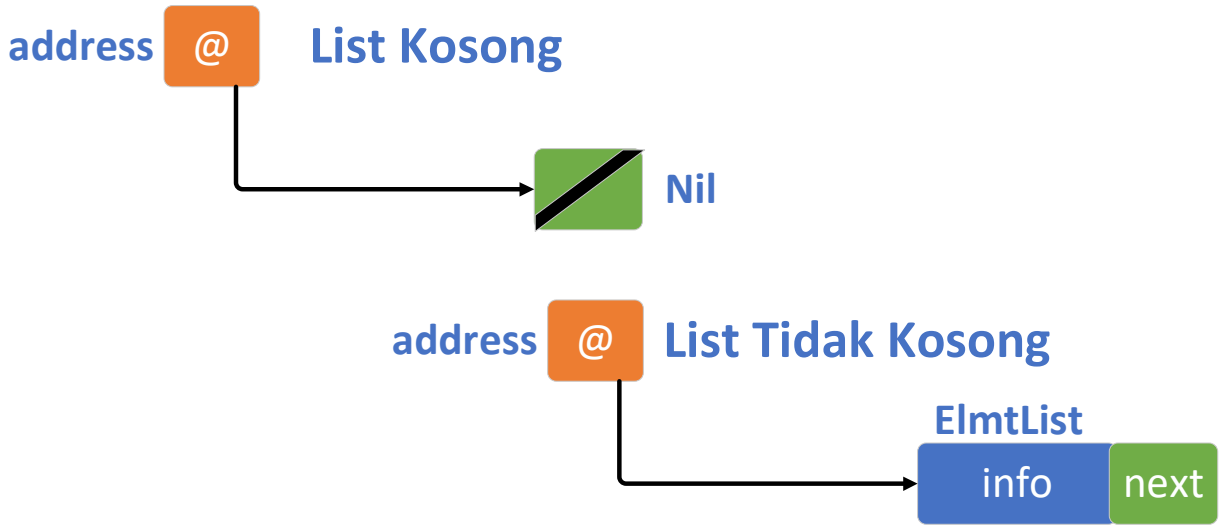
List Linier

Definisi

List linier adalah sekumpulan elemen yang memiliki type sama dan memiliki keterurutan tertentu, dan setiap elemennya terdiri dari dua bagian, yaitu informasi mengenai elemennya, dan informasi mengenai alamat elemen suksesornya (berikutnya). Secara garis besar bahwa list linier didefisikan sebagai berikut:

type **ElmtList** : < **info** : InfoType, **next** : address >



InfoType adalah sebuah type terdefinisi yang menyimpan informasi sebuah elemen list, kemudian next adalah address (alamat) dari elemen berikutnya (suksesor). Dengan demikian, jika didefinisikan First adalah alamat elemen pertama list, maka elemen berikutnya dapat diakses secara suksesif dari field next elemen tersebut.

Alamat yang sudah didefinisikan disebut sudah di alokasi. Didefinisikan suatu konstanta **Nil,** yang artinya alamat yang tidak terdefinisi. Alamat ini nantinya akan didefinisikan secara lebih konkret ketika list linier diimplementasi pada struktur data fisik.

Sebuah list linier dikenali:

- **Elemen pertamanya**, biasanya melalui alamat elemen pertama yang disebut: **First**
- **Alamat elemen berikutnya (suksesor)**, jika kita mengetahui alamat sebuah elemen, yang dapat diakses melalui informasi **Next**. **Next** mungkin ada secara eksplisit, atau secara implisit yaitu lewat kalkulasi atau fungsi suksesor.
- Setiap elemen mempunyai alamat, yaitu tempat elemen disimpan dapat diacu. Untuk mengacu sebuah elemen, alamat harus terdefinisi. Dengan alamat tersebut Informasi yang tersimpan pada elemen list dapat diakses.
- **Elemen terakhirnya**, disebut dengan nama **Last**. Ada berbagai cara untuk mengenali elemen akhir.

Jika **L** adalah list linier, dan **P** adalah address:
Alamat elemen pertama list L dapat diacu dengan notasi :

First(L)

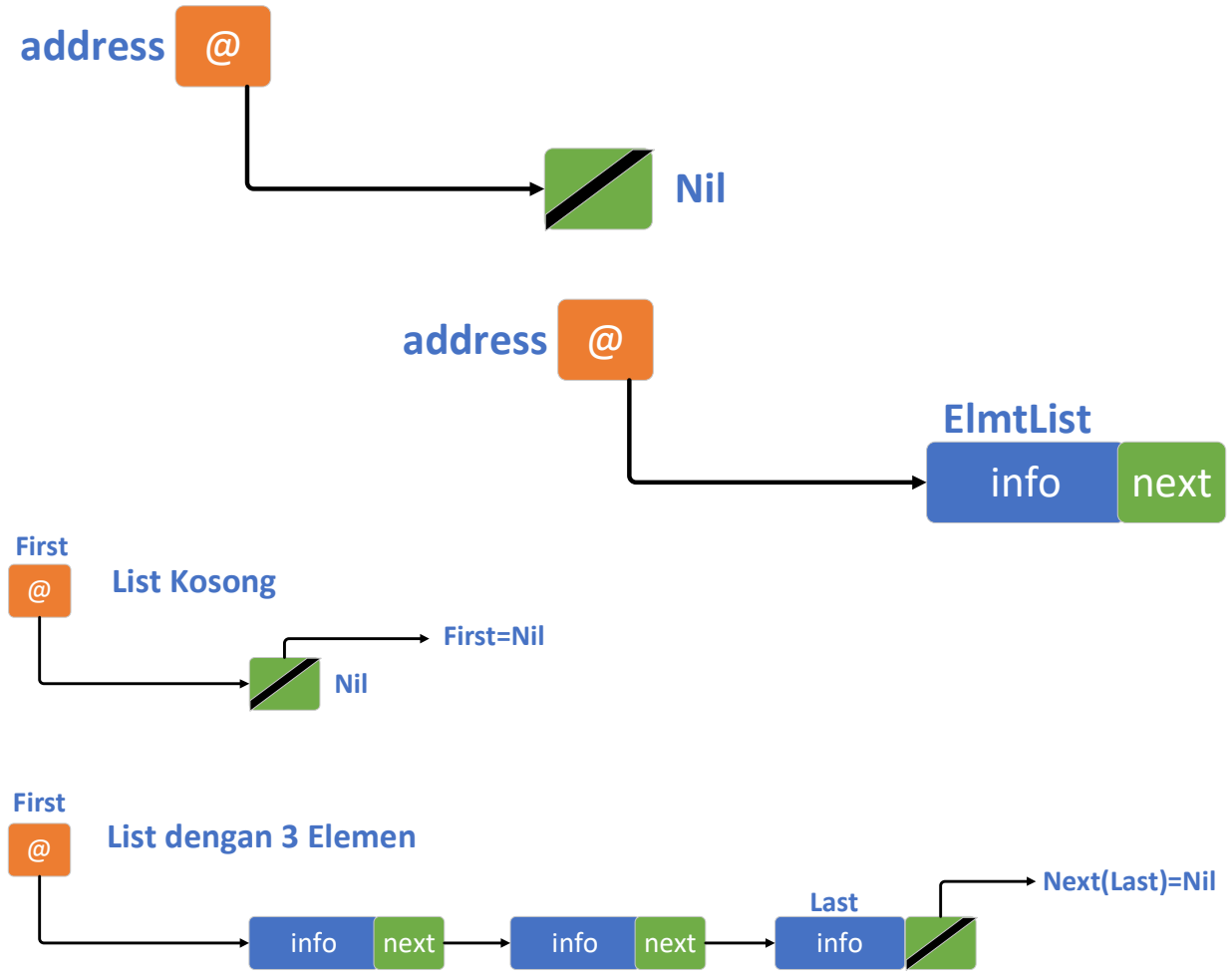
Elemen yang diacu oleh P dapat dikomunikasikan informasinya dengan menggunakan notasi selektor:

info(P)
next(P)

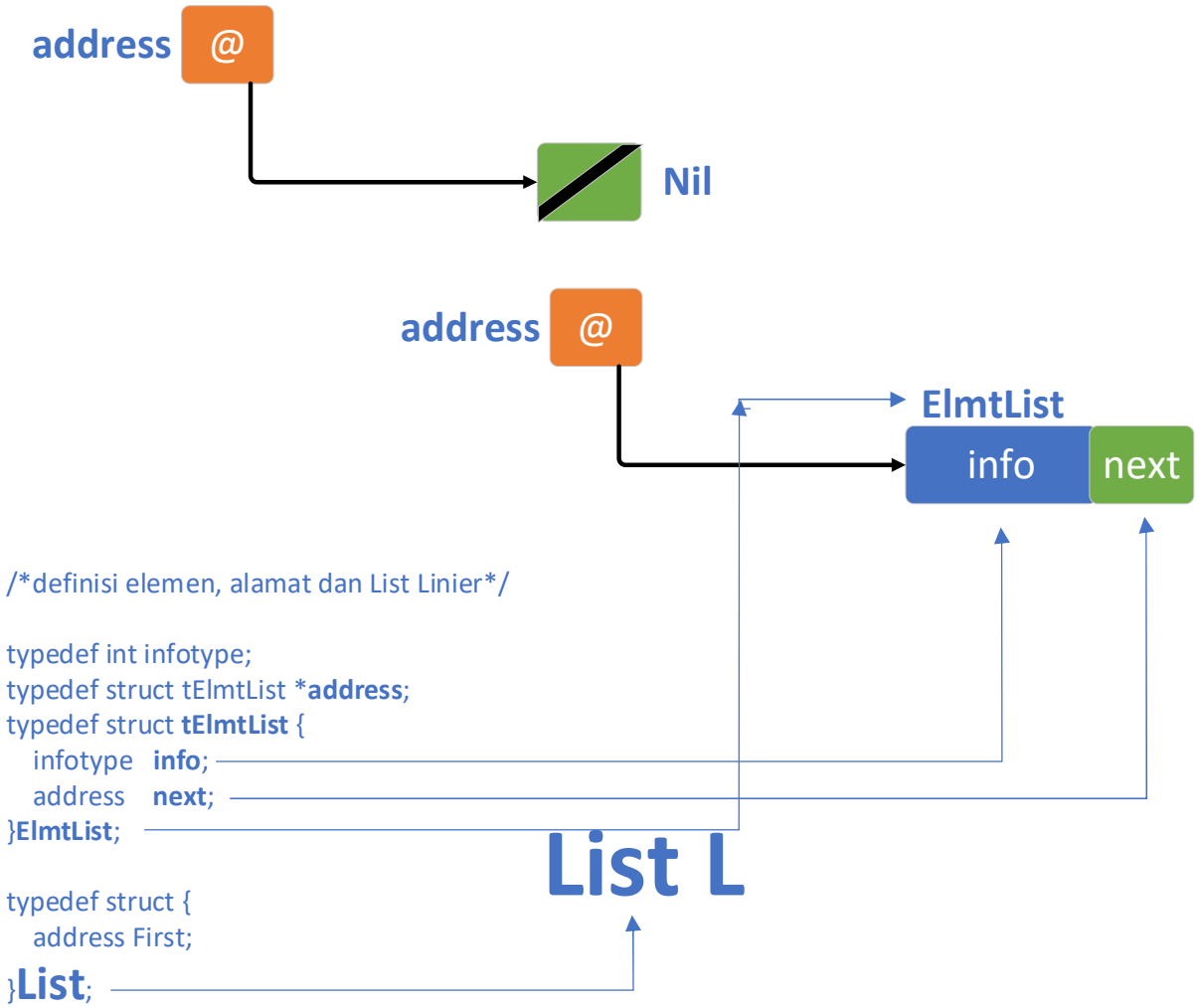
Beberapa definisi :

- List L adalah list kosong, jika **First(L) = Nil**.
- Elemen terakhir dikenali, misalnya jika **Last** adalah alamat element terakhir, maka **next(Last) = Nil**.

Ilustrasi



Representasi Code




```
/*definisi elemen, alamat dan List Linier*/

typedef int infotype;
typedef struct tElmtList *address;
typedef struct tElmtList {
   infotype  info;
   address   next;
}ElmtList;

typedef struct {
   address First;
}List;

/*Definisi Selektor*/
#define Nil          NULL
#define info(P)      (P)->info
#define next(P)      (P)->next
#define First(L)     (L).First
```

Operasi-operasi List Linier

Operasi-operasi yang dapat dilakukan pada list linier terdiri dari sebagai berikut:

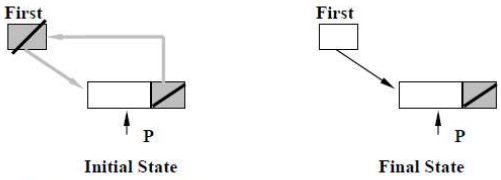
1. Insert First
2. Insert After
3. Insert Last
4. Delete First
5. Delete After
6. Delete Last

Untuk semua operasi list linier dapat dilakukan berdasarkan alamat dan nilai, secara umum proses terhadap list linier dilakukan terhadap alamat kemudian baru dilakukan terhadap nilai.

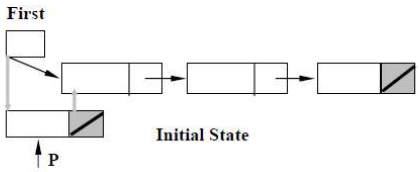
Insert First (diketahui alamatnya)

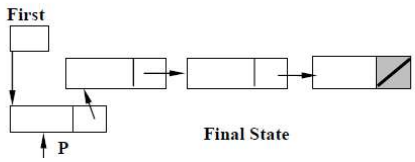
Menambahkan sebuah elemen yang diketahui alamatnya sebagai elemen pertama list L.

Insert elemen pertama, List kosong :



Insert elemen pertama, List tidak kosong :






```
procedure InsertFirst (input/output L : List, input P : address)
{ I.S. List L mungkin kosong, P sudah dialokasi, P ≠ Nil, Next(P)=Nil }
{ F.S. P adalah elemen pertama list L }
{ Insert sebuah elemen beralamat P sebagai elemen pertama list linier L yang
mungkin kosong }
```

```
KAMUS LOKAL
```

```
ALGORITMA
```

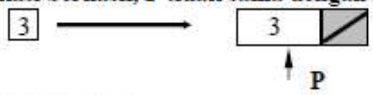
Insert First (diketahui nilainya)

Menambahkan sebuah elemen yang diketahui nilainya sebagai elemen pertama list L.

Tahap pertama :

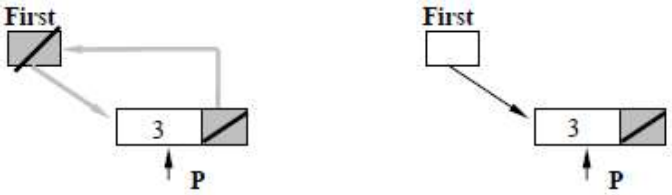
Insert Nilai 3 sebagai elemen pertama, List : karena yang diketahui adalah nilai, maka harus dialokasikan dahulu sebuah elemen supaya nilai 3 dapat di-insert

Jika alokasi berhasil, P tidak sama dengan Nil

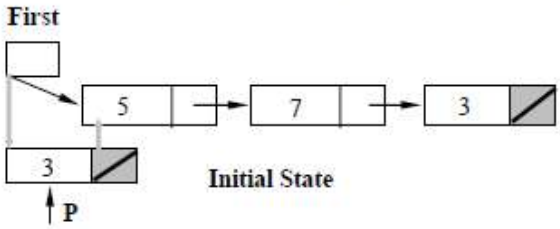


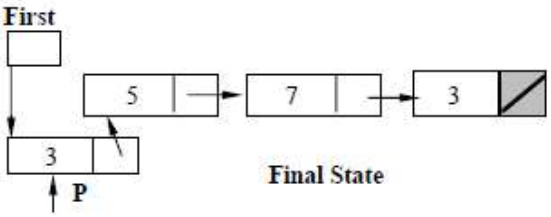
Tahap kedua : insert

Insert ke list kosong



Insert elemen pertama, List tidak kosong :





insVFirst

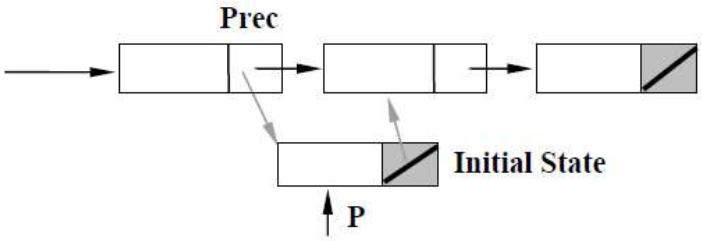
```
procedure InsFirst (input/output L : List, input InfoE : InfoType)
{ I.S. List L mungkin kosong }
{ F.S. Sebuah elemen dialokasi dan menjadi elemen pertama list L, jika alokasi
berhasil. Jika alokasi gagal list tetap seperti semula. }
{ Insert sebuah elemen sbg. elemen pertama list linier L yang mungkin kosong. }
```

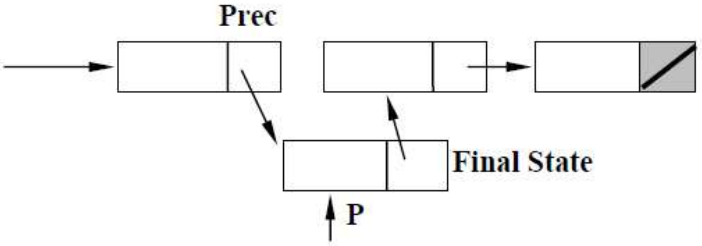
```
KAMUS LOKAL
   function Alokasi (X : infotype) → address
   { Menghasilkan address yang dialokasi. Jika alokasi berhasil,
     Info(P)=InfoE, dan Next(P)=Nil. Jika alokasi gagal, P=Nil }
   P : address { Perhatikan: P adalah variabel lokal!
                 Akan dibahas di kelas }
```

```
ALGORITMA
```

Insert After

Menyisipkan sebuah elemen beralamat **P** setelah alamat **Prec** sebagai suksesor dari sebuah elemen list.






```
procedure InsertAfter (input P, Prec : address)
{ I.S.  Prec adalah elemen list,  Prec ≠ Nil, P sudah dialokasi, P ≠ Nil, Next(P)
= Nil }
{ F.S. P menjadi suksesor Prec }
{ Insert sebuah elemen beralamat P pada List Linier L}
```

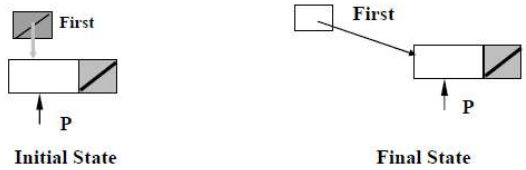
```
KAMUS LOKAL
```

```
ALGORITMA
```

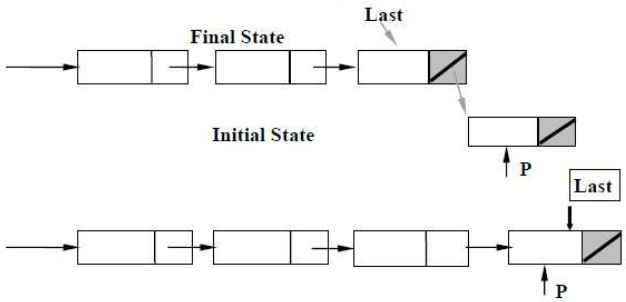
Insert Last

Menyisipkan sebuah elemen beralamat **P** setelah alamat **Last** sebagai elemen terakhir sebuah list linier. Ada dua kemungkinan list kosong atau tidak kosong.

Insert sebagai elemen terakhir list tidak kosong



Insert sebagai elemen terakhir list tidak kosong:



```
procedure InsertLast (input/output L : List, input P : address)
{ I.S. List L mungkin kosong, P sudah dialokasi, P ≠ Nil, Next(P)=Nil }
{ F.S. P adalah elemen trerakhir list L }
{ Insert sebuah elemen beralamat P sbg elemen terakhir dari list linier L yg
mungkin kosong }
```

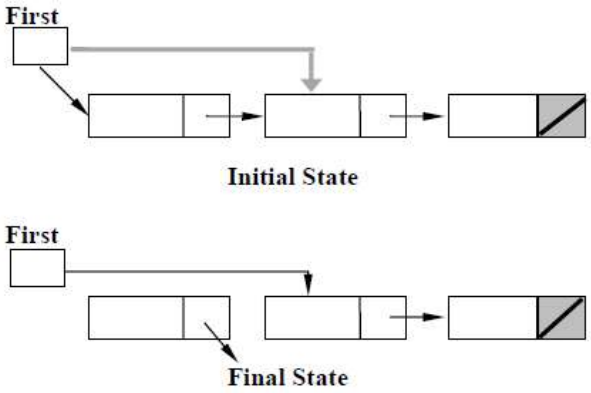
```
KAMUS LOKAL
    Last : address { address untuk traversal,
                     pada akhirnya address elemen terakhir }
```

```
ALGORITMA
```

Delete First

Menghapus elemen pertam list.

a. Elemen yang dihapus dicatat alamatnya :



```
procedure DeleteFirst (input/output L : List, output P : address)
{ I.S. List L tidak kosong, minimal 1 elemen, elemen pertama pasti ada }
{ F.S. First "maju", mungkin bernilai Nil (list menjadi kosong) }
{ Menghapus elemen pertama L, P adalah @ elemen pertama L sebelum penghapusan, L
yang baru adalah Next(L) }
```

```
KAMUS LOKAL
```

```
ALGORITMA
```

b. Elemen yang dihapus dicatat informasinya, dan alamat elemen yang dihapus didealokasi :

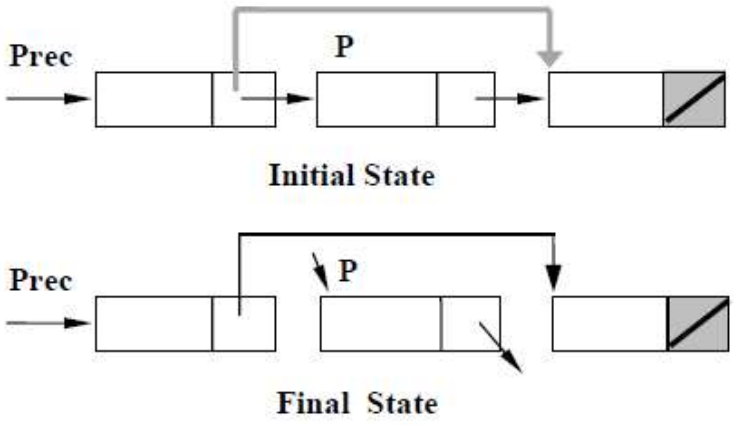
```
procedure DeleteFirst (input/output L : List, output E : InfoType)
{ I.S. List L tidak kosong, minimal 1 elemen, elemen pertama pasti ada }
{ F.S : Menghapus elemen pertama L
  E adalah nilai elemen pertama L sebelum penghapusan, L yang baru
  adalah Next(L) }
```

```
KAMUS LOKAL
   procedure DeAlokasi (input P : address)
   { I.S. P pernah dialokasi. F.S. P=Nil }
   { F.S. Mengembalikan address yang pernah dialokasi. P=Nil}
   P : address
```

```
ALGORITMA
```

Delete After

Menghapus suksesor sebuah elemen list.



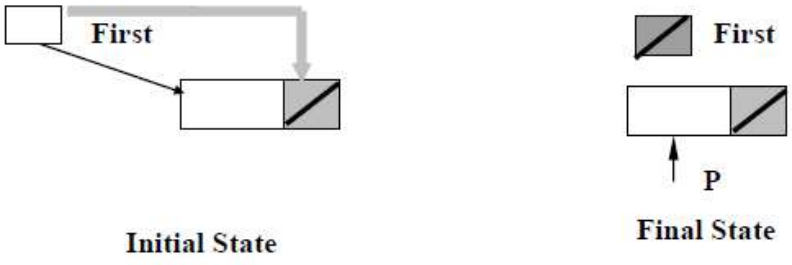
```
procedure DeleteAfter (input Prec : address, output P : address)
{ I.S. List tidak kosong, Prec adalah elemen list, Next(Prec) ≠ Nil }
{ F.S. Next(Prec), yaitu elemen beralamat P dihapus dari List.
       Next(P)=Nil }
{ Menghapus suksesor Prec, P adalah @ suksesor Prec sebelum penghapusan,
  Next(Prec) yang baru adalah suksesor dari suksesor Prec sebelum penghapusan }
KAMUS LOKAL

ALGORITMA
```

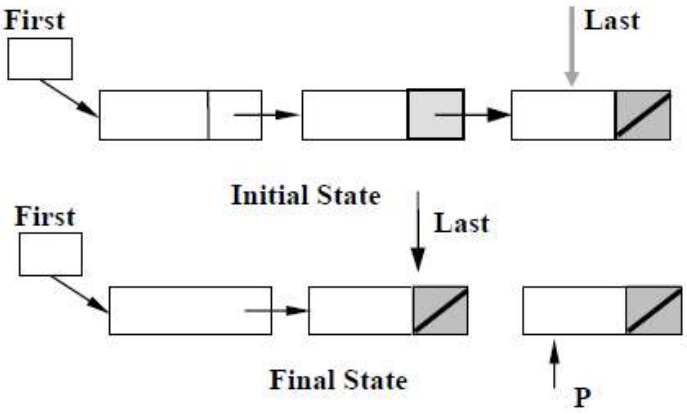
Delete Last

Menghapus elemen terakhir list dapat dilakukan jika alamat dari elemen sebelum elemen terakhir diketahui. Persoalan selanjutnya menjadi persoalan DeleteAfter, kalau Last bukan satu-satunya elemen list linier. Ada dua kasus, yaitu list menjadi kosong atau tidak.

Kasus list menjadi kosong :



List tidak menjadi kosong (masih mengandung elemen) :



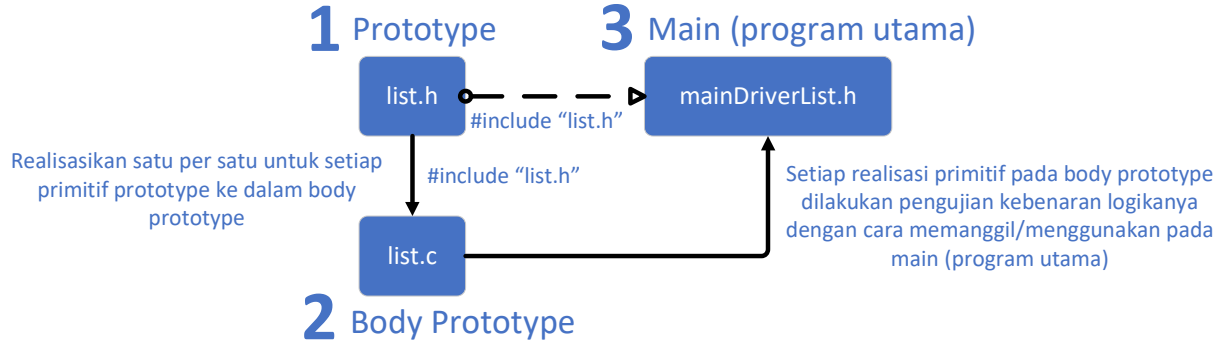
```
procedure DeleteLast (input First : List, output P : address)
{ I.S. List L tidak kosong, minimal mengandung 1 elemen }
{ F.S. P berisi alamat elemen yang dihapus. Next(P)=Nil. List berkurang
elemennya }
{ Menghapus elemen terakhir dari list L, list mungkin menjadi kosong }
```

```
KAMUS LOKAL
    Last, PrecLast : address { address untuk traversal, type terdefinisi
                               pada akhirnya Last adalah alamat elemen
                               terakhir dan PrecLast adalah alamat
                               sebelum yg terakhir }
```

```
ALGORITMA
```

Abstract Data Type (ADT) List Linier

Untuk merealisasikan semua primitif pada list linier digunakan teknik ADT seperti yang diberikan dibawah ini.



Ulangi langkah 2 dan 3 sampai semua primitif selesai dilakukan realisasi.

Prototype List Linier (list.h)

```
#ifndef list_H
#define list_H

#include <stdio.h>
#include <stdlib.h>
#include "boolean.h"

#define Nil          NULL
#define info(P)      (P)->info
#define next(P)      (P)->next
#define First(L)     (L).First

typedef enum{false,true} boolean;
typedef int infotype;
typedef struct tElmtList *address;
typedef struct tElmtList {
    infotype   info;
    address    next;
}ElmtList;

typedef struct {
    address First;
}List;


/* ----- Test List Kosong ----- */
boolean ListEmpty (List L);
/* Mengirim true jika list kosong */

/* ----- Pembuatan List Kosong ----- */
void CreateList (List *L);
/* I.S.           : sembarang
   F.S.           : Terbentuk list kosong */

/* ----- Manajemen Memori ----- */
address Alokasi (infotype X);
/* Mengirimkan address hasil alokasi sebuah elemen */
/* Jika alokasi berhasil, maka address tidak Nil,
   dan misalnya menghasilkan P, maka
   Info(P) = X, Next(P) = Nil
   Jika alokasi gagal, mengirimkan Nil */

void Dealokasi (address *P);
/* I.S.           : P terdefinisi
   F.S.           : P dikembalikan ke sistem
   Melakukan dealokasi/pengembalian address P */

/* ----- Primitif Berdasarkan Alamat ------ */
/* Penambahan Elemen Berdasarkan Alamat */
void InsertFirst (List *L, address P);
/* I.S.     : Sembarang, P sudah dialokasi
   F.S.           : Menambahkan elemen ber-address P sebagai elemen pertama
*/

void InsertAfter (List *L, address P, address Prec);
```

```
/* I.S.           : Prec pastilah elemen list dan bukan elemen terakhir,
                    P sudah dialokasi
   F.S.     : Insert P sebagai elemen sesudah elemen beralamat Prec */

void InsertLast (List *L, address P);
/* I.S.           : Sembarang, P sudah dialokasi
   F.S.           : P ditambahkan sebagai elemen terakhir yang baru */

/* Penghapusan Sebuah Elemen */
void DelFirst (List *L, address *P);
/* I.S.     : List tidak kosong
   F.S.     : P adalah alamat elemen pertama list sebelum penghapusan
              Elemen list berkurang satu (mungkin menjadi kosong)
              First element yang baru adalah suksesor elemen pertama yang
              lama */

void DelLast (List *L, address *P);
/* I.S.     : List tidak kosong
   F.S.     : P adalah alamat elemen terakhir list sebelum penghapusan
              Elemen list berkurang satu (mungkin menjadi kosong)
              Last element baru adalah predesesor elemen pertama yang
              lama, jika ada */

void DelAfter (List *L, address * Pdel, address Prec);
/* I.S.     : List tidak kosong. Prec adalah anggota list L.
   F.S.     : Menghapus Next(Prec) :
              Pdel adalah alamat elemen list yang dihapus

/* ----- Primitif Berdasarkan Nilai ----- */
/* Penambahan Elemen */
void InsVFirst (List *L, infotype X);
/* I.S.           : L mungkin kosong
   F.S.     : X ditambahkan sebagai elemen pertama L
   Proses   : Melakukan alokasi sebuah elemen dan menambahkan elemen
                    pertama dengan nilai X jika alokasi berhasil.
                    Jika alokasi gagal, maka I.S.= F.S. */

void InsVLast (List *L, infotype X);
/* I.S.     : L mungkin kosong
   F.S.           : X ditambahkan sebagai elemen terakhir L
   Proses   : Melakukan alokasi sebuah elemen dan menambahkan elemen list
                    di akhir yaitu
                    jika alokasi berhasil, elemen terakhir yang baru bernilai
X
                    Jika alokasi gagal, maka I.S.= F.S. */

void InsVAfter (List *L, infotype X);
/* I.S.     : L mungkin kosong
   F.S.           : X ditambahkan sebagai elemen setelah Prec
   Proses   : Melakukan alokasi sebuah elemen dan menambahkan elemen list
                    setelah Prec yaitu
                    jika alokasi berhasil, elemen setelah Prec yang baru
bernilai X
                    Jika alokasi gagal, maka I.S.= F.S. */

void AdrLast(List *L, address *Prec, address *Last);
/*mencari alamat elemen terakhir dan sebelumnya*/
```

```
/* Penghapusan ELemen */
void DelVFirst (List *L, infotype *X);
/* I.S.           : List L tidak kosong
   F.S.           : Elemen pertama list dihapus den nilai info disimpan pada
X
                    dan alamat elemen pertama di-dealokasi */


void DelVLast (List *L, infotype *X);
/* I.S.           : List L tidak kosong
   F.S.           : Elemen terakhir list dihapus : nilai info disimpan pada
X
                     dan alamat elemen terakhir di-dealokasi */

void DelVAfter (List *L, infotype *X);
/* I.S.           : List L tidak kosong
   F.S.           : Elemen list setelah Prec dihapus : nilai info disimpan
pada X
                     dan alamat elemen setelah Prec di-dealokasi */

/* Proses Semua Elemen List */
void PrintInfo (List L);
/* I.S.     : List mungkin kosong
   F.S.     : Jika list tidak kosong, semua info yg disimpan pada elemen
              list diprint
              Jika list kosong, hanya menuliskan "list kosong" */

int NbElmt (List L);
/* Mengirimkan banyaknya elemen list; mengirimkan 0 jika list kosong */

#endif
```

Untuk file Boolean.h diberikan pada kode sumber di bawah ini. Alasan diperlukan file Boolean.h adalah untuk mendefinisikan tipe data Boolean yang secara default C ANSI tidak disediakan, sehingga pemrogram harus membuat sendiri. Pada file boolean.h hanya mendefiniskan bahwa true=1 dan false=0.

File boolean.h

```
/*Nama File : boolean.h*/
/*Deskripsi : Header fungsi boolean */
/*Dibuat    : Aris Puji Widodo*/
/*Tanggal   : Thu 20 Sep 2001 09:33:24 AM JAVT*/

/* Kamus */
#ifndef boolean_H
#define boolean_H
#define true 1
#define false 0
#define boolean unsigned char
#endif
```

Body Prototype (list.c)

Contoh realisasi primitif prototype pada bagian body prototype, kemudian lanjutkan untuk primitif-primitif yang lainya.

```
#ifndef list_C
#define list_C
. . .
#include "list.h"
. . .
address Alokasi (infotype X){
/* Mengirimkan address hasil alokasi sebuah elemen */
/* Jika alokasi berhasil, maka address tidak Nil,
   dan misalnya menghasilkan P, maka
   Info(P) = X, Next(P) = Nil
   Jika alokasi gagal, mengirimkan Nil */
/*Kamus Lokal*/
    address P;

/*Algoritma*/
    P=(address)malloc(sizeof(ElmtList));
    if(P!=Nil){
        info(P)=X;
        next(P)=Nil;
    }
    return P;
}

void Dealokasi (address *P){
/* I.S.           : P terdefinisi
   F.S.           : P dikembalikan ke sistem
   Melakukan dealokasi/pengembalian address P */
/*Kamus Lokal*/

/*Algoritma*/
   free(*P);
}

void PrintInfo (List L){
/* I.S.     : List mungkin kosong
   F.S.     : Jika list tidak kosong, semua info yg disimpan pada elemen
              list diprint
              Jika list kosong, hanya menuliskan "list kosong" */
/*Kamus Lokal*/
    address P;

/*Algoritma*/
    if(ListEmpty(L)){
        printf("\nList Kosong\n");
    }
    else{
        P=First(L);
        do{
             printf("%d  ",info(P));
             P=next(P);
        }while(P!=Nil); /* while(P!=First(L)) untuk circular*/
```

```
        printf("\n");
    }
}
. . . /*dan seterusnya*/
#endif
```