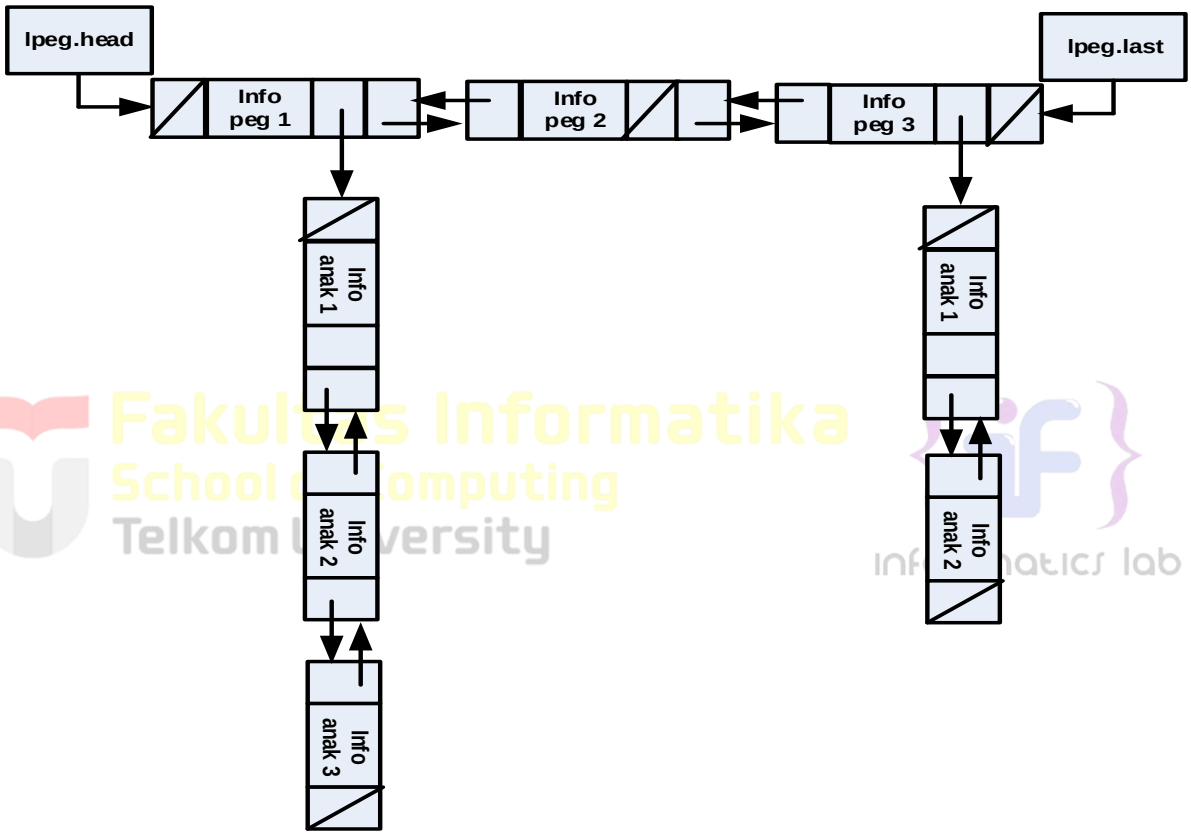
Modul 13 MULTI LINKED LIST

TUJUAN PRAKTIKUM
1. Memahami penggunaan *Multi Linked list*. 2. Mengimplementasikan *Multi Linked list* dalam beberapa studi kasus.

13.1 *Multi Linked List*

Multi List merupakan sekumpulan *list* yang berbeda yang memiliki suatu keterhubungan satu sama lain. Tiap elemen dalam *multi link list* dapat membentuk *list* sendiri.Biasanya ada yang bersifat sebagai *list* induk dan *list* anak .

Contoh *Multi Linked list* dapat dilihat pada gambar berikut.



Gambar 13-1 Multi *Linked list*

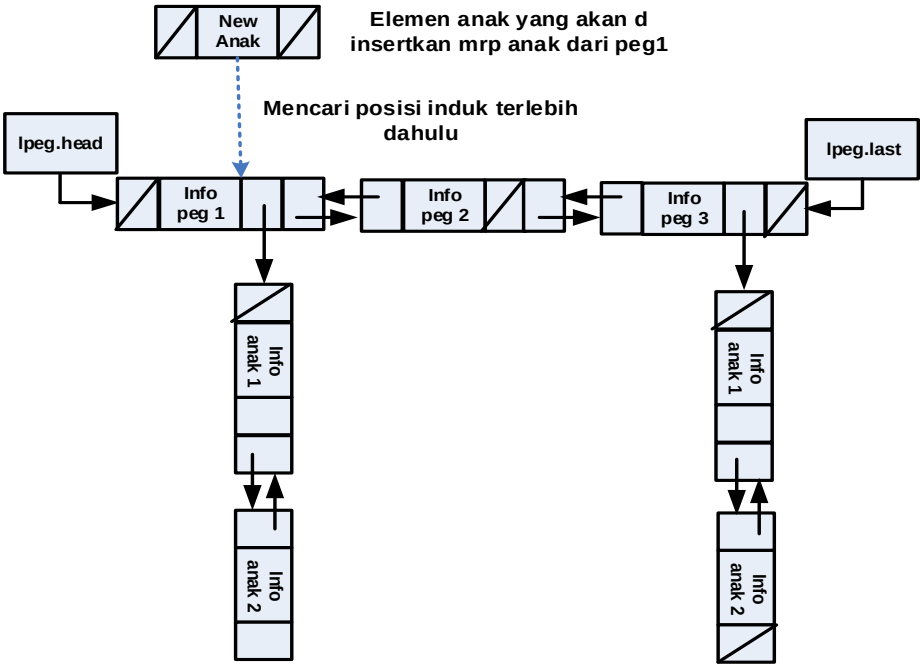
Jadi , dari implementasi di atas akan terdapat dua buah *list*, *list* pegawai dan *list* anak. Dimana untuk *list* pegawai menunjuk satu buah *list* anak. Disini *list* induknya adalah *list* pegawai dan *list* anaknya adalah *list* anak.

13.1.1 Insert

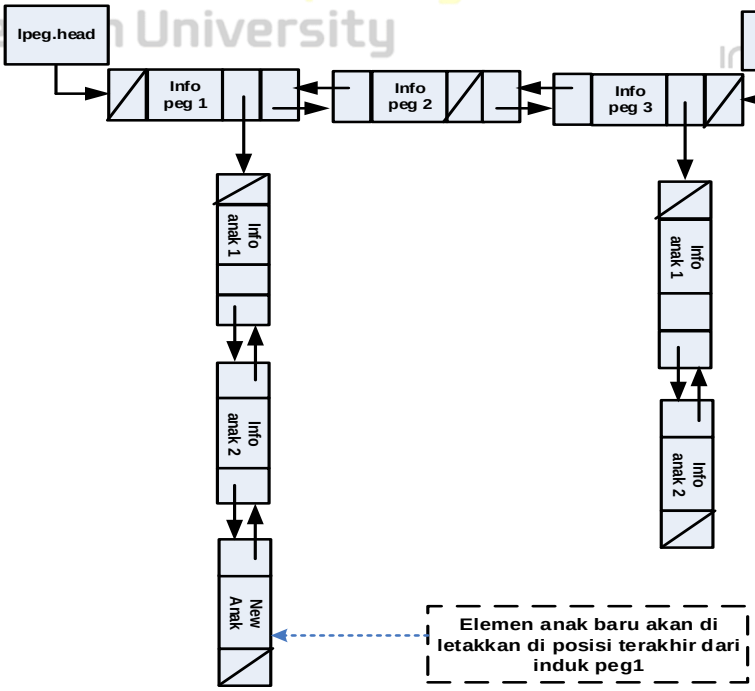
A. *Insert* Anak

Dalam penambahan elemen anak harus diketahui dulu elemen induknya.

Berikut ini ilustrasi *insert* anak dengan konsep *insert last*:



Gambar 13-2 *Multi Linked list Insert* Anak 1



Gambar 13-3 *Multi Linked list Insert* Anak 2

```
/* buat dahulu elemen yang akan disisipkan */
address_anak alokasiAnak(infotypeanak X){
```

```
    address_anak p = alokasi(X);
    next(p) = null;
    prev(p) = null;
    return p;
}
/* mencari apakah ada elemen pegawai dengan info X */
address findElm(listinduk L, infotypeinduk X){
    address cariInduk = head(L);
    do{
        if(cariInduk.info == X){
            return cariInduk;
        }else{
            cariInduk = next(cariInduk);
        }
    }while(cariInduk.info!=X || cariInduk!=last(L))
}
/* menyisipkan anak pada akhir list anak */
void insertLastAnak(listanak &Lanak, address_anak P){
    address_anak ! = head(&Lanak);
    do{
        Q = next(Q);
    }while(next(&Lanak)!=NULL)
    next(Q) = P;
    prev(P) = Q;
    next(P) = NULL;
}
```

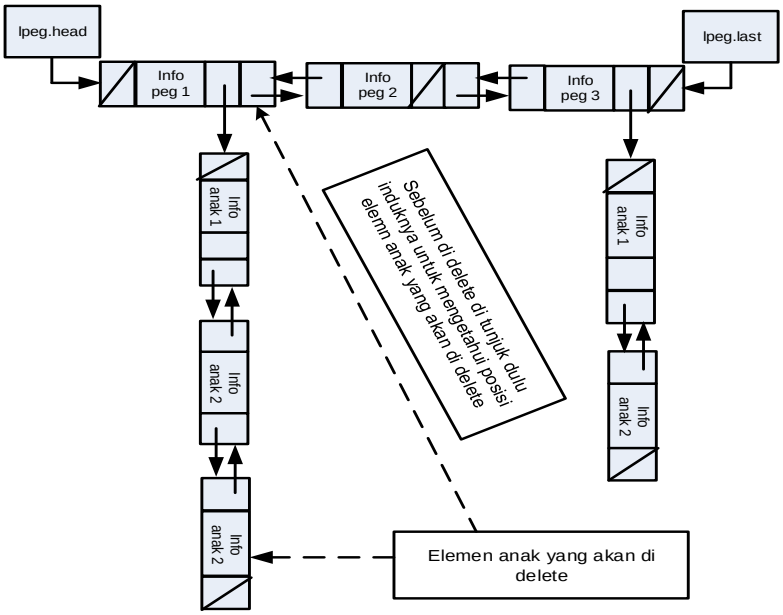
B. *Insert* Induk

Untuk *insert* elemen induk sama dengan konsep *insert* pada *single*, *double* dan *circular linked list*.

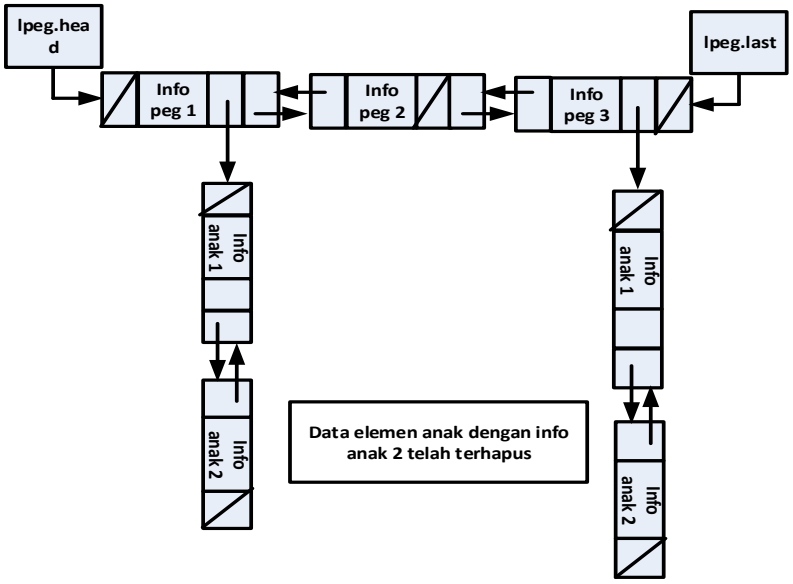
13.1.2 Delete

A. *Delete* Anak

Sama dengan *insert* anak untuk *delete* anak maka harus diketahui dulu induknya.Berikut ini Gambar ilustrasinya untuk *delete last* pada induk peg 1:



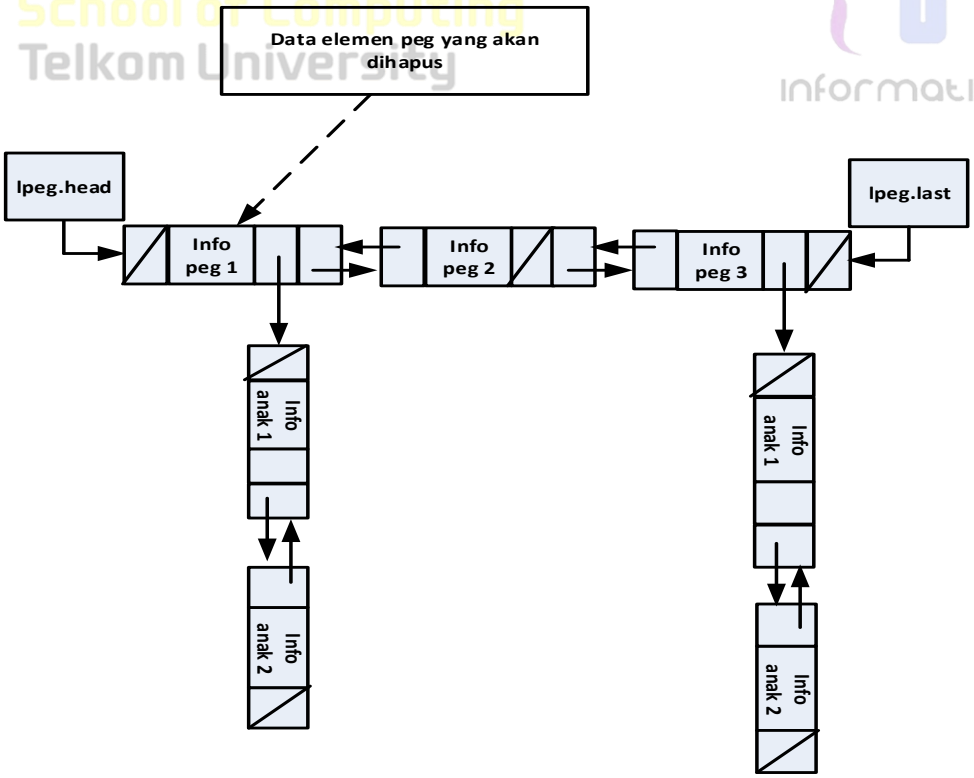
Gambar 13-4 *Multi Linked list Delete* Anak 1



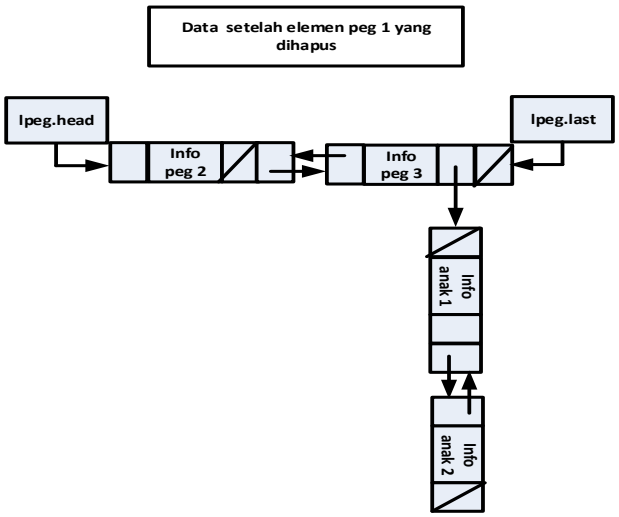
Gambar 13-5 *Multi Linked list Delete* Anak 2

B. *Delete* Induk

Untuk *delete* elemen induk maka saat di hapus maka seluruh anak dengan induk tersebut juga harus dihapus. Berikut ini gambar ilustrasinya:



Gambar 13-6 *Multi Linked list Delete* Induk 1



Gambar 13-7 *Multi Linked list Delete* Induk 2

```
1   /*file : multilist .h*/
2   /* contoh ADT list berkait dengan representasi fisik pointer*/
3   /* representasi address    dengan pointer*/
4
5   /* info tipe adalah integer */
6   #ifndef MULTILIST_H_INCLUDED
7   #define MULTILIST_H_INCLUDED
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  #define last(L) ((L).last)
14
15  typedef int infotypeanak;
16  typedef int infotypeinduk;
17  typedef struct elemen_list_induk *address;
18  typedef struct elemen_list_anak *address_anak;
19  /* define list : */
20
21  /* list kosong jika first(L)=Nil
22  setiap elemen address P dapat diacu info(P) atau next(P)
23  elemen terakhir list jika addressnya last, maka next(last) = Nil */
24  struct elemen_list_anak{
25  /* struct ini untuk menyimpan elemen anak dan pointer penunjuk
26     elemen tetangganya */
27      infotypeanak info;
28      address_anak next;
29      address_anak prev;
30  };
31
32  struct listanak {
33  /* struct ini digunakan untuk menyimpan list anak itu sendiri */
34      address_anak first;
35      address_anak last;
36  };

37  struct elemen_list_induk{
38  /* struct ini untuk menyimpan elemen induk dan pointer penunjuk
39     elemen tetangganya */
40      infotypeanak info;
41      listanak lanak;
42      address next;
43      address prev;
```

```
};
struct listinduk {
/* struct ini digunakan untuk menyimpan list induk itu sendiri */
    address first;
    address last;
};

/********* pengecekan apakah list kosong ***********/
boolean ListEmpty(listinduk L);
/*mengembalikan nilai true jika list induk kosong*/
boolean ListEmptyAnak(listanak L);
/*mengembalikan nilai true jika list anak kosong*/

/********* pembuatan list kosong *********/
void CreateList(listinduk &L);
/* I.S. sembarang
   F.S. terbentuk list induk kosong*/
void CreateListAnak(listanak &L);
/* I.S. sembarang
   F.S. terbentuk list anak kosong*/

/********* manajemen memori *********/
address alokasi(infotypeinduk P);
/* mengirimkan address dari alokasi sebuah elemen induk
   jika alokasi berhasil, maka nilai address tidak Nil dan jika gagal
   nilai address Nil */

address_anak alokasiAnak(infotypeanak P);
/* mengirimkan address dari alokasi sebuah elemen anak
   jika alokasi berhasil, maka nilai address tidak Nil dan jika gagal
   nilai address_anak Nil */

void dealokasi(address P);
/* I.S. P terdefinisi
   F.S. memori yang digunakan P dikembalikan ke sistem */

void dealokasiAnak(address_anak P);
/* I.S. P terdefinisi
   F.S. memori yang digunakan P dikembalikan ke sistem */
/********* pencarian sebuah elemen list *********/
address findElm(listinduk L, infotypeinduk X);
/* mencari apakah ada elemen list dengan info(P) = X
   jika ada, mengembalikan address elemen tab tsb, dan Nil jika sebaliknya
*/
address_anak findElm(listanak Lanak, infotypeanak X);
/* mencari apakah ada elemen list dengan info(P) = X
   jika ada, mengembalikan address elemen tab tsb, dan Nil jika sebaliknya
*/
boolean fFindElm(listinduk L, address P);
/* mencari apakah ada elemen list dengan alamat P
   mengembalikan true jika ada dan false jika tidak ada */
boolean fFindElmanak(listanak Lanak, address_anak P);
/* mencari apakah ada elemen list dengan alamat P
   mengembalikan true jika ada dan false jika tidak ada */

address findBefore(listinduk L, address P);
/* mengembalikan address elemen sebelum P
   jika P berada pada awal list, maka mengembalikan nilai Nil */
address_anak findBeforeAnak(listanak Lanak, infotypeinduk X, address_anak
P);
/* mengembalikan address elemen sebelum P dimana info(P) = X
   jika P berada pada awal list, maka mengembalikan nilai Nil */

/********* penambahan elemen **********/
void insertFirst(listinduk &L, address P);
/* I.S. sembarang, P sudah dialokasikan
   F.S. menempatkan elemen beralamat P pada awal list */
```

```
111
112  void insertAfter(listinduk &L, address P, address Prec);
113  /* I.S. sembarang, P dan Prec alamt salah satu elemen list
114     F.S. menempatkan elemen beralamat P sesudah elemen beralamat Prec */
115
116  void insertLast(listinduk &L, address P);
117  /* I.S. sembarang, P sudah dialokasikan
118     F.S. menempatkan elemen beralamat P pada akhir list */
119
120  void insertFirstAnak(listanak &L, address_anak P);
121  /* I.S. sembarang, P sudah dialokasikan
122     F.S. menempatkan elemen beralamat P pada awal list */
123
124  void insertAfterAnak(listanak &L, address_anak P, address_anak Prec);
125  /* I.S. sembarang, P dan Prec alamt salah satu elemen list
126     F.S. menempatkan elemen beralamat P sesudah elemen beralamat Prec */
127
128  void insertLastAnak(listanak &L, address_anak P);
129  /* I.S. sembarang, P sudah dialokasikan
130     F.S. menempatkan elemen beralamat P pada akhir list */
131
132  /********* penghapusan sebuah elemen *********/
133  void delFirst(listinduk &L, address &P);
134  /* I.S. list tidak kosong
135     F.S. adalah alamat dari alamat elemen pertama list
136     sebelum elemen pertama list dihapus
137     elemen pertama list hilang dan list mungkin menjadi kosong
138     first elemen yang baru adalah successor first elemen yang lama */
139  void delLast(listinduk &L, address &P);
140  /* I.S. list tidak kosong
141     F.S. adalah alamat dari alamat elemen terakhir list
142     sebelum elemen terakhir list dihapus
143     elemen terakhir list hilang dan list mungkin menjadi kosong
144     last elemen yang baru adalah successor last elemen yang lama */
145
146  void delAfter(listinduk &L, address &P, address Prec);
147  /* I.S. list tidak kosng, Prec alamat salah satu elemen list
148     F.S. P adalah alamatdari next(Prec), menghapus next(Prec) dari list */
149  void delP (listinduk &L, infotypeinduk X);
150  /* I.S. sembarang
151     F.S. jika ada elemen list dengan alamat P, dimana info(P)=X, maka P
152  dihapus
153     dan P di-dealokasi, jika tidak ada maka list tetap
154     list mungkin akan menjadi kosong karena penghapusan */
155
156  void delFirstAnak(listanak &L, address_anak &P);
157  /* I.S. list tidak kosong
158     F.S. adalah alamat dari alamat elemen pertama list
159     sebelum elemen pertama list dihapus
160     elemen pertama list hilang dan list mungkin menjadi kosong
161     first elemen yang baru adalah successor first elemen yang lama */
162  void delLastAnak(listanak &L, address_anak &P);
163  /* I.S. list tidak kosong
164     F.S. adalah alamat dari alamat elemen terakhir list
165     sebelum elemen terakhir list dihapus
166     elemen terakhir list hilang dan list mungkin menjadi kosong
167     last elemen yang baru adalah successor last elemen yang lama */
168
169  void delAfterAnak(listanak &L, address_anak &P, address_anak Prec);
170  /* I.S. list tidak kosng, Prec alamat salah satu elemen list
171     F.S. P adalah alamatdari next(Prec), menghapus next(Prec) dari list */
172  void delPAnak (listanak &L, infotypeanak X);
173  /* I.S. sembarang
174     F.S. jika ada elemen list dengan alamat P, dimana info(P)=X, maka P
175  dihapus
176     dan P di-dealokasi, jika tidak ada maka list tetap
177     list mungkin akan menjadi kosong karena penghapusan */
```

```
178  /********** proses semau elemen list *********/
179  void printInfo(list L);
180  /* I.S. list mungkin kosong
181     F.S. jika list tidak kosong menampilkan semua info yang ada pada list
182  */
183
184  int nbList(list L);
185  /* mengembalikan jumlah elemen pada list */
186
187  void printInfoAnak(listanak Lanak);
188  /* I.S. list mungkin kosong
189     F.S. jika list tidak kosong menampilkan semua info yang ada pada list
190  */
191
192  int nbListAnak(listanak Lanak);
193  /* mengembalikan jumlah elemen pada list anak */
194
195  /********** proses terhadap list **********/
196  void delAll(listinduk &L);
197  /* menghapus semua elemen list dan semua elemen di-dealokasi */
198
199  #endif
```

13.2 *Latihan*

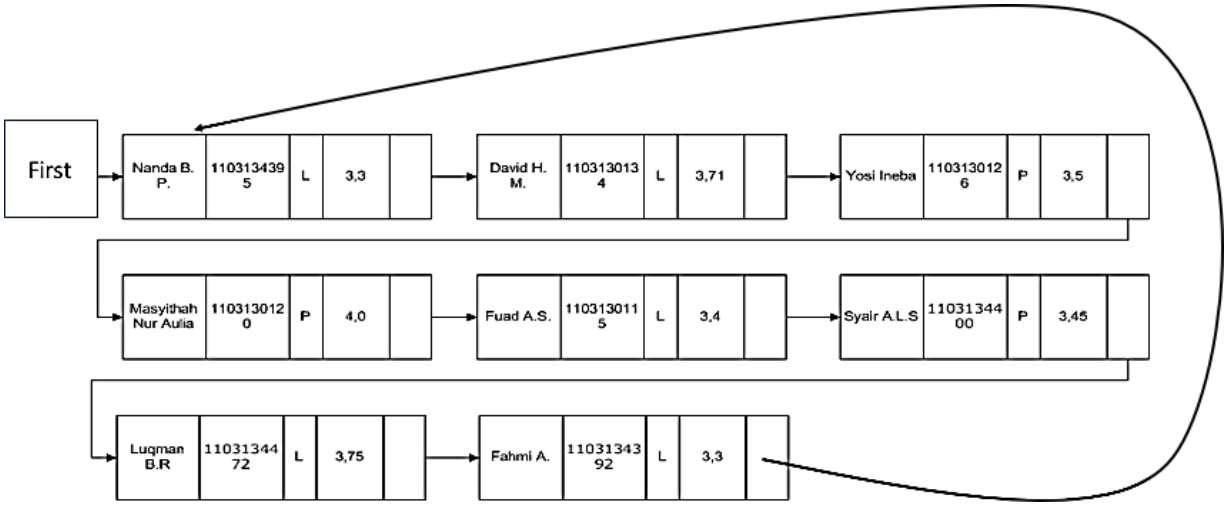
2. Perhatikan program 46 **multilist.h**, buat **multilist.cpp** untuk implementasi semua fungsi pada **multilist.h**. Buat **main.cpp** untuk pemanggilan fungsi-fungsi tersebut.
3. Buatlah ADT *Multi Linked list* sebagai berikut di dalam *file* **"circularlist.h"**:

```
Type infotype : mahasiswa <
    Nama:string
    Nim:string
    Jenis_kelamin:char
    Ipk:float>
Type address : pointer to ElmList
Type ElmList <
    info : infotype
    next :address>
Type List <
    First : address>
```

- Terdapat 11 fungsi/prosedur untuk ADT circularlist
  - prosedur **CreateList**(in/out L : List)
  - fungsi **alokasi**(x : infotype) : address
  - prosedur **dealokasi**(in/out P : address)
  - prosedur **insertFirst**(in/out L : List, in P : address)
  - prosedur **insertAfter**(in/out L : List, in Prec : address, P : address)
  - prosedur **insertLast**(in/out L : List, in P : address)
  - prosedur **deleteFirst**(in/out L : List, in/out P : address)
  - prosedur **deleteAfter**(in/out L : List, in Prec : address, in/out P : address)
  - prosedur **deleteLast**(in/out L : List, in/out P : address)
  - fungsi **findElm**(L : List, x : infotype) : address
  - prosedur **printInfo**(in L : List)

Keterangan :

- fungsi findElm mencari elemen di dalam *list* L berdasarkan nim
  - fungsi mengembalikan elemen dengan dengan info nim == x.nim jika ditemukan
  - fungsi mengembalikan NIL jika tidak ditemukan



Gambar 13-8 Ilustrasi data

Buatlah implementasi ADT *Double Linked list* pada *file* "**circularlist.cpp**"**.** Tambahkan fungsi/prosedur berikut pada *file* **"main.cpp".**

- fungsi **create (in nama, nim : string, jenis_kelamin : char, ipk : float)**
  - **fungsi disediakan**, ketik ulang code yang diberikan
  - fungsi mengalokasikan sebuah elemen *list* dengan info sesuai *input*

```
address createData(string nama, string nim, char jenis_kelamin, float ipk)
{
    /**
    * PR : mengalokasikan sebuah elemen list dengan info dengan info sesuai input
    * FS : address P menunjuk elemen dengan info sesuai input
    */
    infotype x;
    address P;
    x.nama = nama;
    x.nim = nim;
    x.jenis_kelamin = jenis_kelamin;
    x.ipk = ipk;
    P = alokasi(x);
    return P;
}
```

Gambar 13-9 Fungsi *create*

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

```
int main()
{
    List L, A, B, L2;
    address P1 = NULL;
    address P2 = NULL;
    infotype x;
    createList(L);

    cout<<"coba insert first, last, dan after"<<endl;
    P1 = createData("Danu", "04", 'l', 4.0);
    insertFirst(L,P1);

    P1 = createData("Fahmi", "06", 'l',3.45);
    insertLast(L,P1);
    P1 = createData("Bobi", "02", 'l',3.71);
    insertFirst(L,P1);
```

```
    P1 = createData("Ali", "01", 'l', 3.3);
    insertFirst(L,P1);

    P1 = createData("Gita", "07", 'p', 3.75);
    insertLast(L,P1);

    x.nim = "07";
    P1 = findElm(L,x);
    P2 = createData("Cindi", "03", 'p', 3.5);
    insertAfter(L, P1, P2);

    x.nim = "02";
    P1 = findElm(L,x);
    P2 = createData("Hilmi", "08", 'p', 3.3);
    insertAfter(L, P1, P2);

    x.nim = "04";
    P1 = findElm(L,x);
    P2 = createData("Eli", "05", 'p', 3.4);
    insertAfter(L, P1, P2);
    printInfo(L);
    return 0;
}
```

Gambar 13-11 Main.cpp

```
coba insert first, last, dan after
Nama : Ali
NIM  : 01
L/P  : l
IPK  : 3.3

Nama : Bobi
NIM  : 02
L/P  : l
IPK  : 3.71

Nama : Cindi
NIM  : 03
L/P  : p
IPK  : 3.5

Nama : Danu
NIM  : 04
L/P  : l
IPK  : 4

Nama : Eli
NIM  : 05
L/P  : p
IPK  : 3.4

Nama : Fahmi
NIM  : 06
L/P  : l
IPK  : 3.45

Nama : Gita
NIM  : 07
L/P  : p
IPK  : 3.75

Nama : Hilmi
NIM  : 08
L/P  : l
IPK  : 3.3
```

Gambar 13-10 *Output*