LIST_H_INCLUDED
3   #define LIST_H_INCLUDED
4
5   #define Nil NULL
6   #define info(P) (P)->info
7   #define next(P) (P)->next
8   #define first(L) ((L).first)
9   using namespace std;
10  /*deklarasi record dan struktur data list*/
11  typedef int infotype;
12  typdef struct elmlist *address;
13  struct elmlist {
14     infotype info;
15     address next;
16  };
17
18  struct list{
19     address first;
20  };
21  #endif // TEST_H_INCLUDED
```

Misal untuk data mahasiswa yang terdiri dari nama dan nim.

```
1   /*file : list.h*/
2   #ifndef LIST_H_INCLUDED
3   #define LIST_H_INCLUDED
4
5   #define Nil NULL
6   #define info(P) (P)->info
7   #define next(P) (P)->next
8   #define first(L) ((L).first)
9
10  using namespace std;
11  /*deklarasi record dan struktur data list*/
12  struct mahasiswa{
13     char nama[30]
14     char nim[10]
15  }
16  typedef mahasiswa infotype;
17
18  typdef struct elmlist *address;
19  struct elmlist {
20     infotype info;
21     address next;
22  };
23
24  struct list{
25     address first;
26  };
27  #endif // TEST_H_INCLUDED
```

Informatika School of Computing Telkom University
informatics lab

4.2.1 Pembentukan Komponen-Komponen *List*

A. Pembentukan *List*

Adalah sebuah proses untuk membetuk sebuah *list* baru. Biasanya nama fungsi yang digunakan `createList()`. Fungsi ini akan mengeset nilai awal *list* yaitu *first*(*list*) dan *last*(*list*) dengan nilai Nil.

B. Pengalokasian Memori

Adalah proses untuk mengalokasikan memori untuk setiap elemen data yang ada dalam *list*. Fungsi yang biasanya digunakan adalah nama fungsi yang biasa digunakan `alokasi()`.

Sintak alokasi pada C:

```
P = (address) malloc ( sizeof (elmlist));
```

Keterangan:

P	= variabel *pointer* yang mengacu pada elemen yang dialokasikan.
address	= tipe data *pointer* dari tipe data elemen yang akan dialokasikan.
Elmlist	= tipe data atau *record* elemen yang dialokasikan.

Contoh deklarasi struktur data *single linked list*:
Misal untuk data mahasiswa yang terdiri dari nama dan nim.

```
address alokasi(mahasiswa m){
     address p = (address)malloc(sizeof(elmlist));
     info(p) = m;
     return p;
}
```

Namun pada Cpp. Penggunaan **malloc** dapat dipersingkat menggunakan sintak **new.**

Sintak alokasi pada Cpp:

```
P = new elmlist;
```

Keterangan:

P	= variabel *pointer* yang mengacu pada elemen yang dialokasikan.
address	= tipe data *pointer* dari tipe data elemen yang akan dialokasikan.

Contoh deklarasi struktur data *single linked list*:
Misal untuk data mahasiswa yang terdiri dari nama dan nim.

```
address alokasi(mahasiswa m){
     address p = new elmlist;
     info(p) = m;
     return p;
}
```

C. Dealokasi

Untuk menghapus sebuah *memory address* yang tersimpan atau telah dialokasikan dalam bahasa pemrograman C digunakan sintak ***free***, sedangkan pada Cpp digunakan sintak ***delete***, seperti berikut.

Sintak pada C:

```
free( p );
```

Sintak pada Cpp:

```
delete p;
```

D. Pengecekan *List*

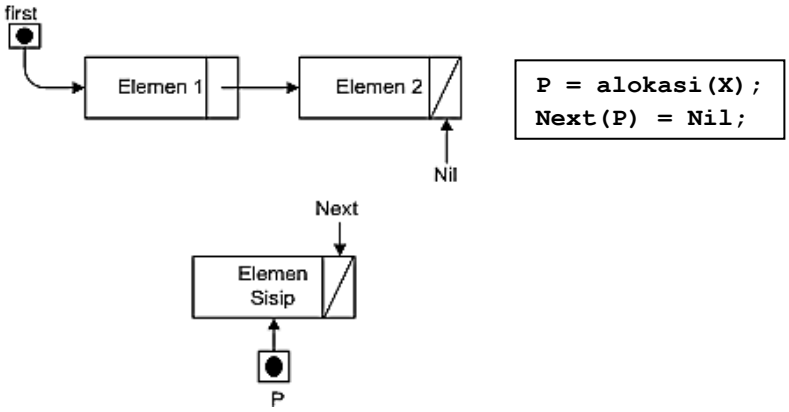
Adalah fungsi untuk mengecek apakah *list* tersebut kosong atau tidak. Akan mengembalikan nilai true jika *list* kosong dan nilai *false* jika *list* tidak kosong. Fungsi yang digunakan adalah `isEmpty()`.

4.2.2 Insert

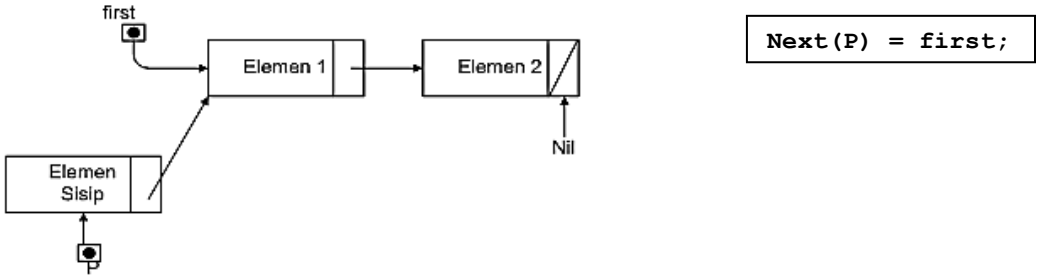
A. *Insert First*

Merupakan metode memasukkan elemen data ke dalam *list* yang diletakkan pada awal *list*.

Langkah-langkah dalam proses *insert first*:

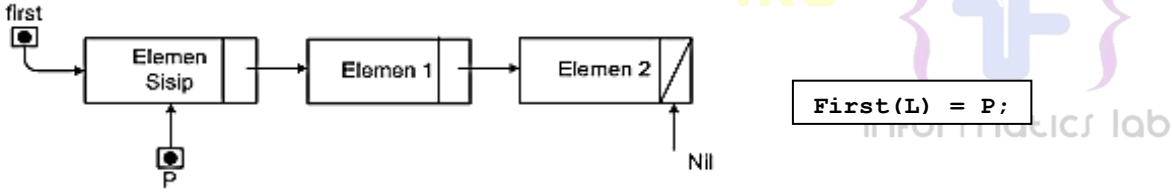


Gambar 4-4 *Single Linked list Insert First* (1)



Gambar 4-5 *Single Linked list Insert First* (2)

Fakultas Informatika



Informatics lab

Gambar 4-6 *Single Linked list Insert First* (3)

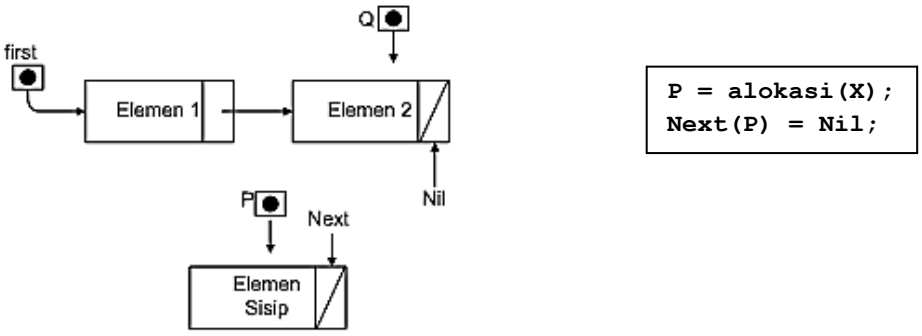
```
/* contoh syntax insert first */
void insertFirst(List &L, address &P){
    next (P) = first(L);
    first(L) = P;

}
```

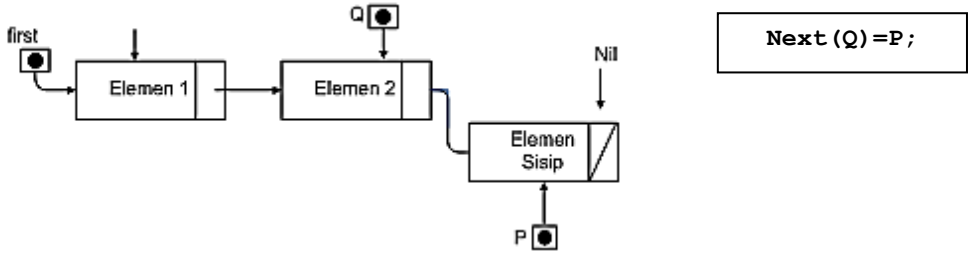
B. *Insert Last*

Merupakan metode memasukkan elemen data ke dalam *list* yang diletakkan pada akhir *list*.

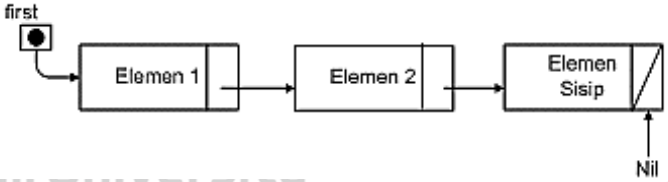
Langkah dalam *insert last* :



Gambar 4-7 *Single Linked list Insert Last* 1



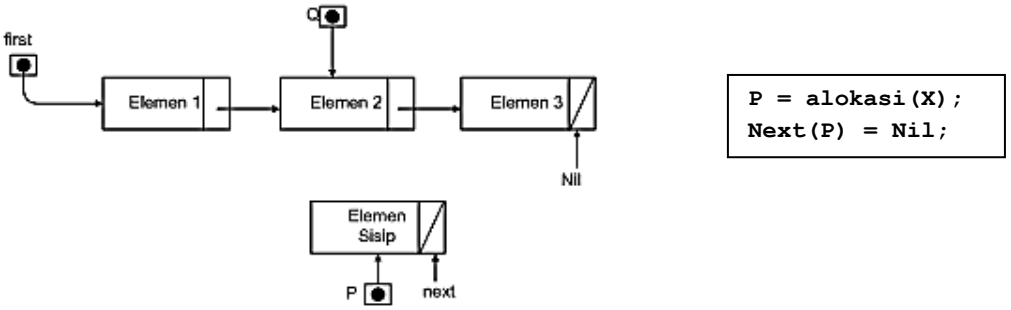
Gambar 4-8 *Single Linked list Insert Last* 2



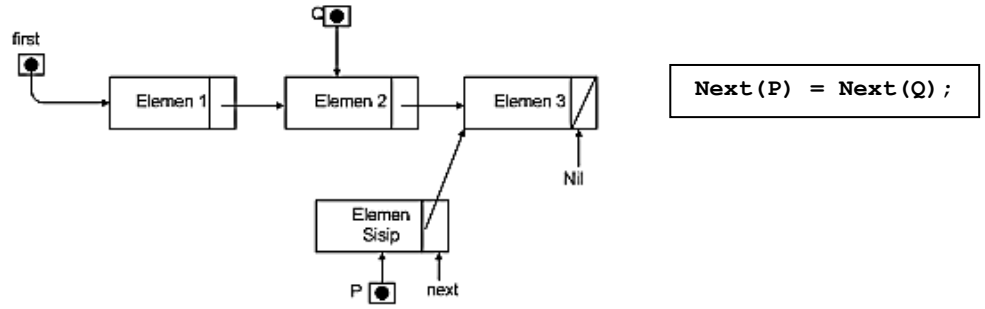
Gambar 4-9 *Single Linked list Insert Last* 3

C. *Insert After*

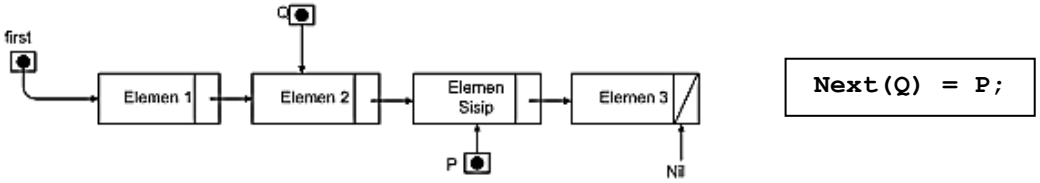
Merupakan metode memasukkan data ke dalam *list* yang diletakkan setelah *node* tertentu yang ditunjuk oleh *user*. Langkah dalam *insert after*:



Gambar 4-10 *Single Linked list Insert After* 1



Gambar 4-11 *Single Linked list Insert After* 2



Gambar 4-12 *Single Linked list Insert After* 3

4.2.3 View

Merupakan operasi dasar pada *list* yang menampilkan isi *node*/simpul dengan suatu penelusuran *list*. Mengunjungi setiap *node* kemudian menampilkan data yang tersimpan pada *node* tersebut.

Semua fungsi dasar diatas merupakan bagian dari ADT dari singgle *linked list*, dan aplikasi pada bahasa pemrograman Cp semua ADT tersebut tersimpan dalam *file* *.c dan *file* *.h.

```
1   /*file : list .h*/
2   /* contoh ADT list berkait dengan representasi fisik pointer*/
3   /* representasi address    dengan pointer*/
4   /* info tipe adalah integer */
5   #ifndef list_H
6   #define list_H
7   #include "boolean.h"
8   #include <stdio.h>
9   #define Nil NULL
10  #define info(P) (P)->info
11  #define next(P) (P)->next
12  #define first(L) ((L).first)
13
14  /*deklarasi record dan struktur data list*/
15  typedef int infotype;
16  typedef struct elmlist *address;
17  struct elmlist{
18      infotype info;
19      address next;
20  };
21
22  /* definisi list : */
23  /* list kosong jika First(L)=Nil */
24  /* setiap elemen address P dapat diacu info(P) atau next(P) */
25  struct list {
26      address first;
27  };
28  /********* pengecekan apakah list kosong ***********/
29  boolean ListEmpty(list L);
30  /*mengembalikan nilai true jika list kosong*/
31
32  /******** pembuatan list kosong *********/
33  void CreateList(list &L);
```

```
34  /* I.S. sembarang
35     F.S. terbentuk list kosong*/
36
37  /********* manajemen memori *********/
38  void dealokasi(address P);
39  /* I.S. P terdefinisi
40     F.S. memori yang digunakan P dikembalikan ke sistem */
41
42  /********* penambahan elemen **********/
43  void insertFirst(list &L, address P);
44  /* I.S. sembarang, P sudah dialokasikan
45     F.S. menempatkan elemen beralamat P pada awal list */
46
47  void insertAfter(list &L, address P, address Prec);
48  /* I.S. sembarang, P dan Prec alamt salah satu elemen list
49     F.S. menempatkan elemen beralamat P sesudah elemen beralamat Prec */
50
51  void insertLast(list &L, address P);
52  /* I.S. sembarang, P sudah dialokasikan
53     F.S. menempatkan elemen beralamat P pada akhir list */
54
55  /********** proses semau elemen list *********/
56  void printInfo(list L);
57  /* I.S. list mungkin kosong
58     F.S. jika list tidak kosong menampilkan semua info yang ada pada list */
59
60  int nbList(list L);
61  /* mengembalikan jumlah elemen pada list */
62
63  #endif
```

4.3 *Latihan*

1. Buatlah ADT *Single Linked list* sebagai berikut di dalam *file* **"singlelist.h"**:

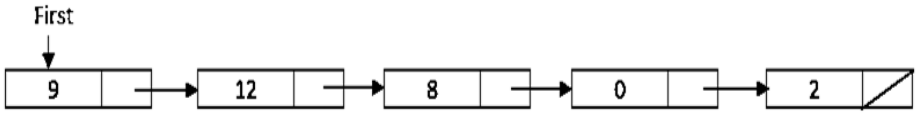
```
Type infotype : int
Type address  : pointer to ElmList

Type ElmList <
   info : infotype
   next : address
>

Type List : < First : address >

prosedur CreateList( in/out L : List )
fungsi alokasi( x : infotype ) : address
prosedur dealokasi( in/out P : address )
prosedur printInfo( in L : List )
prosedur insertFirst( in/out L : List, in P : address )
```

Kemudian buat implementasi ADT *Single Linked list* pada *file* **"singlelist.cpp".**
Adapun isi data



Gambar 4-13 Ilustrasi elemen

Cobalah hasil implementasi ADT pada *file* "**main.cpp**"

```
int main()
{
    List L;
    address P1, P2, P3, P4, P5 = NULL;
    createList(L);

    P1 = alokasi(2);
    insertFirst(L,P1);

    P2 = alokasi(0);
    insertFirst(L,P2);

    P3 = alokasi(8);
    insertFirst(L,P3);

    P4 = alokasi(12);
    insertFirst(L,P4);

    P5 = alokasi(9);
    insertFirst(L,P5);

    printInfo(L)
    return 0;
}
```

```
9 12 8 0 2

Process returned 0 (0x0)   execution time : 0.019 s
Press any key to continue.
```

Gambar 4-14 *Output* singlelist

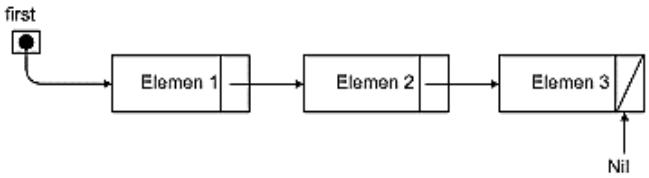
Fakultas Informatika
School of Computing
Telkom University
informatics lab

4.4 *Delete*

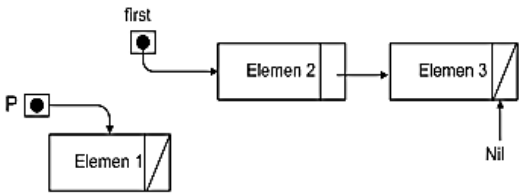
A. *Delete First*

Adalah pengambilan atau penghapusan sebuah elemen pada awal *list*.

Langkah-langkah dalam *delete first*:

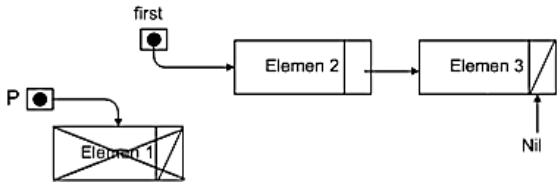


Gambar 4-15 *Single Linked List Delete First* 1




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

Gambar 4-16 *Single Linked list Delete First* 2




```
Return(P);
    Atau
Dealokasi(P);
```

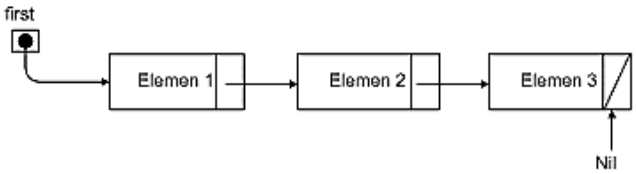
Gambar 4-17 *Single Linked list Delete First* 3

```
/* contoh syntax delete first */
void deleteFirst(List &L, address &P){
    P = first(L);
    first(L) = next(first(L));
    next (P) = null;
}
```

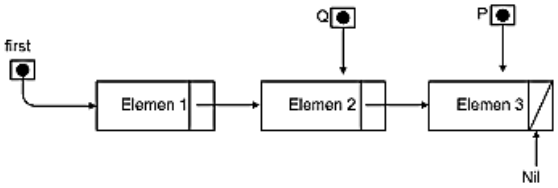
B. *Delete Last*

Merupakan pengambilan atau penghapusan suatu elemen dari akhir *list*.

Langkah-langkah dalam *delete last*:

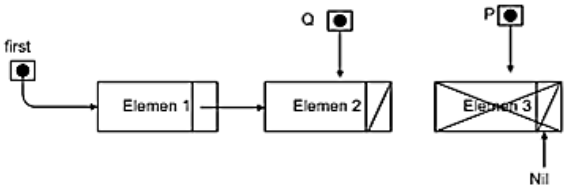


Gambar 4-18 *Single Linked list Delete Last* 1




```
P = getLast(L);
Q = findBefore(P);
```

Gambar 4-19 *Single Linked list Delete Last* 2



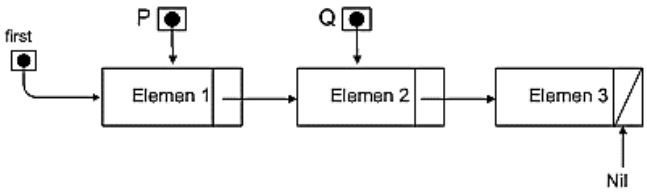

```
Next(Q) = Nil;
return(P);
    Atau
dealokasi(P);
```

Gambar 4-20 *Single Linked list Delete Last* 3

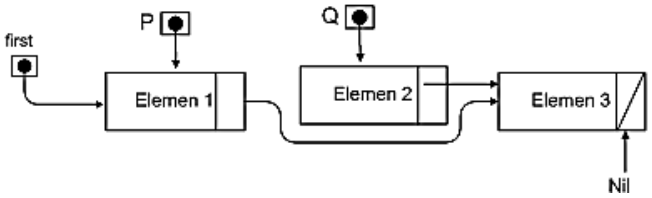
C. *Delete After*

Merupakan pengambilan atau penghapusan *node* setelah *node* tertentu.

Langkah-langkah dalam *delete after*:

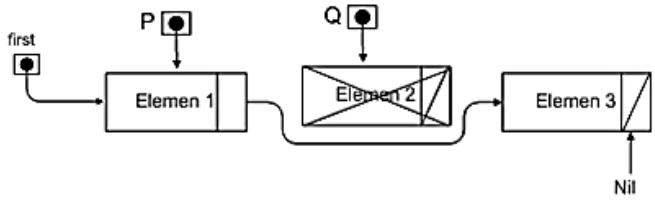


Gambar 4-21 *Single Linked list Delete After* 1




```
Next(P) = Next(Q);
```

Gambar 4-22 *Single Linked list Delete After* 2




```
Next(Q) = Nil;
return(P);
      Atau
dealokasi(P);
```

Gambar 4-23 *Single Linked list Delete After* 3

D. *Delete* Elemen

Adalah operasi yang digunakan untuk menghapus dan membebaskan memori yang dipakai oleh elemen tersebut.

Fungsi yang biasanya dipakai:
1. fungsi dealokasi(P) : membebaskan memori yang dipakai oleh elemen P.
2. fungsi delAll(L) : membebaskan semua memori yang dipakai elemen – elemen yang ada pada *list* L. Hasil akhir *list* L menjadi kosong.

Semua operasi-operasi dasar *list* biasa disebut dengan operasi primitif. Primitif-primitif dalam *list* ini merupakan bagian dari ADT *list* yang tersimpan dalam *file* *.h dan *file* *.cpp, dengan rincian *file* *.h untuk menyimpan prototipe primitif-primitif atau fungsi-fungsi dan menyimpan tipe data yang dipergunakan dalam primitif *list* tersebut.

Untuk bisa mengakses semua primitif tersebut yaitu dengan meng-*include* terhadap *file* *.h-nya.

4.5 *Update*

Merupakan operasi dasar pada *list* yang digunakan untuk mengupdate data yang ada di dalam *list*. Dengan operasi *update* ini kita dapat meng-*update* data-data *node* yang ada di dalam *list*. Proses *update* biasanya diawali dengan proses pencarian terhadap data yang akan di-*update*.