Variasi List Linier (Bagian 2)

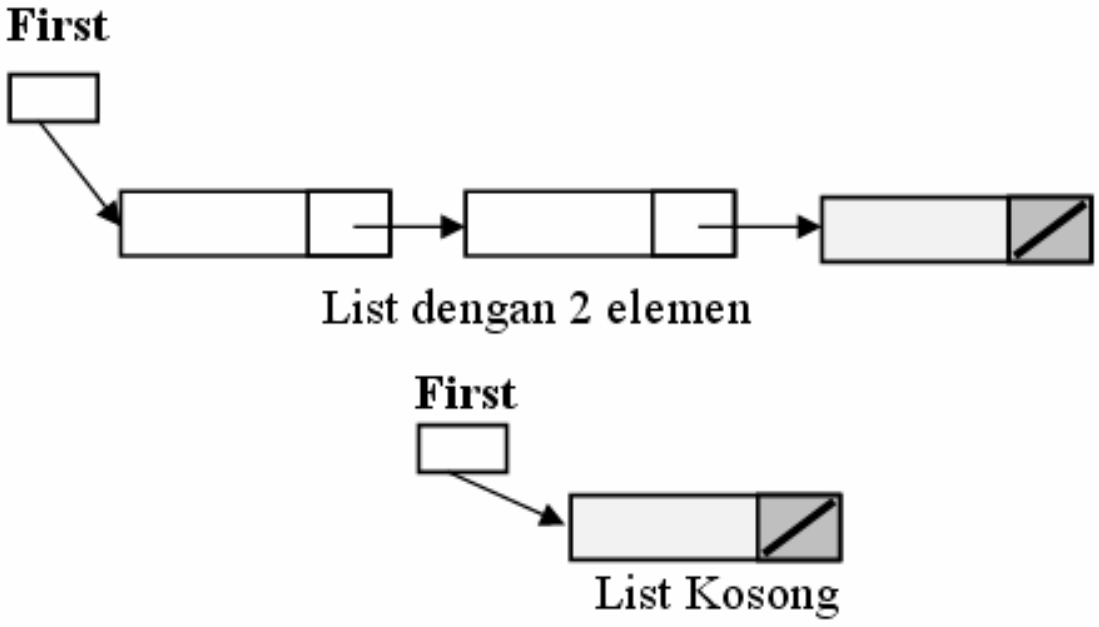
Tim Pengajar IF2030

Semester I/2009-2010

List dengan elemen fiktif (*dummy element*) di akhir

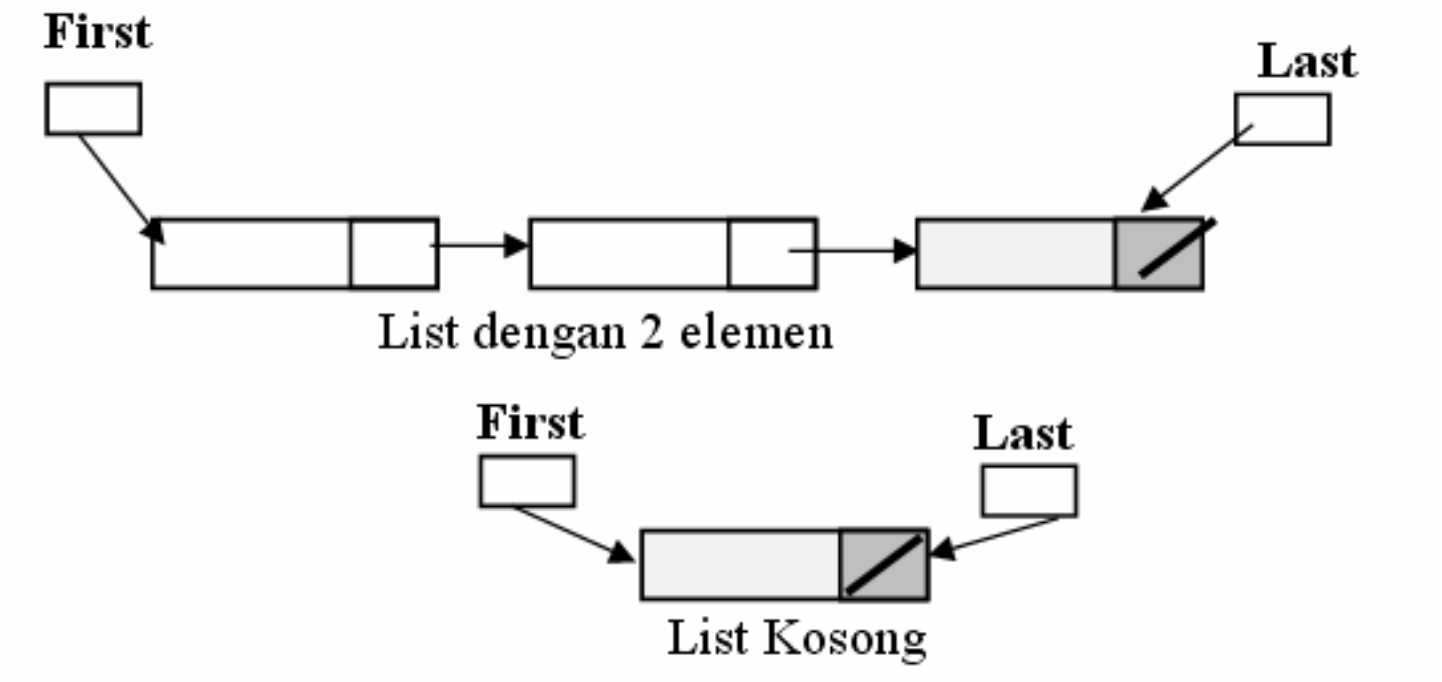
List dengan Elemen Fiktif pada Elemen Terakhir

Elemen pertama : First (L)
Elemen terakhir : dummy@
List kosong : First(L) = dummy@



List dengan Elemen Fiktif di Akhir dan Pencatatan Alamat Elemen Akhir

Elemen pertama : First(L)
Elemen terakhir : Last(L); = dummy@
List kosong : First(L) = Last(L) = dummy@



Beberapa catatan

- Sering dipakai jika *dummy* adalah sentinel, dan pencarian diperlukan sebelum penambahan elemen
  - Nilai yang dicari dapat secara langsung disimpan untuk sementara pada *dummy*, kemudian dilakukan search
  - Jika search tidak berhasil, dan elemen akan ditambahkan, maka dialokasi sebuah *dummy* yang baru, nilai Last berubah
  - Contoh pemakaian dijelaskan pada topological sort (Lihat Studi Kasus 6)
- *Dummy* bisa berupa address yang tetap, bisa sebuah address yang berbeda (setiap kali *dummy* tersebut dipakai sebagai elemen list, dialokasi *dummy* yang baru)

Bahasan - 1

- Buatlah ADT list + driver dengan elemen dummy di akhir:
  - Penunjuk first dan last
  - Alamat dummy: tetap
  - Representasi fisik: berkait dengan pointer

Representasi Fisik dengan Pointer


```
#define Nil NULL

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

/* Definisi list : */
/* List kosong : First(L) = Last(L) = dummy@ */
/* Setiap elemen dengan address P dapat diacu Info(P), Next(P) */
/* Elemen dummy terletak pada last */
typedef struct {
        address First;
        address Last;
} List;

/* Selektor */
#define Info(P) (P)->info
#define Next(P) (P)->next
#define First(L) ((L).First)
#define Last(L) ((L).Last)
```

Beberapa primitif

- Buatlah sebagai latihan:

```
/* PROTOTYPE */
/****************** TEST LIST KOSONG ******************/
boolean ListEmpty (List L);
/* Mengirim true jika list kosong: First(L) = dummy@ dan Last(L)
= dummy@ */
/****************** PEMBUATAN LIST KOSONG ******************/
void CreateList (List * L);
/* I.S. sembarang             */
/* F.S. Terbentuk list L kosong, dengan satu elemen dummy */
/* Jika gagal maka First = Last = Nil dan list gagal terbentuk */
/****************** SEARCHING ******************/
address Search (List L, infotype X)
/* Mencari apakah ada elemen list dengan info(P)= X */
/* Jika ada, mengirimkan address elemen tersebut. */
/* Jika tidak ada, mengirimkan Nil */
```

Beberapa primitif

- Buatlah sebagai latihan:

```
void InsertFirst (List * L, address P);
/* I.S. Sembarang, P sudah dialokasi  */
/* F.S. Menambahkan elemen ber-address P sebagai elemen pertama */
void InsertLast (List * L, address P);
/* I.S. Sembarang, P sudah dialokasi  */
/* F.S. P ditambahkan sebagai elemen terakhir yang baru, */
/*      yaitu menjadi elemen sebelum dummy */
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 yg 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 sebelum dummy pada list
          sebelum penghapusan  */
/*       Elemen list berkurang satu (mungkin menjadi kosong) */
```

Beberapa primitif

```
boolean ListEmpty (List L)
/* Mengirim true jika list kosong */
{       /* Kamus Lokal */

        /* Algoritma */
        return ((First(L) == Last(L)) && (Last(L) != Nil));
}
```

Beberapa primitif


```
void CreateList (List * L)
/* I.S. sembarang              */
/* F.S. Terbentuk list L kosong, dengan satu elemen dummy,
jika alokasi dummy berhasil */
/* Jika gagal maka First = Last = Nil dan list gagal terbentuk */
{       /* Kamus Lokal */
        address Pdummy;

        /* Algoritma */
        Pdummy = (address) malloc (sizeof(ElmtList));
        if (Pdummy != Nil) {
                Next(Pdummy) = Nil; /* Info(P) tidak didefinisikan */
                First(*L) = Pdummy;
                Last(*L) = Pdummy;
        } else /* List gagal terbentuk */ {
                First(*L) = Nil;
                Last(*L) = Nil;
        }
}
```

Beberapa primitif


```
address Search (List L, infotype X)
/* Mencari apakah ada elemen list dengan info(P)= X */
/* Jika ada, mengirimkan address elemen tersebut. */
/* Jika tidak ada, mengirimkan Nil */
{       /* Kamus Lokal */
        address Pt;

        /* Algoritma */
        if (ListEmpty(L)) {
                return Nil;
        } else /* list tidak kosong */ {
                Info(Last(L)) = X; /* letakkan sentinel */
                Pt = First(L);
                while (Info(Pt) != X) {
                        Pt = Next(Pt);
                } /* Info(Pt) = X */
                if (Pt != Last(L)) { /* tidak ketemu di sentinel */
                        return Pt;
                } else /* Pt = Last(L), ketemu di sentinel */ {
                        return Nil;
                }
        }
}
```

Beberapa primitif


```
void InsertFirst (List * L, address P)
/* I.S. Sembarang, P sudah dialokasi  */
/* F.S. Menambahkan elemen ber-address P sebagai elemen pertama */
{       /* Kamus Lokal */

        /* Algoritma */
        Next(P) = First(*L);
        First(*L) = P;
}
```

Beberapa primitif


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

        /* Algoritma */
        if (ListEmpty(*L)) {
                InsertFirst(L,P);
        } else {
                last = First(*L);
                while (Next(last) != Last(*L)) {
                        last = Next(last);
                } /* Next(last) == Last(*L) alias dummy */
                InsertAfter(L,P,last);
        }
}
```

Beberapa primitif


```
void DelFirst (List * L, address * P)
/* I.S. List L tidak kosong */
/* F.S. P adalah alamat elemen pertama list sebelum penghapusan */
/*      Elemen list berkurang satu (mungkin menjadi kosong) */
/* First element yg baru adalah suksesor elemen pertama yang lama */
{       /* Kamus Lokal */

        /* Algoritma */
        *P = First(*L);
        First(*L) = Next(First(*L));
        Next(*P) = Nil;
}
```

Beberapa primitif

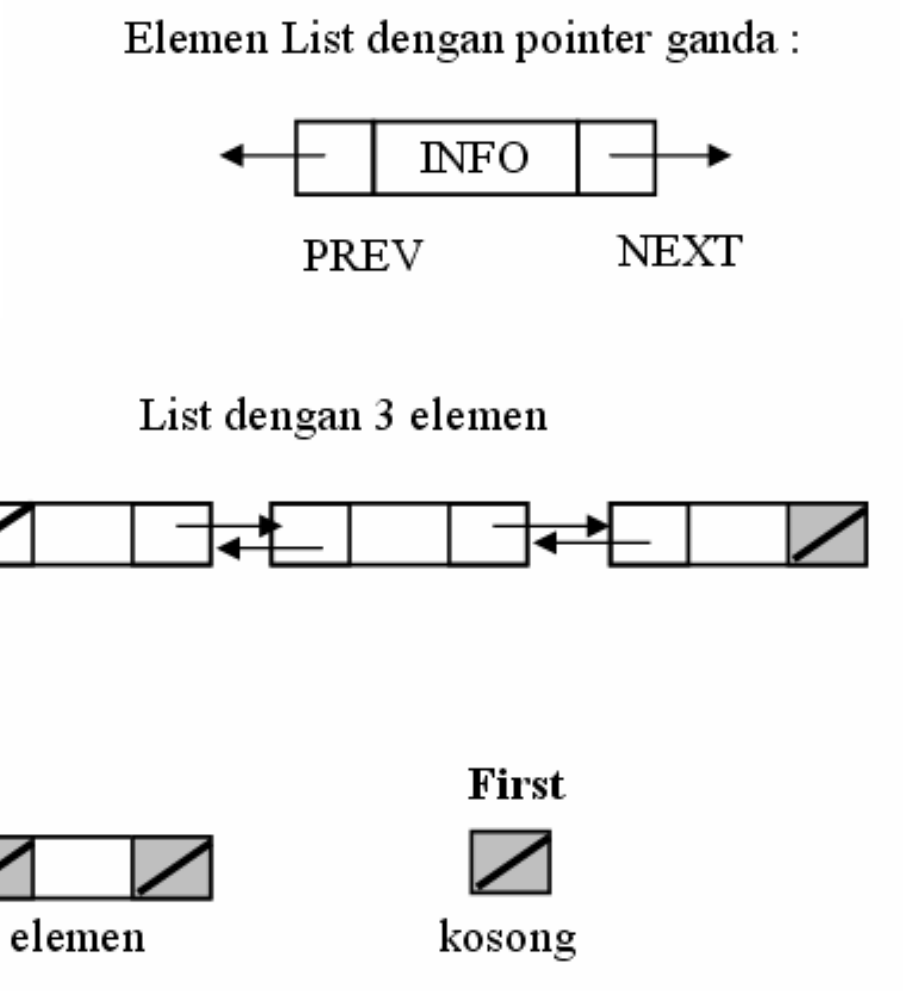

```
void DelLast (List * L, address * P)
/* I.S. List tidak kosong */
/* F.S. P adalah alamat elemen terakhir list sebelum penghapusan  */
/*      Elemen list berkurang satu (mungkin menjadi kosong) */
/* Last element baru adalah predesesor elemen pertama yg lama, */
/* jika ada */
{       /* Kamus Lokal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

        /* Algoritma */
        last = First(*L);
        preclast = Nil;
        while (Next(last) != Last(*L)) {
                preclast = last;
                last = Next(last);
        }
        *P = last;
        if (preclast == Nil) { /* kasus satu elemen */
                First(*L) = Last(*L);
        } else {
                Next(preclast) = Last(*L);
        }
}
```

List dengan Double Pointer

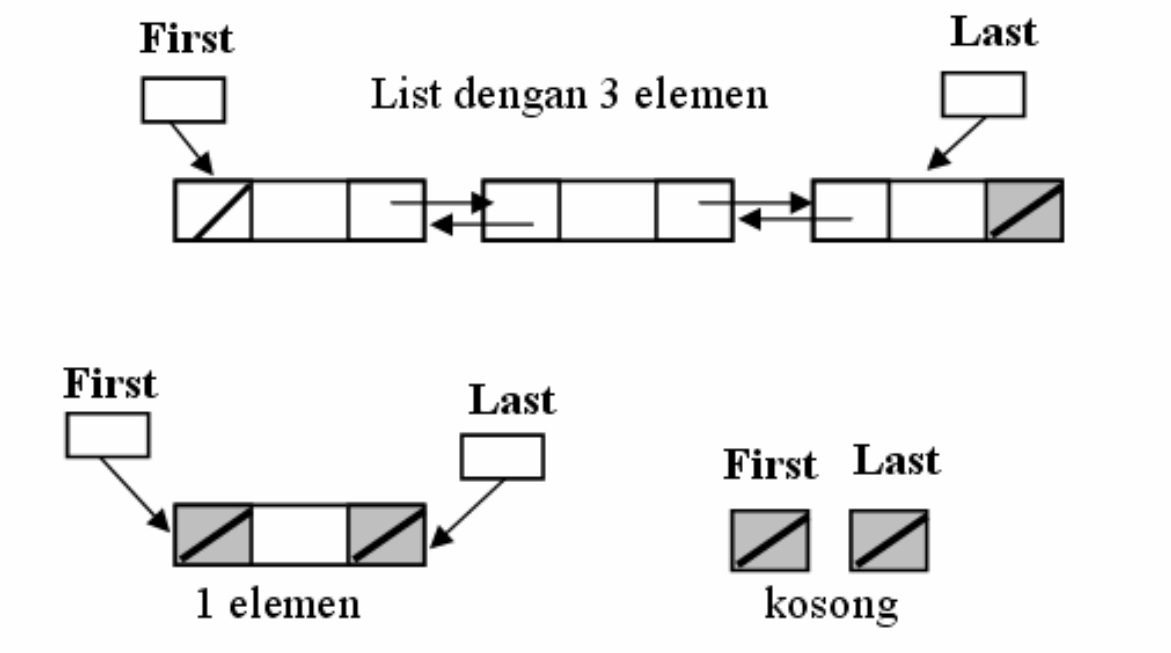
List dengan Pointer Ganda

Elemen pertama : First(L)
Elemen terakhir : Last(L) = P;
Next(P) = Nil
List kosong : First(L) = Nil



List dengan Pointer Ganda, dicatat Last

Elemen pertama : First(L)
Elemen terakhir : Last(L); Next(Last(L)) = Nil
List kosong : First(L) = Last (L) = Nil



Beberapa catatan

- Dibutuhkan jika harus dilakukan banyak operasi terhadap elemen suksesor dan juga predesesor.
  - Dengan tersedianya alamat predesesor pada setiap elemen list, maka memorisasi Prec pada beberapa algoritma yang pernah ditulis dapat dihindari
- Operasi dasar menjadi sangat “banyak”
- Memori yang dibutuhkan membesar
- Jika list lojik semacam ini direpresentasi secara kontigu dengan tabel, maka sangat menguntungkan karena memorisasi Prev dan Next dilakukan dengan kalkulasi

Bahasan - 2

- Buatlah ADT list dengan elemen berpointer ganda + driver :
  - Representasi fisik: berkait dengan pointer
  - Penunjuk First dan Last

Rep. Fisik dengan Pointer


```
#define Nil NULL
typedef int infotype;
typedef struct tElmtList *address;
typedef struct tElmtList {
        infotype info;
        address prev;
        address next;
} ElmtList;
/* Definisi list : */
/* List kosong : First = Nil and Last = Nil */
/* Setiap elemen dengan address P dapat diacu Info(P), Prev(P),
Next(P) */
typedef struct {
        address First;
        address Last;
} List;
/* Selektor */
#define Info(P) (P)->info
#define Prev(P) (P)->prev
#define Next(P) (P)->next
#define First(L) ((L).First)
#define Last(L) ((L).Last)
```

Beberapa Primitif

- Buatlah sebagai latihan:

```
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, */
```

Beberapa Primitif

- Buatlah sebagai latihan:

```
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 yg 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 sebelum dummy pada list
        sebelum penghapusan  */
/*      Elemen list berkurang satu (mungkin menjadi kosong) */
void DelAfter (List * L, address * Pdel, address Prec);
/* I.S. List tidak kosong. Prec adalah anggota list  */
/* F.S. Menghapus Next(Prec): */
/*      Pdel adalah alamat elemen list yang dihapus  */
```

Beberapa Primitif


```
void InsertFirst (List * L, address P)
/* I.S. Sembarang, P sudah dialokasi  */
/* F.S. Menambahkan elemen ber-address P sebagai elemen pertama */
{       /* Kamus Lokal */

        /* Algoritma */
        Next(P) = First(*L);
        if (!ListEmpty(*L)) {
                Prev(First(*L)) = P;
        } else /* L kosong */ {
                Last(*L) = P;
        }
        First(*L) = P;
}
```

Beberapa Primitif


```
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 */
{       /* Kamus Lokal */

        /* Algoritma */
        Prev(Next(Prec)) = P;
        Next(P) = Next(Prec);
        Prev(P) = Prec;
        Next(Prec) = P;
}
```

Beberapa Primitif


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

        /* Algoritma */
        Prev(P) = Last(*L);
        if (!ListEmpty(*L)) {
                Next(Last(*L)) = P;
        } else /* L kosong */ {
                First(*L) = P;
        }
        Last(*L) = P;
}
```

Beberapa Primitif

```
void DelFirst (List * L, address * P)
/* I.S. List L tidak kosong */
/* F.S. P adalah alamat elemen pertama list sebelum penghapusan */
/*      Elemen list berkurang satu (mungkin menjadi kosong) */
/* First element yg baru adalah suksesor elemen pertama yang lama */
{       /* Kamus Lokal */

        /* Algoritma */
        *P = First(*L);
        if (First(*L) == Last(*L)) {
                Last(*L) = Nil;
        }
        First(*L) = Next(First(*L));
        if (First(*L) != Nil) {
                Prev(First(*L)) = Nil;
        }
        Next(*P) = Nil;
}
```

Beberapa Primitif


```
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 yg lama, */
/* jika ada */
{       /* Kamus Lokal */

        /* Algoritma */
        *P = Last(*L);
        if (First(*L) == Last(*L)) {
                First(*L) = Nil;
        }
        Last(*L) = Prev(Last(*L));
        if (Last(*L) != Nil) {
                Next(Last(*L)) = Nil;
        }
        Prev(*P) = Nil;
}
```

Beberapa Primitif

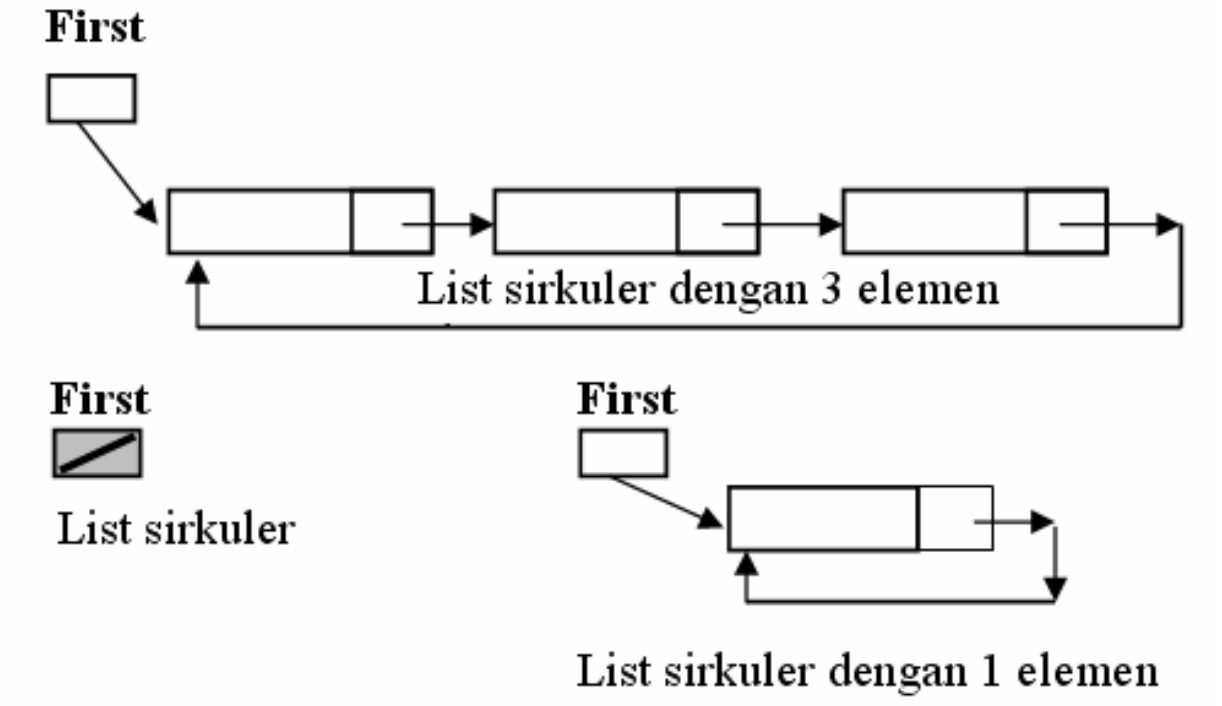

```
void DelAfter (List * L, address * Pdel, address Prec)
/* I.S. List tidak kosong. Prec adalah anggota list  */
/* F.S. Menghapus Next(Prec): */
/*      Pdel adalah alamat elemen list yang dihapus  */
{       /* Kamus Lokal */

        /* Algoritma */
        *Pdel = Next(Prec);
        Next(Prec) = Next(Next(Prec));
        Prev(Next(Prec)) = Prec;
        Next(*Pdel) = Nil;
        Prev(*Pdel) = Nil;
}
```

List Sirkuler

List Sirkuler

Elemen pertama : First(L)
Elemen terakhir : Last(L) = P; Next(P) = First(L)
List kosong : First(L) = Nil



Beberapa catatan

- List dengan representasi ini sebenarnya tidak mempunyai “First”
  - First adalah *“Current Pointer”*
- Representasi ini dipakai jika dilakukan proses terus menerus terhadap anggota list (misalnya dalam *round robin services* pada sistem operasi)
- Penambahan dan penghapusan pada elemen pertama akan berakibat harus melakukan traversal untuk mengubah Next dari elemen Last.

Bahasan - 3

- Buatlah ADT list sirkuler + driver:
  - Penunjuk First
  - Representasi fisik: berkait dengan pointer

Rep. Fisik dengan Pointer


```
#define Nil NULL

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

/* Definisi list : */
/* List kosong : First(L) = Nil */
/* Setiap elemen dengan address P dapat diacu Info(P), Next(P) */
/* Elemen terakhir list : jika addressnya Last,
maka Next(Last) = First */
typedef struct {
        address First;
} List;

/* Selektor */
#define Info(P) (P)->info
#define Next(P) (P)->next
#define First(L) ((L).First)
```

Beberapa Primitif

- Buatlah sebagai latihan:

```
boolean FSearch (List L, address P);
/* Mencari apakah ada elemen list yang beralamat P */
/* Mengirimkan true jika ada, false jika tidak ada */
void InsertFirst (List * L, address P);
/* I.S. Sembarang, P sudah dialokasi  */
/* F.S. Menambahkan elemen ber-address P sebagai elemen pertama */
void InsertLast (List * L, address P);
/* I.S. Sembarang, P sudah dialokasi  */
/* F.S. P ditambahkan sebagai elemen terakhir yang baru */
```

Beberapa Primitif

- Buatlah sebagai latihan:

```
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 yg 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 yg lama, */
/* jika ada */
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" */
```

Beberapa Primitif


```
boolean FSearch (List L, address P)
/* Mencari apakah ada elemen list yang beralamat P */
/* Mengirimkan true jika ada, false jika tidak ada */
{       /* Kamus Lokal */
        address Pt;

        /* Algoritma */
        if (ListEmpty(L)) {
                return false;
        } else {
                Pt = First(L);
                while ((Next(Pt) != First(L)) && (Pt != P)) {
                        Pt = Next(Pt);
                }  /* Next(Pt) = First(L) or Pt = P */
                if (Pt == P) {
                        return true;
                } else {
                        return false;
                }
        }
}
```

Beberapa Primitif


```
void InsertFirst (List * L, address P)
/* I.S. Sembarang, P sudah dialokasi  */
/* F.S. Menambahkan elemen ber-address P sebagai elemen pertama */
{       /* Kamus Lokal */
        address last;

        /* Algoritma */
        if (ListEmpty(*L)) {
                Next(P) = P;
        } else /* L tidak kosong */ {
                last = First(*L);
                while (Next(last) != First(*L)) {
                        last = Next(last);
                } /* Next(last) = First(L) ==> elemen terakhir */
                Next(P) = First(*L);
                Next(last) = P;
        }
        First(*L) = P;
}
```

Beberapa Primitif


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

        /* Algoritma */
        if (ListEmpty(*L)) {
                InsertFirst(L,P);
        } else {
                last = First(*L);
                while (Next(last) != First(*L)) {
                        last = Next(last);
                } /* Next(last) = First(L) */
                InsertAfter(L,P,last);
        }
}
```

Beberapa Primitif


```
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 yg baru adalah suksesor elemen pertama yang lama */
{       /* Kamus Lokal */
        address last;

        /* Algoritma */
        *P = First(*L);
        if (Next(First(*L)) == First(*L)) { /* satu elemen */
                First(*L) = Nil;
        } else {
                last = First(*L);
                while (Next(last) != First(*L)) {
                        last = Next(last);
                } /* Next(last) = First(L) */
                First(*L) = Next(First(*L));
                Next(last) = First(*L);
        }
        Next(*P) = Nil;
}
```

Beberapa Primitif

```
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 yg lama, */
/* jika ada */
{       /* Kamus Lokal */
        address Last, PrecLast;

        /* Algoritma */
        Last = First(*L);
        PrecLast = Nil;
        while (Next(Last) != First(*L)) {
                PrecLast = Last;
                Last = Next(Last);
        } /* Next(Last) = First(*L) */
        *P = Last;
        if (PrecLast == Nil) { /* kasus satu elemen */
                First(*L) = Nil;
        } else {
                Next(PrecLast) = First(*L);
        }
        Next(*P) = Nil;
}
```

Beberapa Primitif


```
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("List Kosong \n");
        } else {
                P = First(L);
                printf("List : \n");
                do {
                        printf("%d \n", Info(P));
                        P = Next(P);
                } while (P != First(L));
        }
}
```

PR

- Untuk praktikum 11 (23-24 Nov 2009):
  - Modul pra-praktikum: P11. Variasi List Linier
    - Bagian 1. ADT List First-Last dengan Dummy pada Last
    - Bagian 2. ADT List Sirkuler
    - Bagian 3. ADT List dengan Double Pointer