

# Variasi Struktur List Linier

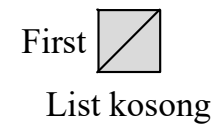
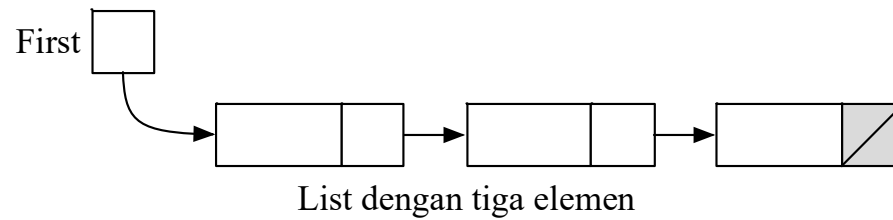
IF2110/IF2111 – Algoritma dan Struktur Data  
Sekolah Teknik Elektro dan Informatika  
Institut Teknologi Bandung

# List Linier

Elemen pertama ???

Elemen terakhir ???

List kosong ???

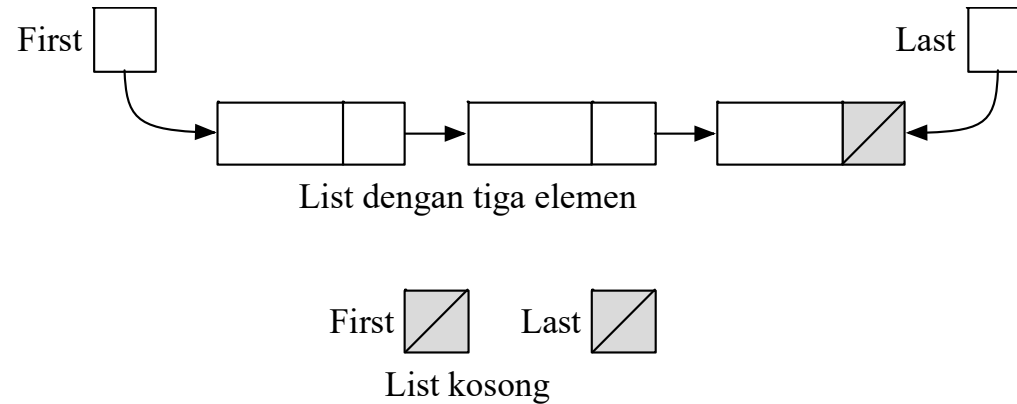


# List Linier yang dicatat alamat elemen pertama dan terakhir

Elemen pertama ????

Elemen terakhir ???

List kosong ???

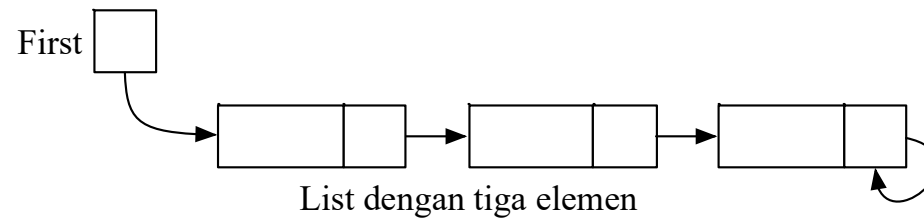


# List yang Elemen Terakhir Menunjuk pada Diri Sendiri

Elemen pertama ????

Elemen terakhir ???

List kosong ???

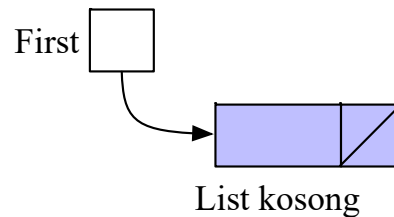
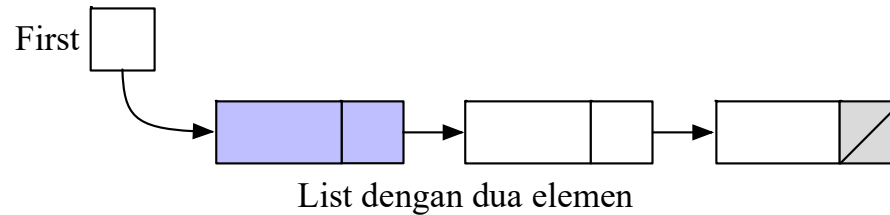


# List dengan “*Dummy Element*” pada Elemen Pertama

Elemen pertama ????

Elemen terakhir ???

List kosong ???

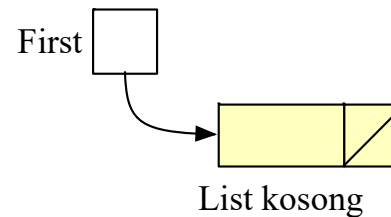
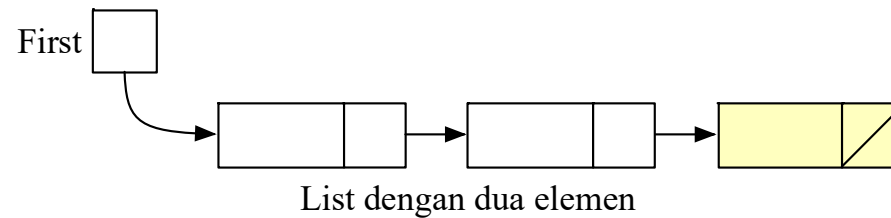


# List dengan “*Dummy Element*” pada Elemen Terakhir

Elemen pertama ????

Elemen terakhir ???

List kosong ???

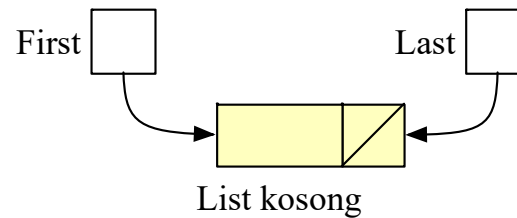
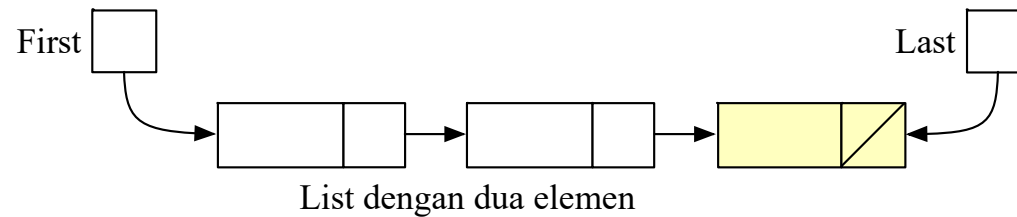


# List dengan *Dummy Element* di Akhir dan Pencatatan Alamat Elemen Akhir

Elemen pertama ????

Elemen terakhir ???

List kosong ???

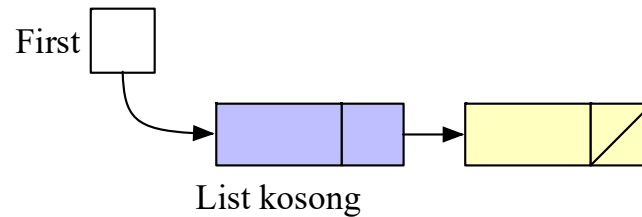
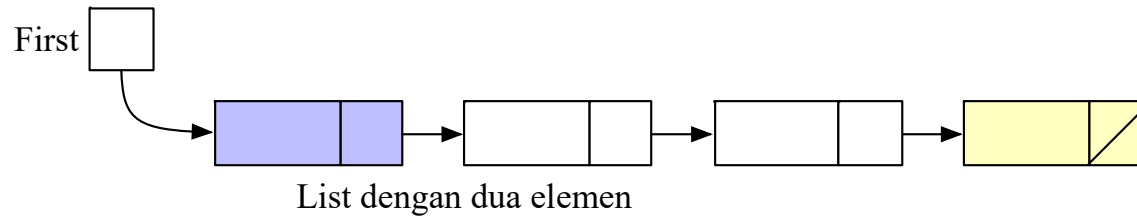


# List dengan *Dummy Element* pada Elemen Pertama dan Terakhir

Elemen pertama ????

Elemen terakhir ???

List kosong ???



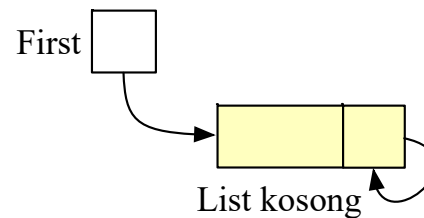
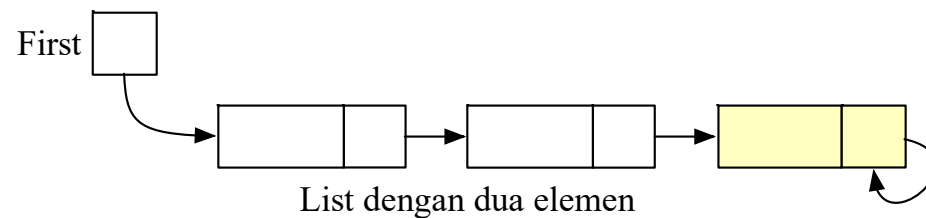


# List dengan *Dummy Element* pada Elemen Terakhir yang Menunjuk ke Diri Sendiri

Elemen pertama ????

Elemen terakhir ???

List kosong ???

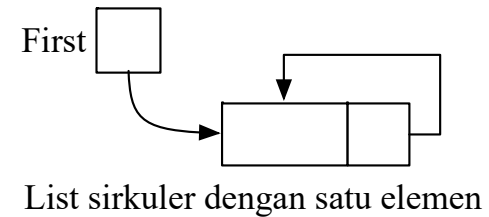
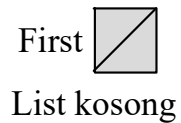
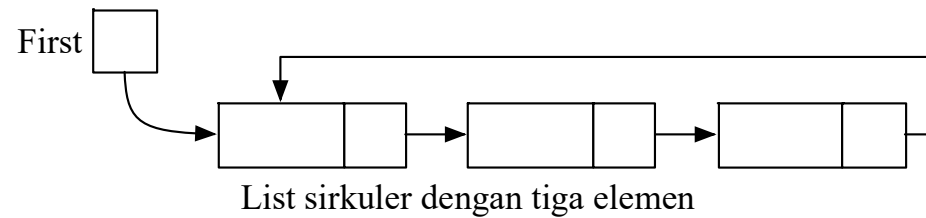


# List Sirkuler

Elemen pertama ????

Elemen terakhir ???

List kosong ???

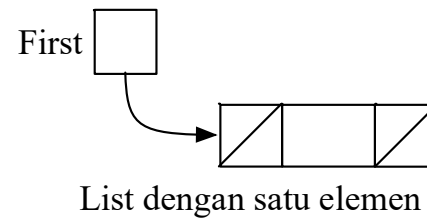
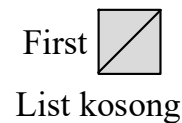
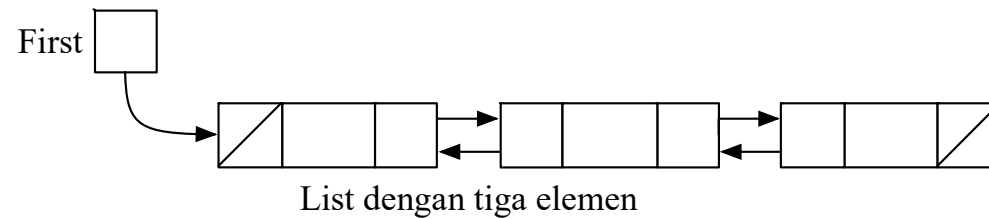
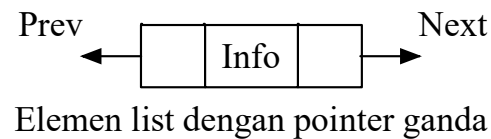


# List dengan Pointer Ganda (*doubly linked list*)

Elemen pertama ????

Elemen terakhir ???

List kosong ???

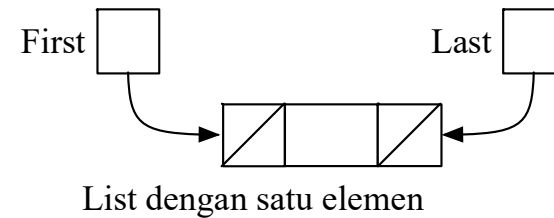
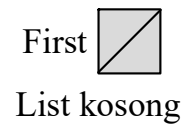
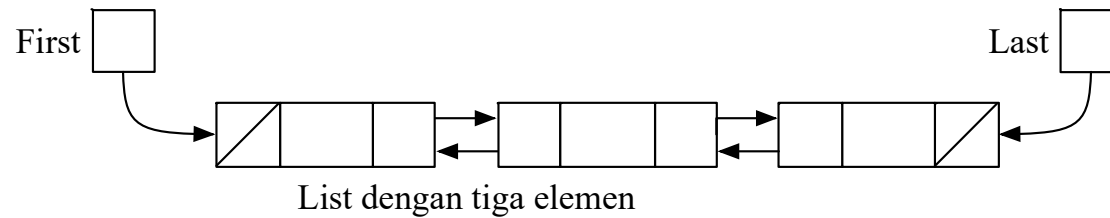


# List dengan Pointer Ganda dan Pencatatan Alamat Elemen Terakhir

Elemen pertama ????

Elemen terakhir ???

List kosong ???

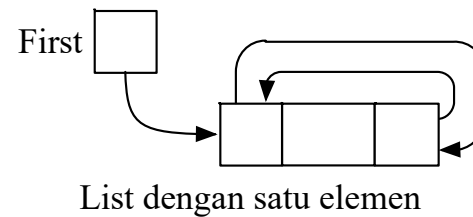
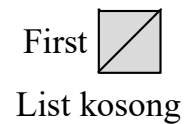
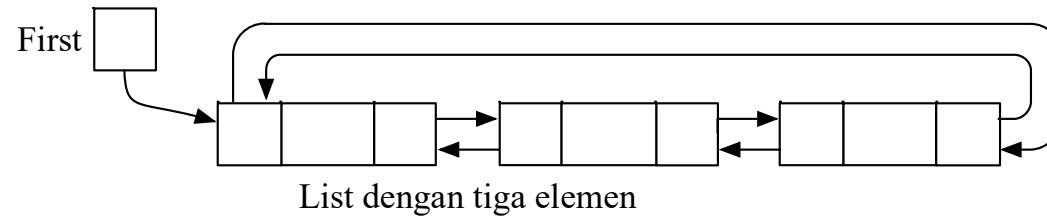


# List dengan Pointer Sirkuler Ganda

Elemen pertama ????

Elemen terakhir ???

List kosong ???




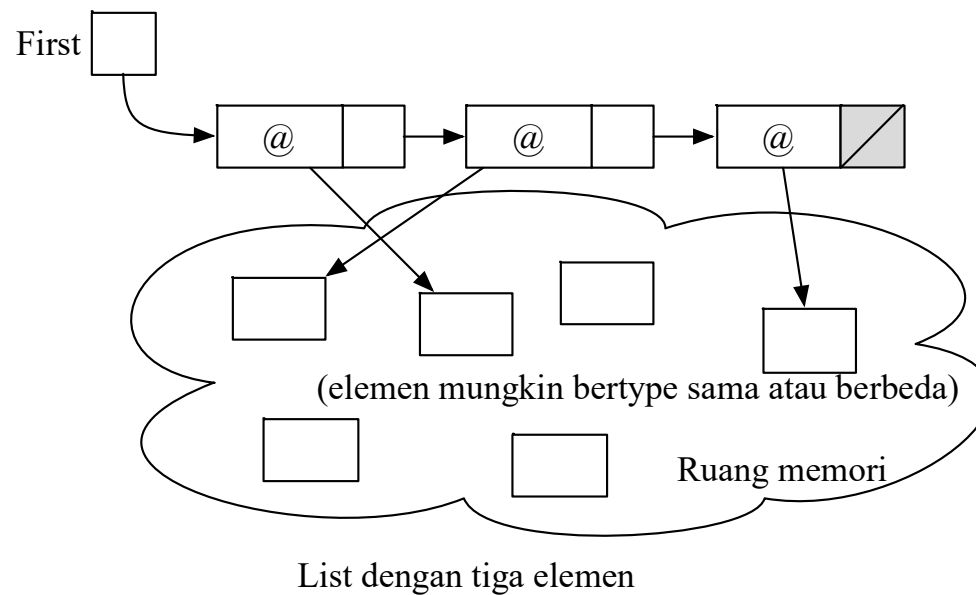
# List yang Informasi Elemennya adalah Alamat dari Suatu Informasi

Elemen pertama ????

Elemen terakhir ???

List kosong ???

First   
List kosong



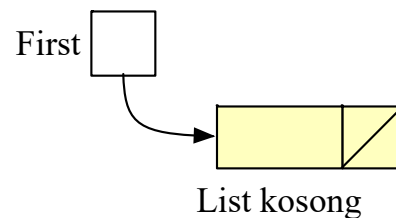
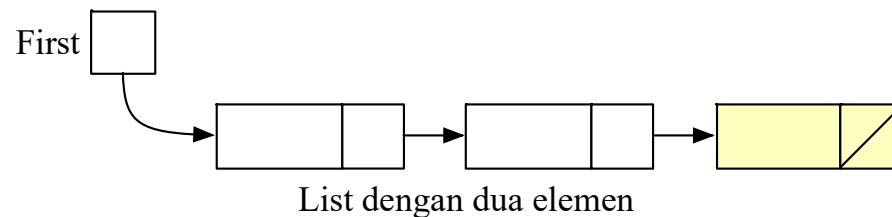
# List dengan *dummy element* di akhir

# List dengan “*Dummy Element*” pada Elemen Terakhir

Elemen pertama:  $\text{First}(l) = l$

Elemen terakhir: ber-address p, dengan  $\text{Next}(p) = \text{dummy@}$

List kosong:  $\text{First}(l) = \text{dummy@}$



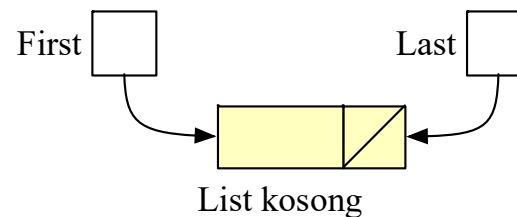
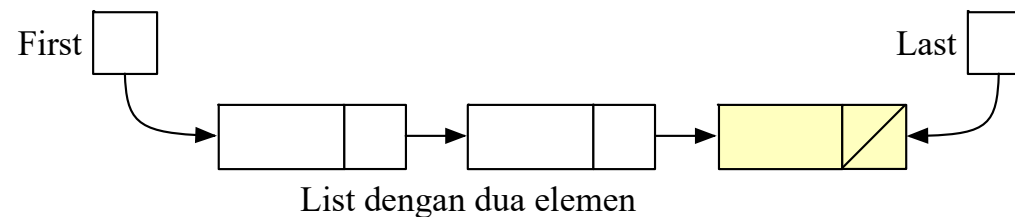


# List dengan *Dummy Element* di Akhir dan Pencatatan Alamat Elemen Akhir

Elemen pertama:  $\text{First}(l) = l$

Elemen terakhir: ber-address  $p$ , dengan  $\text{Next}(p) = \text{dummy}@$

List kosong:  $\text{First}(l) = \text{Last}(l) = \text{dummy}@$



# 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)

# Variasi List Linier

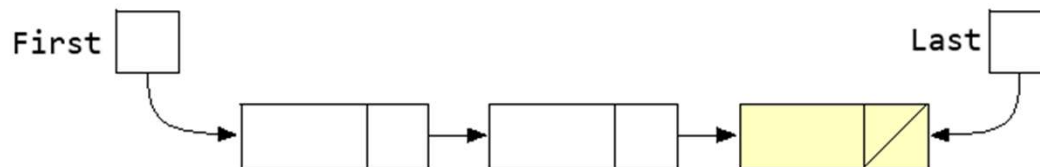
# List Linier dengan Dummy Element

IF2110/IF2111 – Algoritma dan Struktur Data  
Sekolah Teknik Elektro dan Informatika  
Institut Teknologi Bandung

# Bahasan

Buatlah ADT list + driver dengan elemen dummy di akhir:

- Penunjuk first dan last (last element menunjuk ke elemen dummy)
- Alamat dummy: tetap
- Representasi fisik: berkait dengan pointer



# Representasi Fisik dengan Pointer

```
#define NIL NULL
#define IDX_UNDEF -1
typedef int ElType;
typedef struct node* Address;
typedef struct node { ElType info; Address next; } Node;

/* 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 isEmpty(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(l) = LAST(l) = NIL dan list gagal terbentuk */
/***** SEARCHING *****/
int indexOf(List l, ElType x);
/* Mengembalikan indeks node list dengan nilai X, atau IDX_UNDEF jika tidak ada */
```

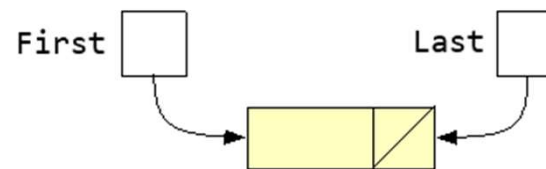
# Beberapa primitif

Buatlah sebagai latihan:

```
void insertFirst(List *l, ElType x);
/* I.S. List l terdefinisi */
/* F.S. Menambahkan elemen x sebagai elemen pertama List l */
void insertLast(List *l, ElType x);
/* I.S. List l terdefinisi */
/* F.S. x ditambahkan sebagai elemen terakhir l yang baru, */
/*      yaitu menjadi elemen sebelum elemen dummy */
void deleteFirst(List *l, ElType *x);
/* I.S. List l tidak kosong */
/* F.S. x adalah nilai elemen pertama list l sebelum penghapusan */
/*      Elemen list berkurang satu (mungkin menjadi kosong) */
/*      First element yg baru adalah suksesor elemen pertama yang lama */
void deleteLast(List *l, ElType *x);
/* I.S. List l tidak kosong */
/* F.S. x adalah terakhir sebelum dummy pada list sebelum penghapusan */
/*      Elemen list berkurang satu (mungkin menjadi kosong) */
```

# Beberapa primitif

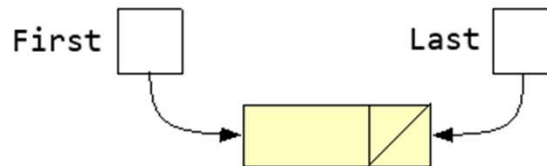
```
boolean isEmpty(List l) {  
  /* Mengirim true jika list kosong */  
  /* Kamus Lokal */  
  
  /* Algoritma */  
  return (FIRST(l) == LAST(l));  
}
```





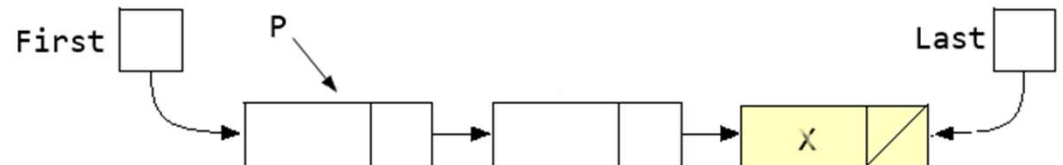
# Beberapa primitif

```
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(l) = LAST(l) = NIL dan list gagal terbentuk */  
    /* Kamus Lokal */  
    Address pDummy;  
  
    /* Algoritma */  
    pDummy = newNode(0); /* INFO(pDummy) tidak didefinisikan */  
    if (Pdummy != Nil) {  
        FIRST(*L) = pDummy;  
        LAST(*L) = pDummy;  
    } else /* List gagal terbentuk */ {  
        FIRST(*L) = NIL;  
        LAST(*L) = NIL;  
    }  
}
```



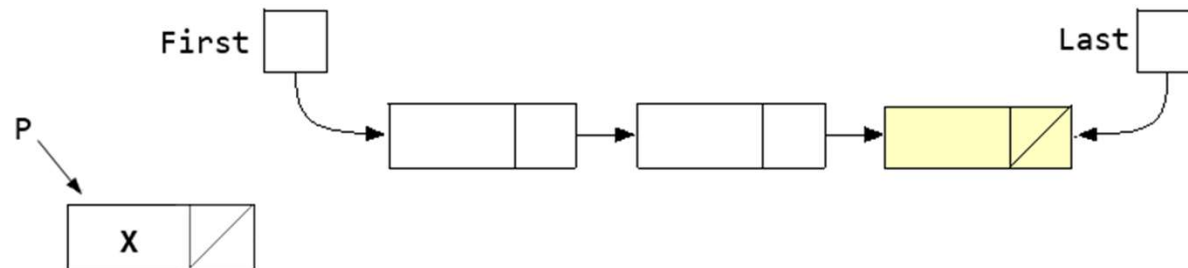
# Beberapa primitif

```
int indexOf(List l, ElType x) {  
  /* Mengembalikan indeks node list dengan nilai X, atau IDX_UNDEF jika tidak ada */  
  /* Kamus Lokal */  
  Address p;  
  int idx;  
  /* Algoritma */  
  INFO(LAST(l)) = x; /* letakkan X di sentinel */  
  p = FIRST(l);  
  idx = 0;  
  while (INFO(p) != x) {  
    p = NEXT(p);  
    idx++;  
  } /* INFO(p) = x */  
  if (p != LAST(l)) { /* bukan ketemu di sentinel */  
    return idx;  
  } else /* p = LAST(l), ketemu di sentinel */ {  
    return IDX_UNDEF;  
  }  
}
```



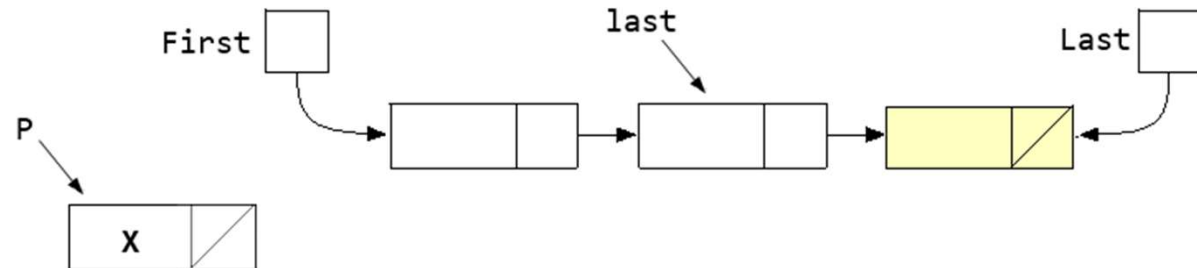
# Beberapa primitif

```
void insertFirst(List *l, ElType x) {  
  /* I.S. List l terdefinisi */  
  /* F.S. Menambahkan elemen x sebagai elemen pertama List l */  
  /* Kamus Lokal */  
  Address p;  
  
  /* Algoritma */  
  p = newNode(x);  
  if (p != NIL) {  
    NEXT(p) = FIRST(*l);  
    FIRST(*l) = p;  
  }  
}
```



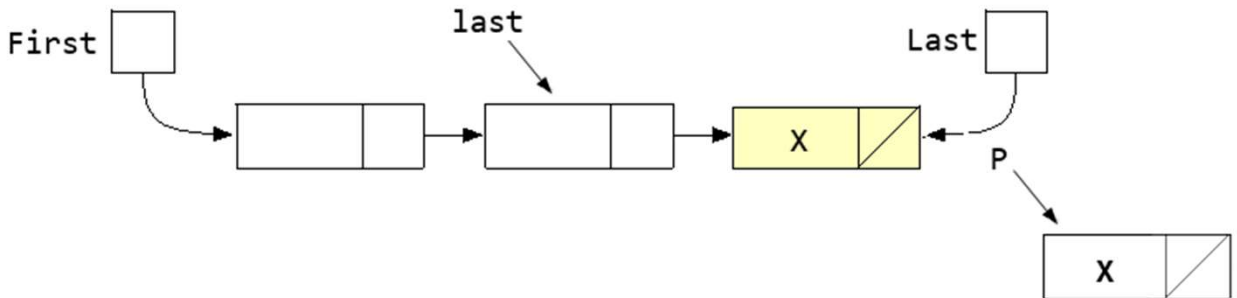
# Beberapa primitif

```
void insertLast(List *l, ElType x) {  
    /* I.S. List l terdefinisi */  
    /* F.S. x ditambahkan sebagai elemen terakhir l yang baru */  
    /* Alamat elemen dummy tidak berubah */  
    /* Kamus Lokal */  
    Address p, last;  
  
    /* Algoritma */  
    if (isEmpty(*l)) {  
        insertFirst(l,x);  
    } else {  
        p = newNode(x);  
        if (p != NIL) {  
            last = FIRST(*l);  
            while (NEXT(last) != LAST(*l)) {  
                last = NEXT(last);  
            } /* NEXT(last) == LAST(*l) alias dummy */  
            NEXT(last) = p;  
            NEXT(p) = LAST(*l);  
        }  
    }  
}
```



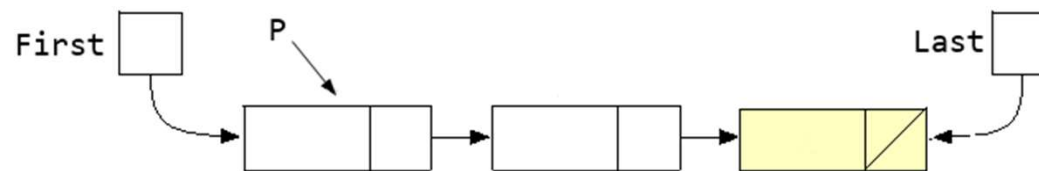
# Beberapa primitif

```
void insertLast(List *l, ElType x) {  
  /* I.S. List l terdefinisi */  
  /* F.S. x ditambahkan sebagai elemen terakhir l yang baru */  
  /* Versi jika alamat elemen dummy boleh berubah */  
  /* Kamus Lokal */  
  
  /* Algoritma */  
  if (isEmpty(*l)) {  
    insertFirst(l,x);  
  } else {  
    INFO(LAST(*l)) = x;  
    p = newNode(x); /* dummy baru */  
    if (p != NIL) {  
      NEXT(LAST(*l)) = p;  
      LAST(*l) = p;  
    }  
  }  
}
```



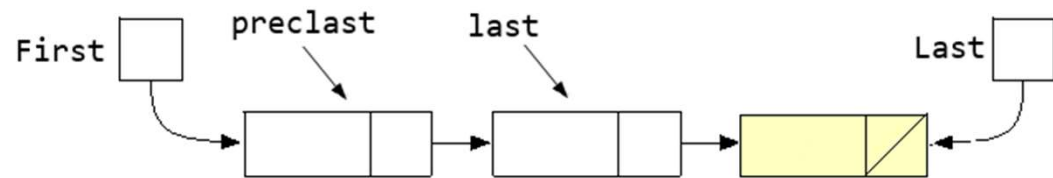
# Beberapa primitif

```
void deleteFirst(List *l, ElType *x) {  
    /* I.S. List l tidak kosong */  
    /* F.S. x adalah elemen pertama list l sebelum penghapusan */  
    /*      Elemen list berkurang satu (mungkin menjadi kosong) */  
    /*      First element yg baru adalah suksesor elemen pertama yang lama */  
    /* Kamus Lokal */  
    Address p;  
  
    /* Algoritma */  
    p = FIRST(*l);  
    *x = INFO(p);  
    FIRST(*l) = NEXT(FIRST(*l));  
    free(p);  
}
```



# Beberapa primitif

```
void deleteLast(List *l, ElType *x) {  
    /* I.S. List l tidak kosong */  
    /* F.S. x adalah elemen terakhir list l sebelum penghapusan */  
    /*     Elemen list berkurang satu (mungkin menjadi kosong) */  
    /*     Last element baru adalah predesesor elemen terakhir 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);  
    }  
    *x = INFO(last);  
    if (preclast == NIL) { /* kasus satu elemen */  
        FIRST(*l) = LAST(*l);  
    } else {  
        NEXT(preclast) = LAST(*l);  
    }  
    free(last);  
}
```



# Variasi List Linier

# List dengan Pointer Ganda

IF2110/IF2111 – Algoritma dan Struktur Data  
Sekolah Teknik Elektro dan Informatika  
Institut Teknologi Bandung

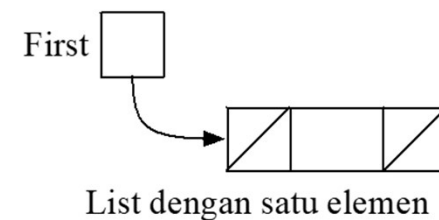
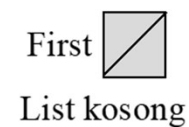
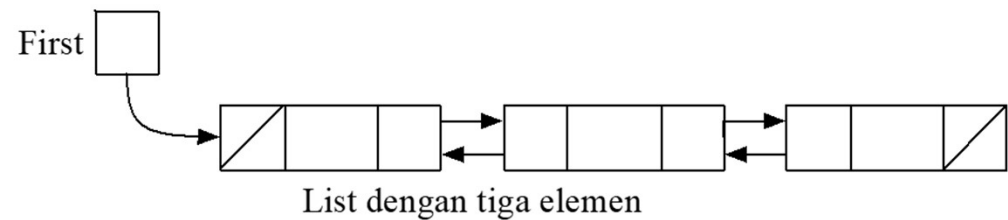
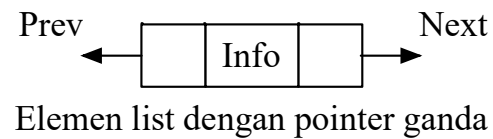


# List dengan Pointer Ganda

Elemen pertama:  $\text{First}(L)$

Elemen terakhir:  $\text{Next}(P) = \text{Nil}$

List kosong:  $\text{First}(L) = \text{Nil}$

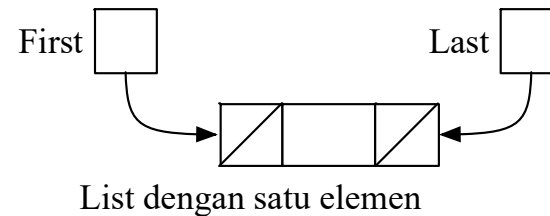
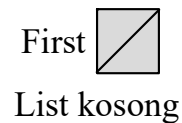
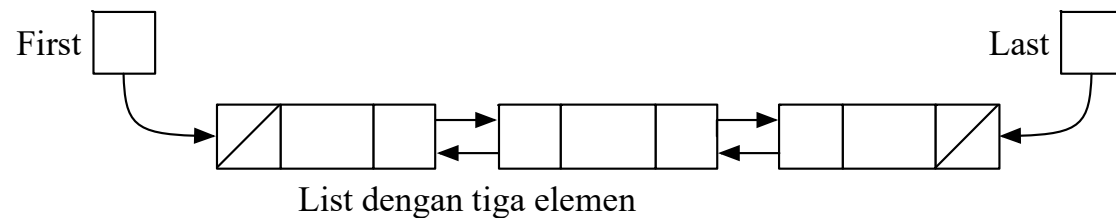


# List dengan Pointer Ganda + Pencatatan Last

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 logik semacam ini direpresentasi secara kontigu dengan tabel, maka sangat menguntungkan karena memorisasi Prev dan Next dilakukan dengan kalkulasi

# Bahasan

Buatlah ADT list dengan elemen berpointer ganda dilengkapi driver:

- Representasi fisik: berkait dengan pointer
- Penunjuk First dan Last

# Rep. Fisik dengan Pointer

```
#define NIL NULL
typedef int ElType;
typedef struct node *Address;
typedef struct node {
    ElType info;
    Address prev;
    Address next;
} Node;
/* 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, ElType x);  
/* I.S. List l terdefinisi */  
/* F.S. Menambahkan elemen baru x sebagai elemen pertama */  
  
void insertLast(List *l, ElType x);  
/* I.S. List l terdefinisi */  
/* F.S. x ditambahkan sebagai elemen terakhir l yang baru */
```

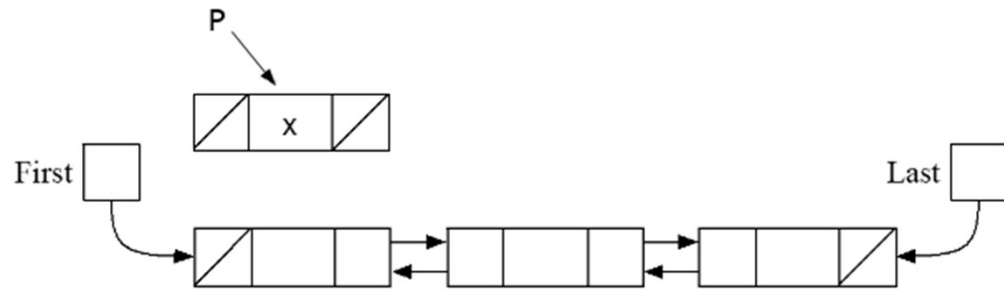
# Beberapa primitif

Buatlah sebagai latihan:

```
void deleteFirst(List *l, ElType *x);  
/* I.S. List l tidak kosong */  
/* F.S. x adalah elemen pertama list l sebelum penghapusan */  
/*      Elemen list berkurang satu (mungkin menjadi kosong) */  
/*      First element yg baru adalah suksesor elemen pertama yang lama */  
  
void deleteLast(List *l, ElType *x) {  
/* I.S. List l tidak kosong */  
/* F.S. x adalah elemen terakhir list l sebelum penghapusan */  
/*      Elemen list berkurang satu (mungkin menjadi kosong) */  
/*      Last element baru adalah predesesor elemen pertama yg lama, jika ada */
```

# Beberapa primitif

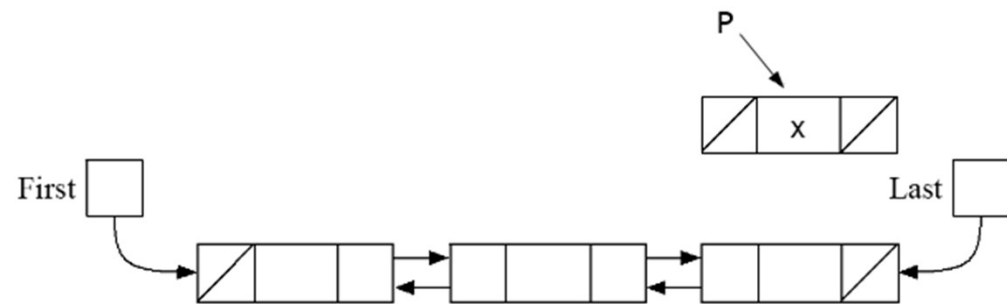
```
void insertFirst(List *l, ElType x);  
/* I.S. List l terdefinisi */  
/* F.S. Menambahkan elemen baru x sebagai elemen pertama */  
/* Kamus Lokal */  
Address p;  
  
/* Algoritma */  
p = newNode(x);  
if (p != NIL) {  
    NEXT(p) = FIRST(*l);  
    if (!isEmpty(*l)) {  
        PREV(FIRST(*l)) = p;  
    } else /* l kosong */ {  
        LAST(*l) = p;  
    }  
    FIRST(*l) = p;  
}
```





# Beberapa primitif

```
void insertLast(List *l, ElType x);  
/* I.S. List l terdefinisi */  
/* F.S. x ditambahkan sebagai elemen terakhir l yang baru */  
/* Kamus Lokal */  
Address p;  
  
/* Algoritma */  
p = newNode(x);  
if (p != NIL) {  
    PREV(p) = LAST(*l);  
    if (!isEmpty(*l)) {  
        NEXT(LAST(*l)) = p;  
    } else /* l kosong */ {  
        FIRST(*l) = p;  
    }  
    LAST(*l) = p;  
}
```



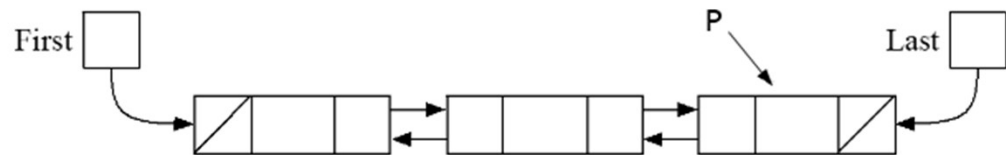
# Beberapa primitif

```
void deleteFirst(List *l, ElType *x);  
/* I.S. List l tidak kosong */  
/* F.S. x adalah elemen pertama list l sebelum penghapusan */  
/*      Elemen list berkurang satu (mungkin menjadi kosong) */  
/*      First element yg baru adalah suksesor elemen pertama yang lama */  
/* Kamus Lokal */  
Address p;  
  
/* Algoritma */  
p = FIRST(*l);  
*x = INFO(p);  
if (FIRST(*l) == LAST(*l)) { /* l hanya 1 elemen */  
    LAST(*l) = NIL;  
} else { /* l > 1 elemen */  
    PREV(NEXT(FIRST(*l))) = NIL;  
}  
FIRST(*l) = NEXT(FIRST(*l));  
free(P);  
}
```



# Beberapa primitif

```
void deleteLast(List *l, ElType *x) {  
    /* I.S. List l tidak kosong */  
    /* F.S. x adalah elemen terakhir list l sebelum penghapusan */  
    /*      Elemen list berkurang satu (mungkin menjadi kosong) */  
    /*      Last element baru adalah predesesor elemen pertama yg lama, jika ada */  
    /* Kamus Lokal */  
    Address p;  
  
    /* Algoritma */  
    p = LAST(*l);  
    *x = INFO(p);  
    if (FIRST(*l) == LAST(*l)) { /* l hanya 1 elemen */  
        FIRST(*l) = NIL;  
    } else { /* L > 1 elemen */  
        NEXT(PREV(LAST(*l))) = NIL;  
    }  
    LAST(*l) = PREV(LAST(*l));  
    free(p);  
}
```



# Variasi List Linier List Sirkuler

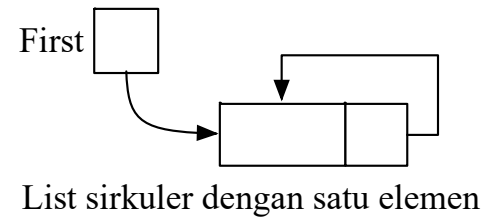
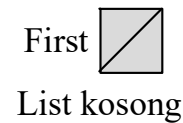
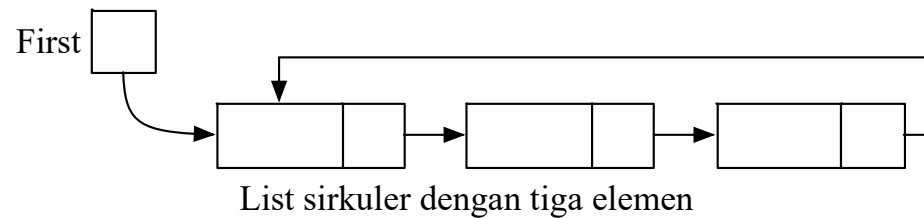
IF2110/IF2111 – Algoritma dan Struktur Data  
Sekolah Teknik Elektro dan Informatika  
Institut Teknologi Bandung

# 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 ElType;
typedef struct node* Address;
typedef struct node { ElType info; Address next; } Node;

/* 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(l) */
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 addrSearch(List l, Address p);  
/* Mencari apakah ada elemen list l yang beralamat p */  
/* Mengirimkan true jika ada, false jika tidak ada */  
void insertFirst(List *l, ElType x);  
/* I.S. List l terdefinisi */  
/* F.S. Menambahkan elemen bernilai x sebagai elemen pertama */  
void insertLast(List *l, ElType x);  
/* I.S. List l terdefinisi */  
/* F.S. x ditambahkan sebagai elemen terakhir l yang baru */
```

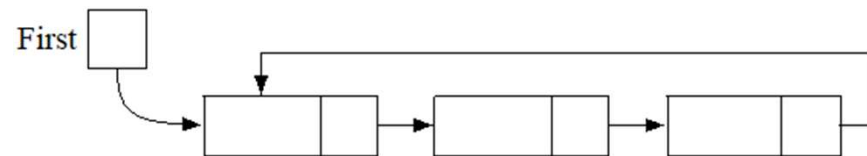
# Beberapa primitif

Buatlah sebagai latihan:

```
void deleteFirst(List *l, ElType *x);
/* I.S. List l tidak kosong */
/* F.S. x adalah elemen pertama list l sebelum penghapusan */
/*      Elemen list berkurang satu (mungkin menjadi kosong) */
/*      First element yg baru adalah suksesor elemen pertama yang lama */
void deleteLast(List *l, ElType *x);
/* I.S. List l tidak kosong */
/* F.S. x adalah elemen terakhir list sebelum penghapusan */
/*      Elemen list berkurang satu (mungkin menjadi kosong) */
/*      Last element baru adalah predesesor elemen pertama yg lama, jika ada */
void displayList(List l);
/* I.S. List l mungkin kosong */
/* F.S. Jika list tidak kosong, semua nilai (info) yg disimpan pada elemen list diprint */
/*      Jika list kosong, hanya menuliskan "list kosong" */
```

# Beberapa primitif

```
boolean addrSearch(List l, Address p) {  
  /* Mencari apakah ada elemen list l yang beralamat p */  
  /* Mengirimkan true jika ada, false jika tidak ada */  
  /* Kamus Lokal */  
  Address pt;  
  /* Algoritma */  
  if (isEmpty(l)) {  
    return false;  
  } else {  
    pt = FIRST(l);  
    while ((NEXT(pt) != FIRST(l)) && (pt != p)) {  
      pt = NEXT(pt);  
    }  
    /* NEXT(pt) = FIRST(l) or pt = p */  
    return pt == p;  
  }  
}
```

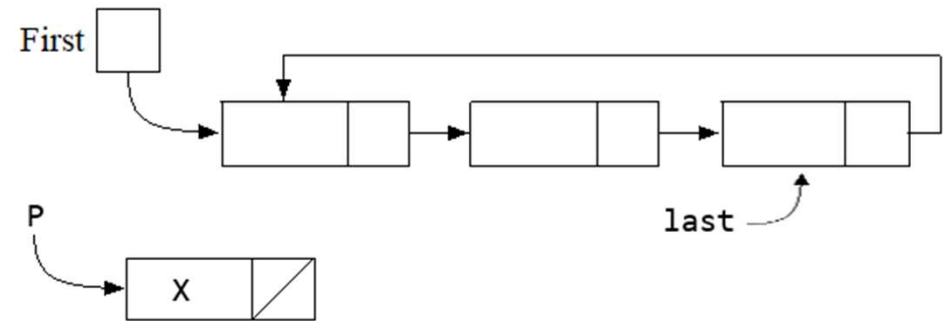


# Beberapa primitif

```

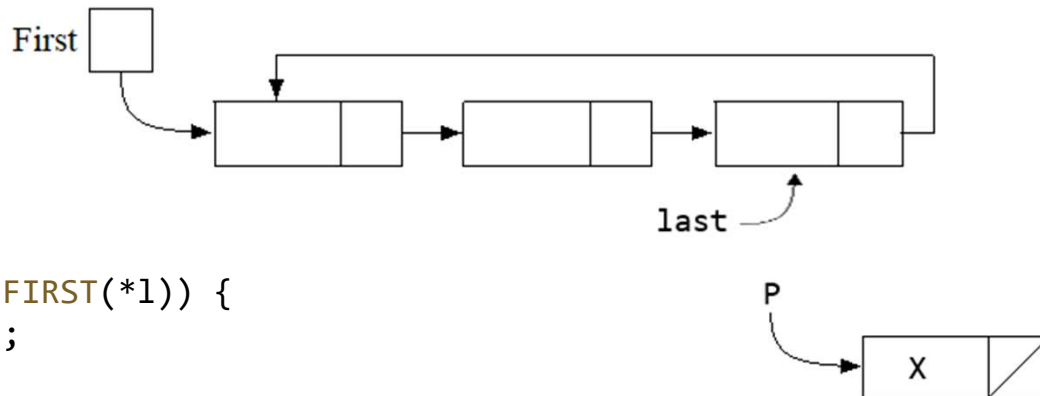
void insertFirst(List *l, ElType x) {
/* I.S. List l terdefinisi */
/* F.S. Menambahkan elemen bernilai x sebagai elemen pertama l */
/* Kamus Lokal */
    Address p, last;
/* Algoritma */
    p = newNode(x);
    if (p != NIL) {
        if (isEmpty(*l)) {
            NEXT(p) = p;
        } else /* *l 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;
    }
}

```



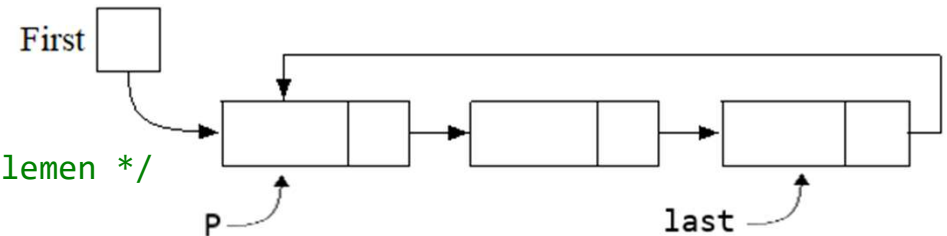
# Beberapa primitif

```
void insertLast(List *l, ElType x) {  
  /* I.S. List l terdefinisi */ /* F.S. x ditambahkan sebagai elemen terakhir l yang baru */  
  /* Kamus Lokal */  
  Address p, last;  
  /* Algoritma */  
  if (isEmpty(*l)) {  
    insertFirst(l,x);  
  } else {  
    p = newNode(x);  
    if (p != NIL) {  
      last = FIRST(*l);  
      while (NEXT(last) != FIRST(*l)) {  
        last = NEXT(last);  
      }  
      /* NEXT(last) = FIRST(*l) */  
      NEXT(last) = p;  
      NEXT(p) = FIRST(*l);  
    }  
  }  
}
```



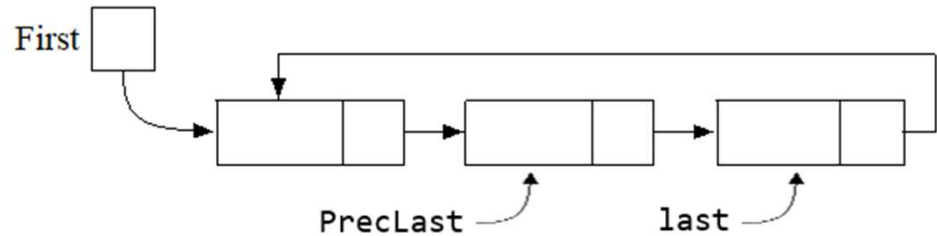
# Beberapa primitif

```
void deleteFirst(List *l, ElType *x) {  
    /* I.S. List tidak kosong */  
    /* F.S. x adalah elemen pertama list l sebelum penghapusan */  
    /*      Elemen list berkurang satu (mungkin menjadi kosong) */  
    /*      First element yg baru adalah suksesor elemen pertama yang lama */  
    /* Kamus Lokal */  
    Address p, last;  
    /* Algoritma */  
    p = FIRST(*l); *x = INFO(p);  
    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);  
    }  
    free(p);  
}
```



# Beberapa primitif

```
void deleteLast(List *l, ElType *x) {  
  /* I.S. List l tidak kosong */  
  /* F.S. x adalah elemen terakhir list l sebelum penghapusan */  
  /*      Elemen list berkurang satu (mungkin menjadi kosong) */  
  /*      Last element baru adalah predesesor elemen pertama yg lama, jika ada */  
  /* Kamus Lokal */  
  Address last, preLast;  
  /* Algoritma */  
  last = FIRST(*l); preLast = NIL;  
  while (NEXT(last) != FIRST(*l)) {  
    preLast = last; last = NEXT(last);  
  } /* NEXT(last) = FIRST(*l) */  
  if (preLast == NIL) { /* kasus satu elemen */  
    FIRST(*l) = NIL;  
  } else {  
    NEXT(preLast) = FIRST(*l);  
  }  
  *x = INFO(last);  
  free(last);  
}
```



# Beberapa primitif

```
void displayList(List l) {
/* I.S. List l mungkin kosong */
/* F.S. Jika list tidak kosong, semua nilai (info) yg disimpan pada elemen list diprint */
/*      Jika list kosong, hanya menuliskan "list kosong" */
/* Kamus Lokal */
    Address p;

    /* Algoritma */
    if (isEmpty(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));
    }
}
```



# Latihan Soal Variasi List Linier

IF2110/IF2111 – Algoritma dan Struktur Data  
Sekolah Teknik Elektro dan Informatika  
Institut Teknologi Bandung

# Soal No. 1

Buatlah prosedur **insertFirst** untuk sebuah list *l* dengan *l* adalah List yang mencatat elemen pertama (First) dan elemen terakhir (Last)

procedure insertFirst(input/output *l*:List, input *x*:ElType)

List *l* mungkin kosong.

## Soal No. 2

Buatlah prosedur **insertLast** dari sebuah list l dengan l adalah List yang mencatat elemen pertama (First) dan elemen terakhir (Last)

procedure insertLast(input/output l:List, input x:ElType)

List l mungkin kosong.

## Soal No. 3

Buatlah fungsi **search** untuk mengetahui apakah sebuah nilai x terdapat dalam sebuah list l. l adalah List yang mencatat elemen pertama (First) dan elemen terakhir (Last) dan elemen terakhir adalah dummy.

function search(l:List, x:ElType) → boolean

List l mungkin kosong

# Stack dengan Struktur Berkait

IF2110/IF2111 – Algoritma dan Struktur Data  
Sekolah Teknik Elektro dan Informatika  
Institut Teknologi Bandung

# Stack

**Stack**, sederetan elemen yang:

- dikenali elemen puncaknya (Top)
- aturan penambahan dan penghapusan elemennya tertentu:  
**Penambahan** selalu dilakukan **"di atas" Top**  
**Penghapusan** selalu dilakukan **pada Top**

Top adalah satu-satunya lokasi terjadinya operasi

Elemen Stack tersusun secara LIFO (Last In First Out)

# Definisi operasi

Jika diberikan  $S$  adalah Stack dengan elemen  $ElmtS$

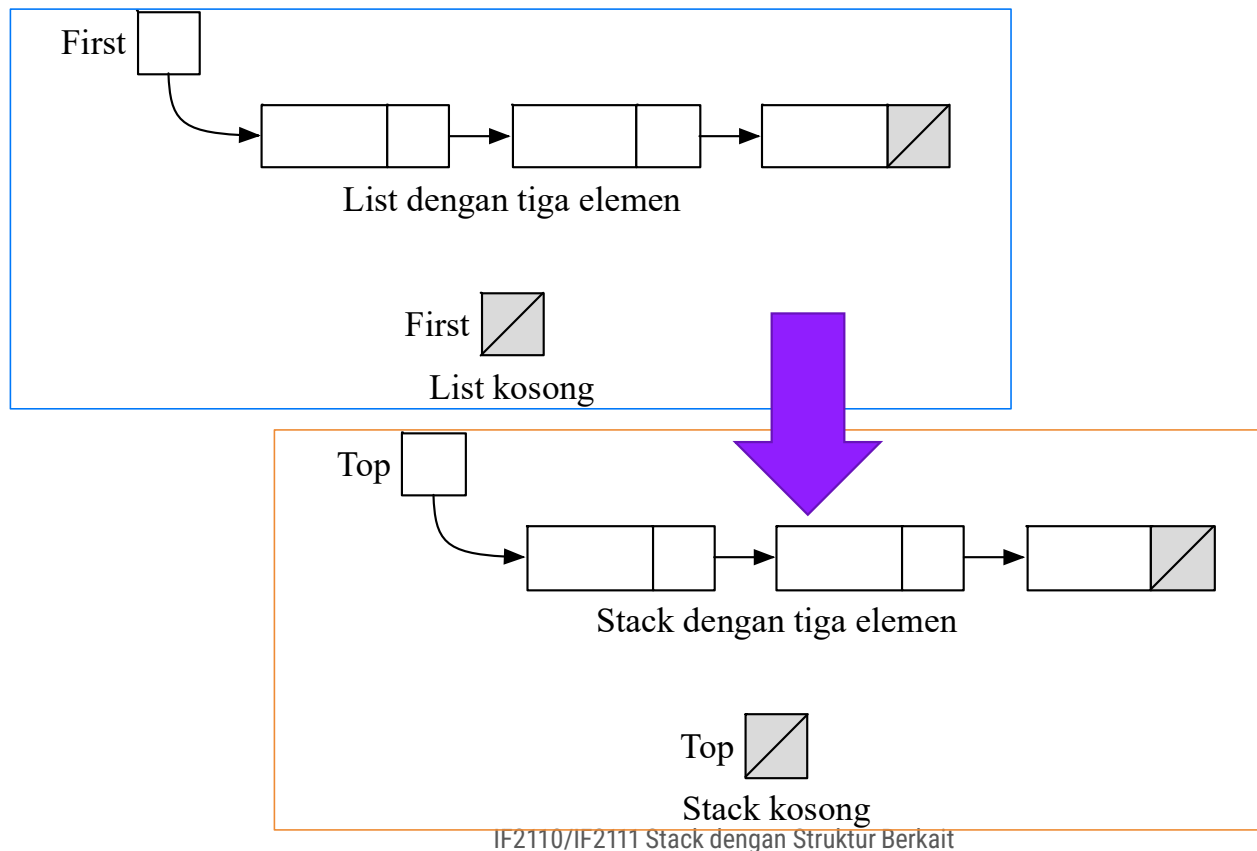
$CreateStack: \rightarrow S$	{ Membuat sebuah tumpukan kosong }
$top: S \rightarrow ElmtS$	{ Mengirimkan elemen teratas $S$ saat ini }
$length: S \rightarrow \underline{integer}$	{ Mengirimkan banyaknya elemen $S$ saat ini }
$push: ElmtS \times S \rightarrow S$	{ Menambahkan sebuah elemen $ElmtS$ sebagai TOP, TOP berubah nilainya }
$pop: S \rightarrow S \times ElmtS$	{ Mengambil nilai elemen TOP, sehingga TOP yang baru adalah elemen yang datang sebelum elemen TOP, mungkin $S$ menjadi kosong }
$isEmpty: S \rightarrow \underline{boolean}$	{ Test stack kosong, true jika $S$ kosong, false jika $S$ tidak kosong }

**Representasi berkait seperti apa yang paling cocok untuk stack?**



# Stack dengan Struktur Berkait

List linier “biasa”  $\approx$  stack



# Operasi-Operasi Dasar pada Stack

CreateStack  $\approx$  CreateList

push  $\approx$  insertFirst

pop  $\approx$  deleteFirst

length  $\approx$  length

isEmpty  $\approx$  isEmpty

# ADT Stack Representasi List

```
/* File: stack_linked.h */
#ifndef STACK_LINKED_H
#define STACK_LINKED_H
#include "boolean.h"
#include <stdlib.h>

#define NIL NULL
/* Deklarasi infotype */
typedef int ElType;
/* Stack dengan representasi berkait dengan pointer */
typedef struct node* Address;
typedef struct node {
    ElType info;
    Address next;
} Node;

/* Type stack dengan ciri Top: */
typedef struct {
    Address addrTop; /* alamat Top: elemen puncak */
} Stack;
```

# ADT Stack Representasi List

```
/* Selektor */
#define NEXT(p) (p)->next
#define INFO(p) (p)->info
#define ADDR_TOP(s) (s).addrTop
#define TOP(s) (s).addrTop->Info

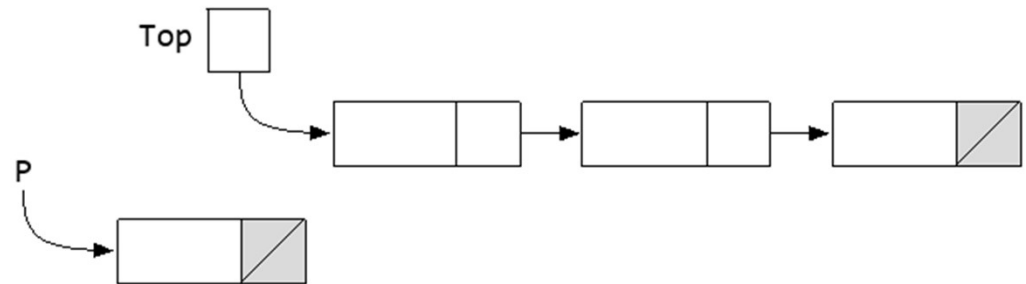
/* Prototype manajemen memori */
Address newNode(EIType x);
/* Mengembalikan alamat sebuah Node hasil alokasi dengan info = x,
   atau NIL jika alokasi gagal */
```

# ADT Stack Representasi List

```
/* ***** PROTOTYPE REPRESENTASI LOGIK STACK ***** */
boolean isEmpty(Stack s);
/* Mengirim true jika Stack kosong: TOP(s) = NIL */
void CreateStack(Stack *s);
/* I.S. sembarang */
/* F.S. Membuat sebuah stack s yang kosong */
void push(Stack *s, ElType x);
/* Menambahkan x sebagai elemen Stack s */
/* I.S. s mungkin kosong, x terdefinisi */
/* F.S. x menjadi Top yang baru jika alokasi x berhasil, */
/*      jika tidak, s tetap */
/* Pada dasarnya adalah operasi insertFirst pada list linier */
void pop(Stack *s, ElType *x);
/* Menghapus Top dari Stack s */
/* I.S. s tidak kosong */
/* F.S. x adalah nilai elemen Top yang lama, */
/*      elemen Top yang lama didealokasi */
/* Pada dasarnya adalah operasi deleteFirst pada list linier */
#endif
```

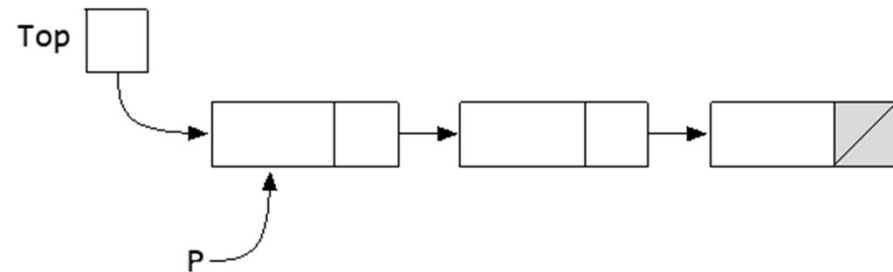
# ADT Stack Representasi List

```
void push(Stack *s, ElType x);  
/* Menambahkan x sebagai elemen Stack s */  
/* I.S. s mungkin kosong, x terdefinisi */  
/* F.S. x menjadi Top yang baru jika alokasi x berhasil, */  
/*     jika tidak, s tetap */  
/* Pada dasarnya adalah operasi insertFirst pada list linier */  
  
/* Kamus Lokal */  
Address p;  
  
/* Algoritma */  
p = newNode(x);  
if (p != NIL) {  
    NEXT(p) = ADDR_TOP(*s);  
    ADDR_TOP(*s) = p;  
} /* else: alokasi gagal, s tetap */  
}
```



# ADT Stack Representasi List

```
void pop(Stack *s, ElType *x);  
/* Menghapus Top dari Stack s */  
/* I.S. s tidak mungkin kosong */  
/* F.S. x adalah nilai elemen Top yang lama, */  
/*     elemen Top yang lama didealokasi */  
/* Pada dasarnya adalah operasi deleteFirst pada list linier */  
  
/* Kamus Lokal */  
Address p;  
  
/* Algoritma */  
*x = TOP(*s);  
p = ADDR_TOP(*s);  
ADDR_TOP(*s) = NEXT(ADDR_TOP(*s));  
NEXT(p) = NIL;  
free(p);  
}
```



# Queue dengan Struktur Berkait

IF2110/IF2111 – Algoritma dan Struktur Data  
Sekolah Teknik Elektro dan Informatika  
Institut Teknologi Bandung



# Queue



**Queue** adalah sederetan elemen yang:

- dikenali elemen pertama (HEAD) dan elemen terakhirnya (TAIL).
- aturan penambahan dan penghapusan elemennya didefinisikan sebagai berikut:  
**Penambahan** selalu dilakukan setelah **elemen terakhir**,  
**Penghapusan** selalu dilakukan pada **elemen pertama**.

# Definisi operasi

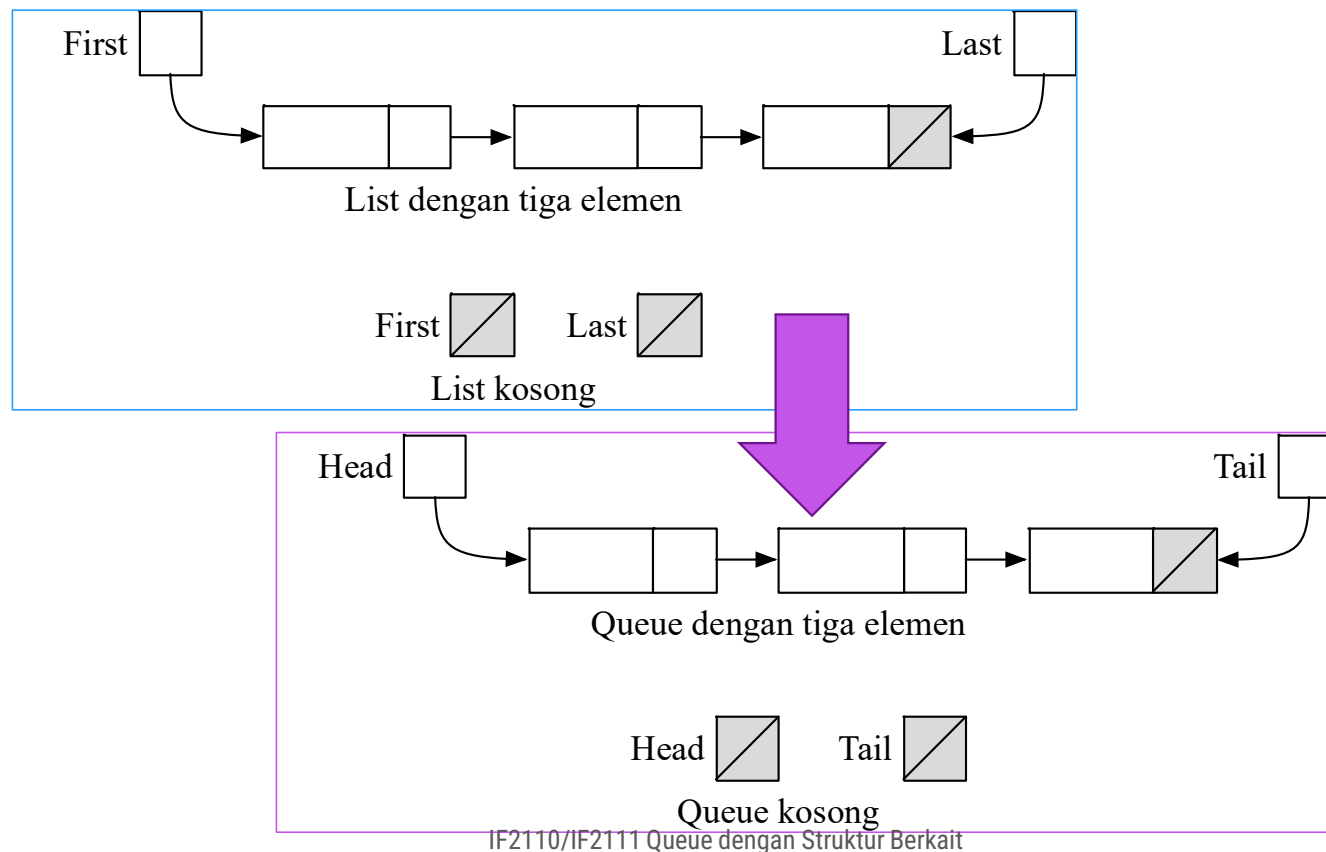
Jika diberikan  $Q$  adalah Queue dengan elemen  $ElmtQ$

$CreateQueue: \rightarrow Q$	{ Membuat sebuah antrian kosong }
$head: Q \rightarrow ElmtQ$	{ Mengirimkan elemen terdepan $Q$ saat ini }
$length: Q \rightarrow \underline{integer}$	{ Mengirimkan banyaknya elemen $Q$ saat ini }
$enqueue: ElmtQ \times Q \rightarrow Q$	{ Menambahkan sebuah elemen setelah elemen paling belakang Queue }
$dequeue: Q \rightarrow Q \times ElmtQ$	{ Menghapus kepala Queue, mungkin $Q$ menjadi kosong }
$isEmpty: Q \rightarrow \underline{boolean}$	{ Tes terhadap $Q$ : true jika $Q$ kosong, false jika $Q$ tidak kosong }

**Representasi berkait seperti apa yang paling cocok untuk queue?**

# Queue dengan Struktur berkait

List Linier yang dicatat first dan last  $\approx$  queue



# Operasi-operasi dasar pada Queue

CreateQueue  $\approx$  CreateList

enqueue  $\approx$  insertLast

dequeue  $\approx$  deleteFirst

length  $\approx$  length

isEmpty  $\approx$  isEmpty

# ADT Queue dengan Rep. List

```
/* File: queue_linked.h */
#ifndef QUEUE_LINKED_H
#define QUEUE_LINKED_H
#include "boolean.h"
#include <stdlib.h>

#define NIL NULL
/* Deklarasi infotype */
typedef int ElType;
/* Queue dengan representasi berkait dengan pointer */
typedef struct node* Address;
typedef struct node {
    ElType info;
    Address next;
} Node;

/* Type queue dengan ciri HEAD dan TAIL: */
typedef struct {
    Address addrHead; /* alamat penghapusan */
    Address addrTail; /* alamat penambahan */
} Queue;
```

# ADT Queue dengan Rep. List

```
/* Selektor */
#define NEXT(p) (p)->next
#define INFO(p) (p)->info

#define ADDR_HEAD(q) (q).addrHead
#define ADDR_TAIL(q) (q).addrTail
#define HEAD(q) (q).addrHead->info

/* Prototype manajemen memori */
Address newNode(ElType x);
/* Mengembalikan alamat sebuah Node hasil alokasi dengan info = x,
   atau NIL jika alokasi gagal */
```

# ADT Queue dengan Rep. List

```
boolean isEmpty(Queue q);
/* Mengirim true jika q kosong: ADDR_HEAD(q)=NIL and ADDR_TAIL(q)=NIL */
int length(Queue q);
/* Mengirimkan banyaknya elemen queue. Mengirimkan 0 jika q kosong */

/** Kreator */
void CreateQueue(Queue *q);
/* I.S. sembarang */
/* F.S. Sebuah q kosong terbentuk */

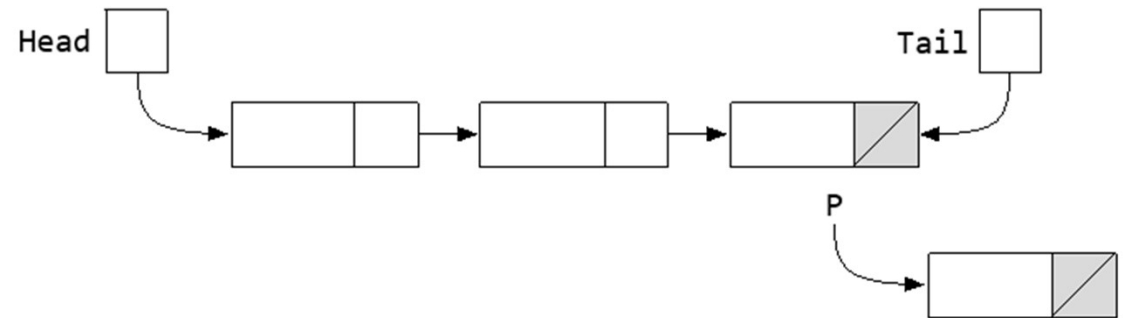
/** Primitif Enqueue/Dequeue */
void enqueue(Queue *q, ElType x);
/* Proses: Mengalokasi x dan menambahkan x pada bagian Tail dari q
   jika alokasi berhasil; jika alokasi gagal q tetap */
/* Pada dasarnya adalah proses insertLast */
/* I.S. q mungkin kosong */
/* F.S. x menjadi Tail, Tail "maju" */
void dequeue(Queue *q, ElType *x);
/* Proses: Menghapus x pada bagian HEAD dari q dan mendealokasi elemen HEAD */
/* Pada dasarnya operasi deleteFirst */
/* I.S. q tidak mungkin kosong */
/* F.S. x = nilai elemen HEAD pd I.S., HEAD "mundur" */
#endif
```



# ADT Queue dengan Rep. List

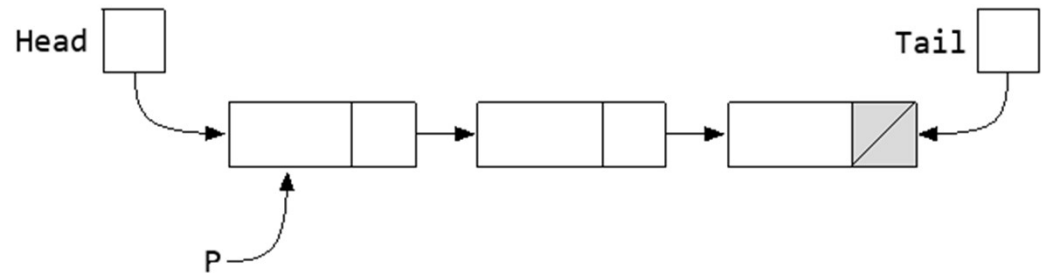
```
void enqueue(Queue *q, ElType x) {  
    /* Proses: Mengalokasi x dan menambahkan x pada bagian Tail dari q  
       jika alokasi berhasil; jika alokasi gagal q tetap */  
    /* Pada dasarnya adalah proses insertLast */  
    /* I.S. q mungkin kosong */  
    /* F.S. x menjadi Tail, Tail "maju" */
```

```
    /* Kamus Lokal */  
    Address p;  
    /* Algoritma */  
    p = newNode(x);  
    if (p != NIL) {  
        if (isEmpty(*q)) {  
            ADDR_HEAD(*q) = p;  
        } else {  
            NEXT(ADDR_TAIL(*q)) = p;  
        }  
        ADDR_TAIL(*q) = p;  
    } /* else: alokasi gagal, q tetap */  
}
```



# ADT Queue dengan Rep. List

```
void dequeue(Queue *q, ElType *x) {  
    /* Proses: Menghapus x pada bagian HEAD dari q dan mendealokasi  
       elemen HEAD */  
    /* Pada dasarnya operasi delete first */  
    /* I.S. q tidak mungkin kosong */  
    /* F.S. x = nilai elemen HEAD pd I.S., HEAD "mundur" */  
  
    /* Kamus Lokal */  
    Address p;  
    /* Algoritma */  
    *x = HEAD(*q);  
    p = ADDR_HEAD(*q);  
    ADDR_HEAD(*q) = NEXT(ADDR_HEAD(*q));  
    if (ADDR_HEAD(*q) == NIL) {  
        ADDR_TAIL(*q) = NIL;  
    }  
    NEXT(p) = NIL;  
    free(p);  
}
```



# Priority Queue

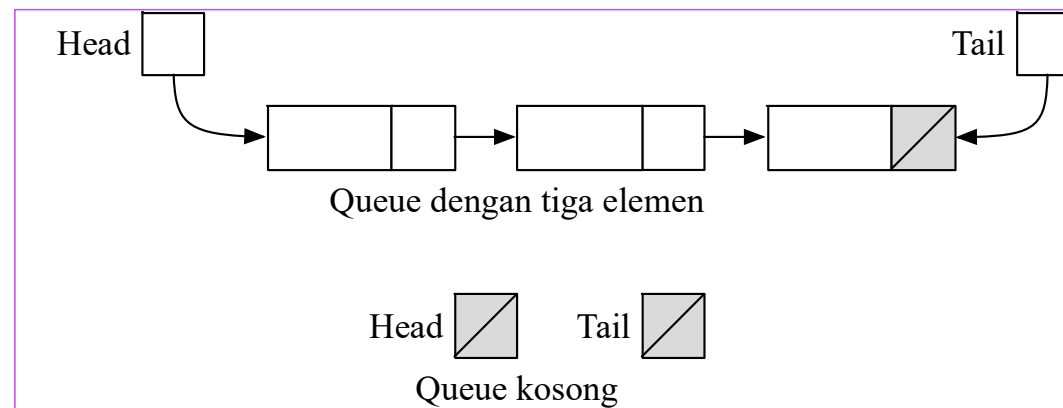
Priority queue:

- Elemen queue terurut menurut suatu prioritas tertentu
- Sering dianggap sebagai *modified queue*
- Menambahkan elemen berarti menambahkan elemen sesuai urutan prioritas
- Menghapus elemen adalah menghapus elemen dengan prioritas tertinggi/terendah (pada bagian Head)

**Representasi berkait seperti apa yang paling cocok untuk priority queue?**

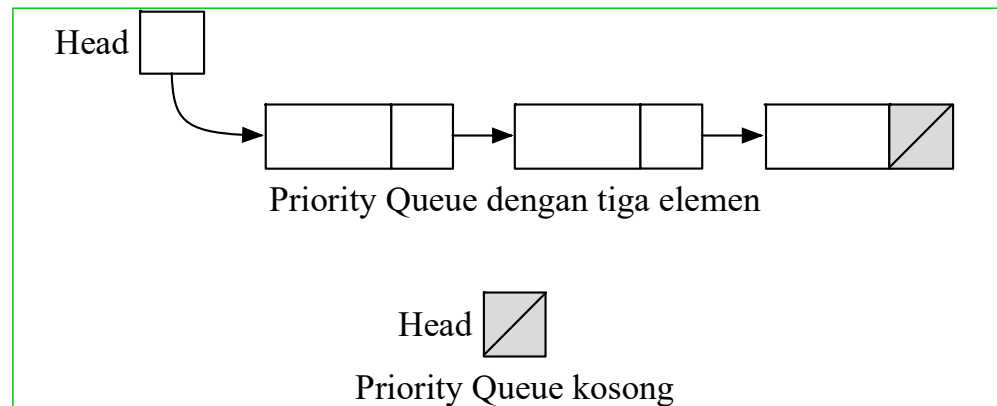
# Priority Queue dengan Rep. List

List Linier yang dicatat first dan last  $\approx$  queue “biasa”



# Priority Queue dengan Rep. List

List linier “biasa” ~ priority queue



# Operasi-operasi dasar Priority Queue

Node elemen queue:

```
type Node: < info: ElType,  
                prio: integer,  
                next: Address >
```

- Enqueue → menambahkan elemen secara terurut mengikuti prioritas tertentu
- Dequeue → menghapus elemen dengan prioritas tertinggi (yaitu di Head)
  - Pada dasarnya sama saja dengan Dequeue pada queue/list biasa

Perhatikan representasi logik yang digunakan



---

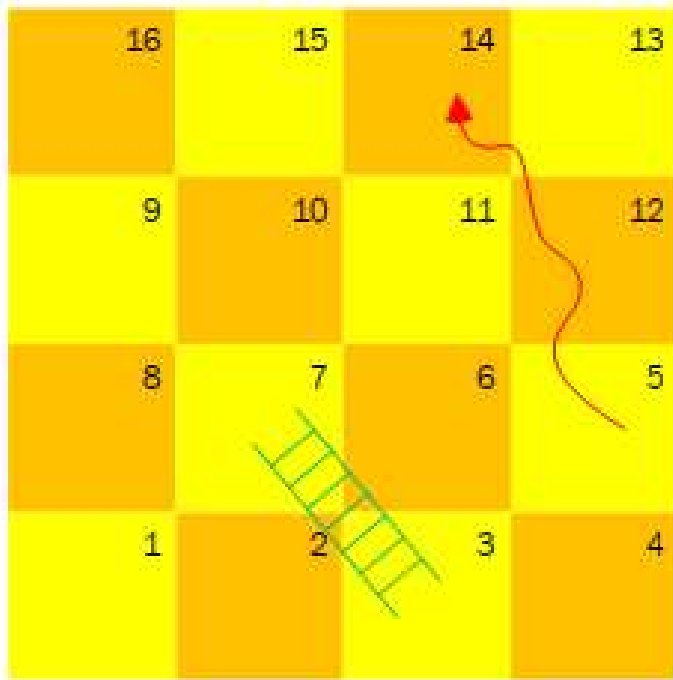
# Contoh kegunaan list berkait dengan *dummy element*

SAR

---

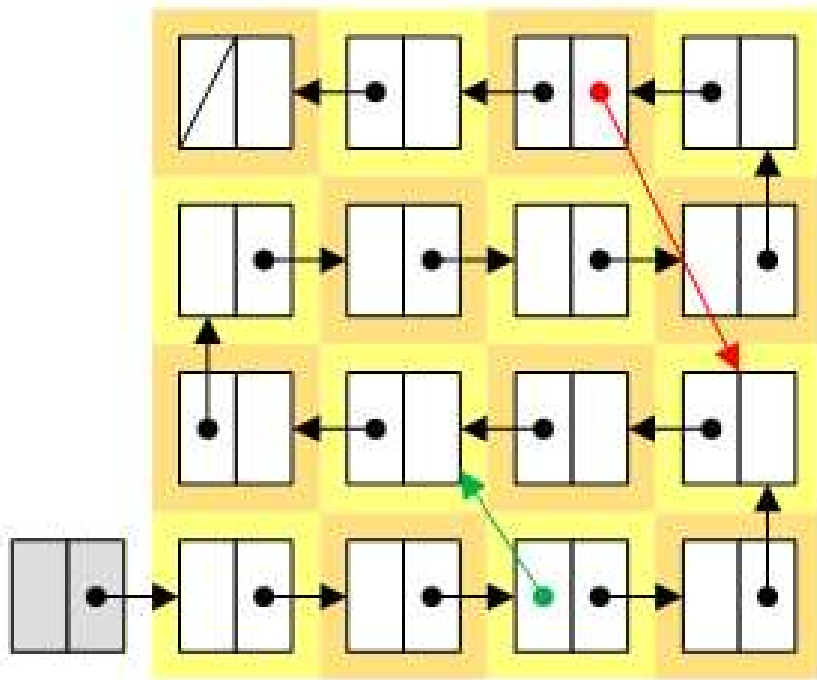


# Board game night, anyone?



- › “Snakes & ladders”
- › Dari India kuno (~tahun 2 A.D.)
- › Terdapat  $N$  buah kotak di papan, dinomori  $1..N$ .
  - › Umumnya  $N=100$ , tapi tidak selalu.
- › Pemain mulai dari luar papan dan menang jika berhasil mencapai kotak terakhir.

# Papan permainan dalam struktur berkait



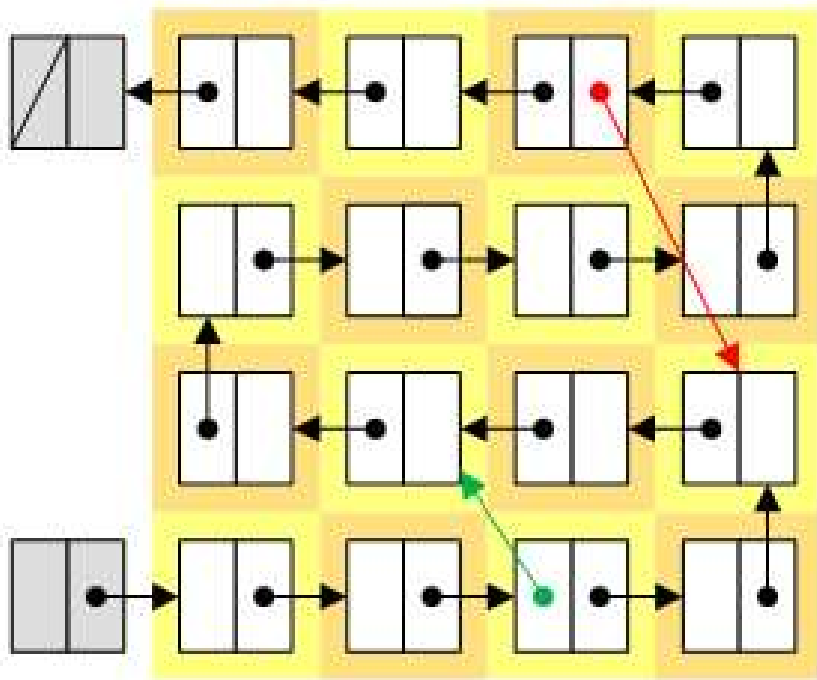
- › Mulai dari luar papan: *basically* menduduki “kotak nomor 0”.
- › Jika diimplementasikan dengan struktur berkait, dapat digunakan *dummy element* sebagai elemen pertama:
  - › Setiap pemain memulai permainan dengan pion menunjuk ke *dummy element*.

# Board game variations

---

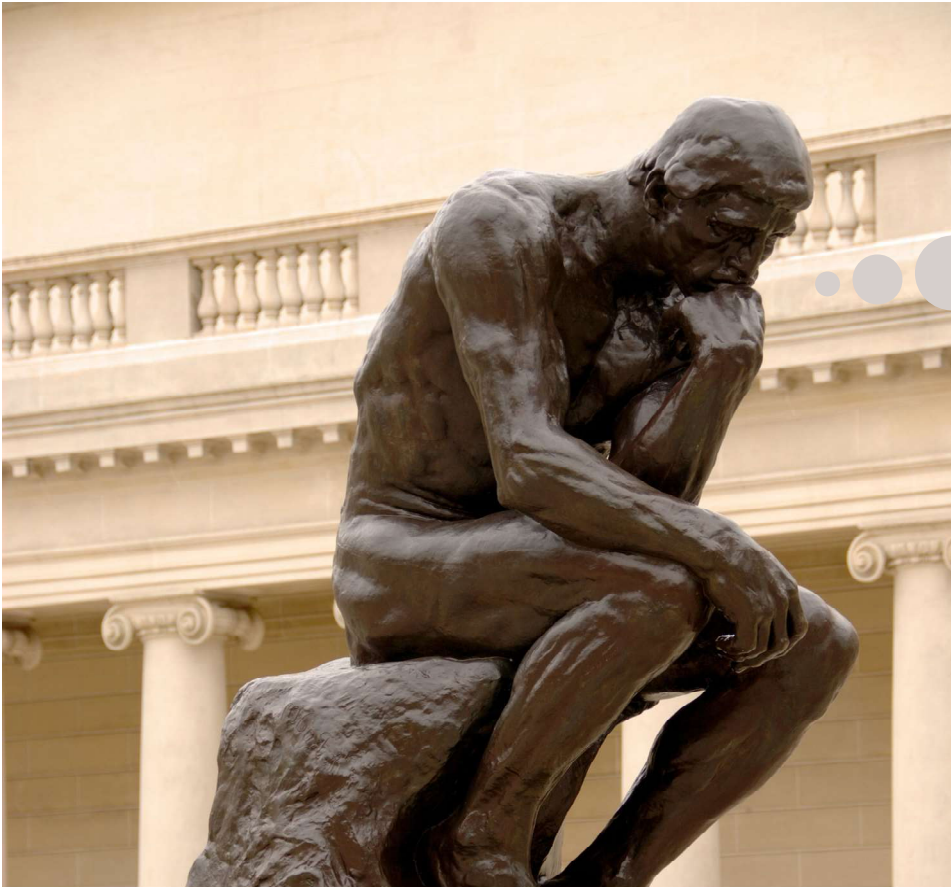
- › *Board game* biasanya memiliki variasi aturan permainan, ataupun ada permainan-permainan lain yang memiliki kemiripan aturan → kategori.
  - › Sebagai contoh, permainan ular tangga termasuk kategori permainan *racing*.
- › Beberapa permainan *racing* memiliki *winning condition* yang agak berbeda: pion pemain harus sampai keluar dari papan.
  - › Meminjam istilah yang tadi sudah kita gunakan, pion harus melampaui kotak nomor *N*.
- › Mari kita ubah sedikit aturan ular tangga dengan *winning condition* tersebut.

## Variasi: pemain harus melampaui kotak nomor $N$



- › Melampaui kotak nomor  $N$ : *basically* menduduki “kotak nomor  $N+1$ ”.
- › Dengan struktur berkait, dapat digunakan *dummy element* untuk elemen pertama dan elemen terakhir.
  - › Pemain dinyatakan menang jika pion menunjuk ke *dummy element* di akhir.
- › Meskipun jumlah kotak divariasikan, “kotak nomor 0” dan “kotak nomor  $N+1$ ” selalu diperlukan.

# Tapi...



...apakah implementasi  
*racing board game*  
dengan struktur berkait  
lebih baik dari pada  
*array*?

“Does anyone actually use LinkedList? I wrote it,  
and I never use it.” —[Joshua Bloch](#)  
(konteks: bahasa pemrograman Java)