IF1210 Algoritma dan Pemrograman 1

Stack dan Queue Representasi dengan Struktur Berkait

Tim Pengajar IF1210

Sekolah Teknik Elektro dan Informatika

Stack dengan Struktur Berkait

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

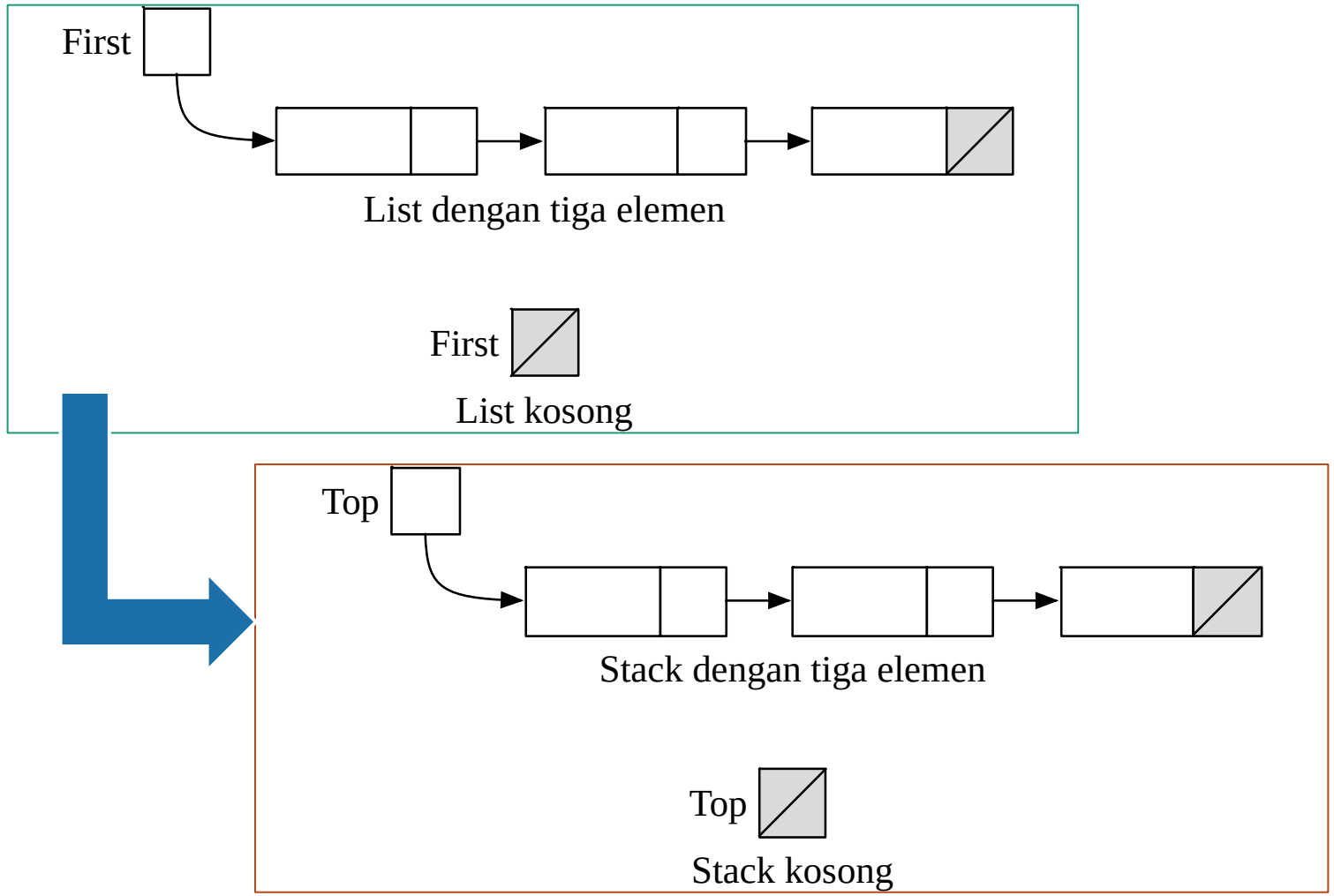
Jika diberikan *S* adalah Stack dengan elemen *ElmtS*

CreateStack: → *S*	{ Membuat sebuah tumpukan kosong }
top: *S* → *ElmtS*	{ Mengirimkan elemen teratas S saat ini }
length: *S* → *integer*	{ Mengirimkan banyaknya elemen S saat ini }
push: *ElmtS* × *S* → *S*	{ Menambahkan sebuah elemen ElmtS sebagai TOP, TOP berubah nilainya }
pop: *S* → *S* × *ElmtS*	{ Mengambil nilai elemen TOP, sehingga TOP yang baru adalah elemen yang datang sebelum elemen TOP, mungkin S menjadi kosong }
isEmpty: *S* → *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" ≃ stack



Operasi-Operasi Dasar pada Stack

- CreateStack ≍ CreateList
- push ≍ insertFirst
- pop ≍ deleteFirst
- length ≍ length
- isEmpty ≍ 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

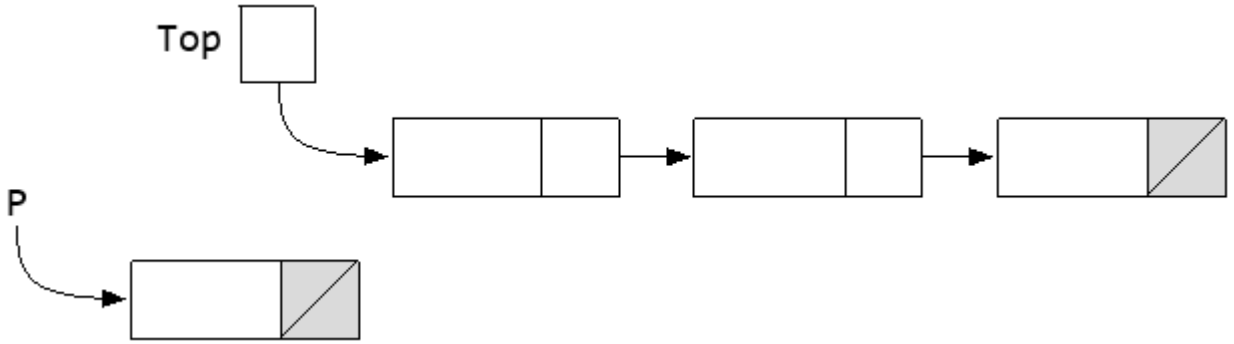
```
/* Selektor */
#define     NEXT(p) (p)->next
#define     INFO(p) (p)->info
#define ADDR_TOP(s) (s).addrTop
#define      TOP(s) (s).addrTop->Info

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

ADT Stack Representasi List

```
/* ********* PROTOTYPE REPRESENTASI LOJIK 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
```

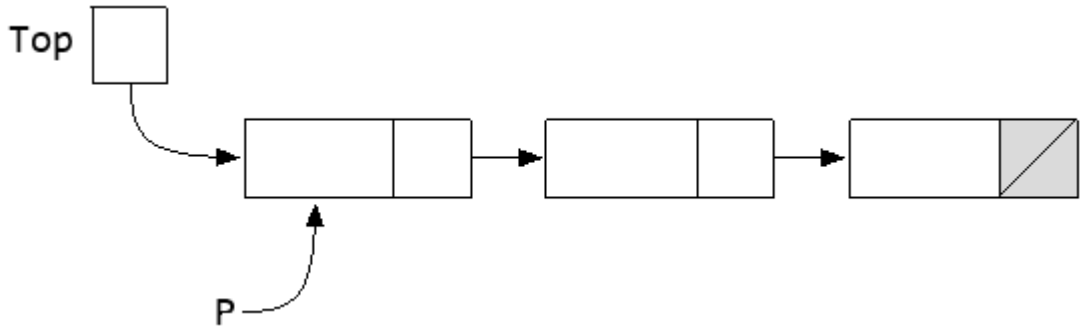
ADT Stack Representasi List

```
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 insertFirs

    /* 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

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

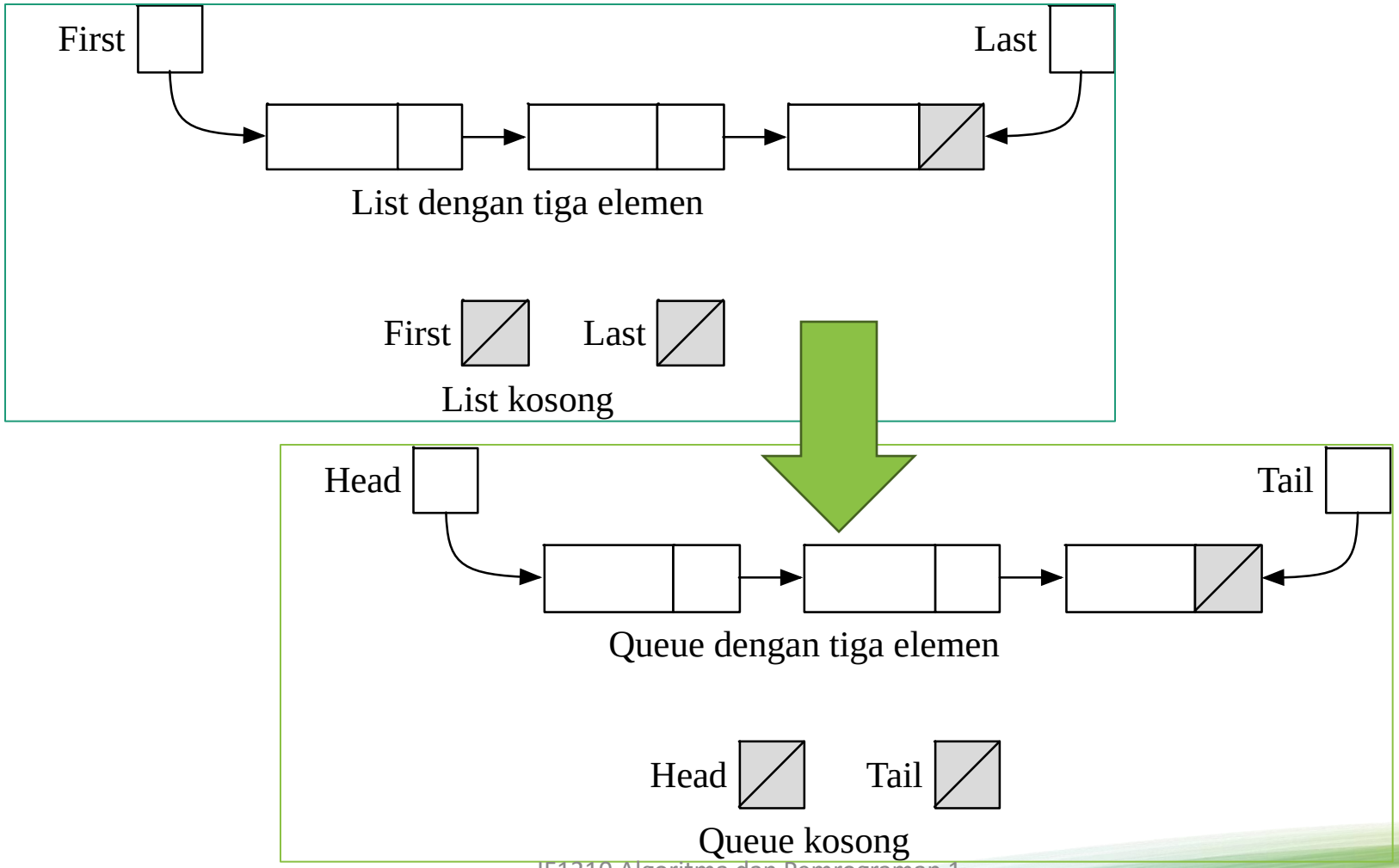
Jika diberikan *Q* adalah Queue dengan elemen *ElmtQ*

CreateQueue: → *Q*	{ Membuat sebuah antrian kosong }
head: *Q* → *ElmtQ*	{ Mengirimkan elemen terdepan Q saat ini }
length: *Q* → *integer*	{ Mengirimkan banyaknya elemen Q saat ini }
enqueue: *ElmtQ* × *Q* → *Q*	{ Menambahkankan sebuah elemen setelah elemen paling belakang Queue }
dequeue: *Q* → *Q* × *ElmtQ*	{ Menghapus kepala Queue, mungkin Q menjadi kosong }
isEmpty: *Q* → *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 ≍ queue



Operasi-operasi dasar pada Queue

- CreateQueue ≍ CreateList
- enqueue ≍ insertLast
- dequeue ≍ deleteFirst
- length ≍ length
- isEmpty ≍ 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

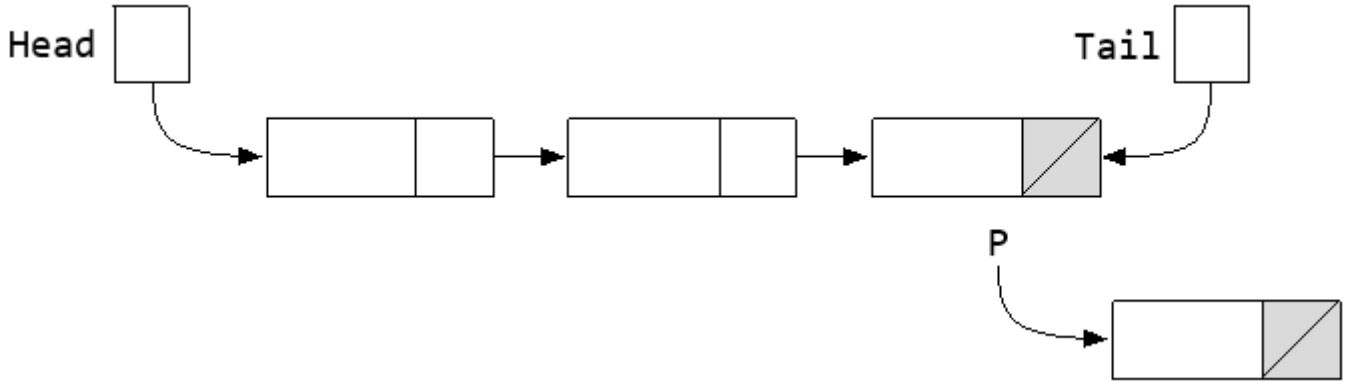
```
/* Selektor */
#define NEXT(p) (p)->next
#define INFO(p) (p)->info

#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
```

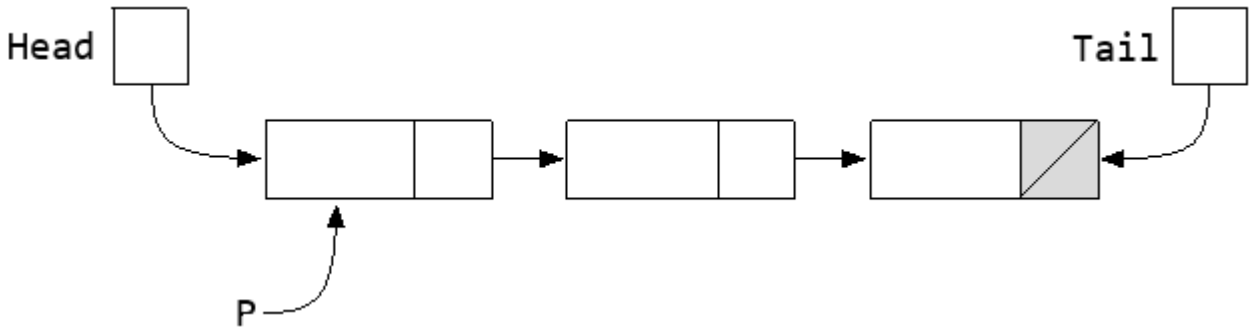
ADT Queue dengan Rep. List

```
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);
}
```



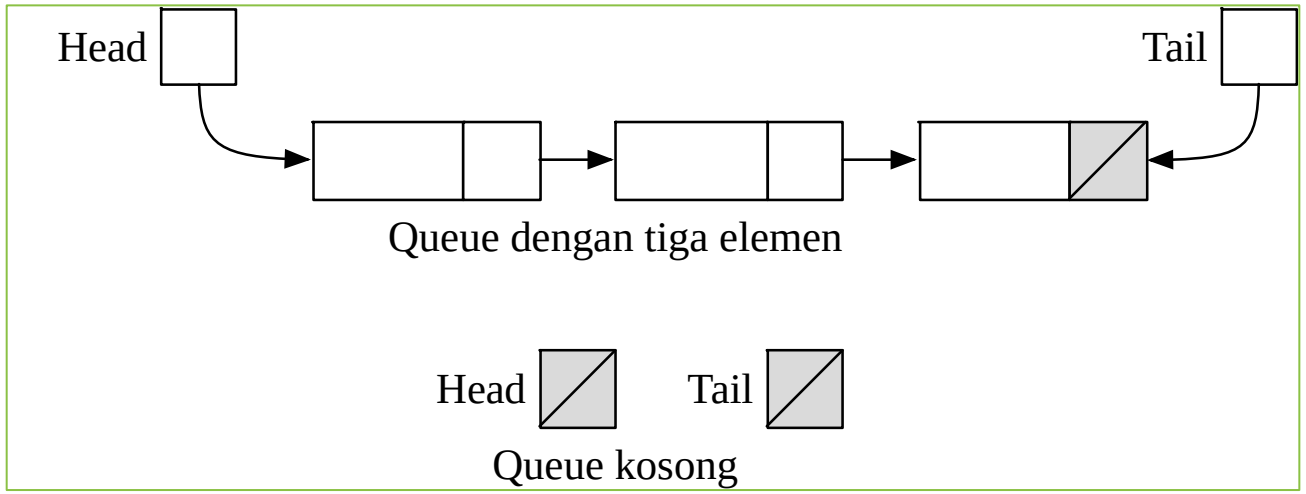
Priority Queue

- 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?

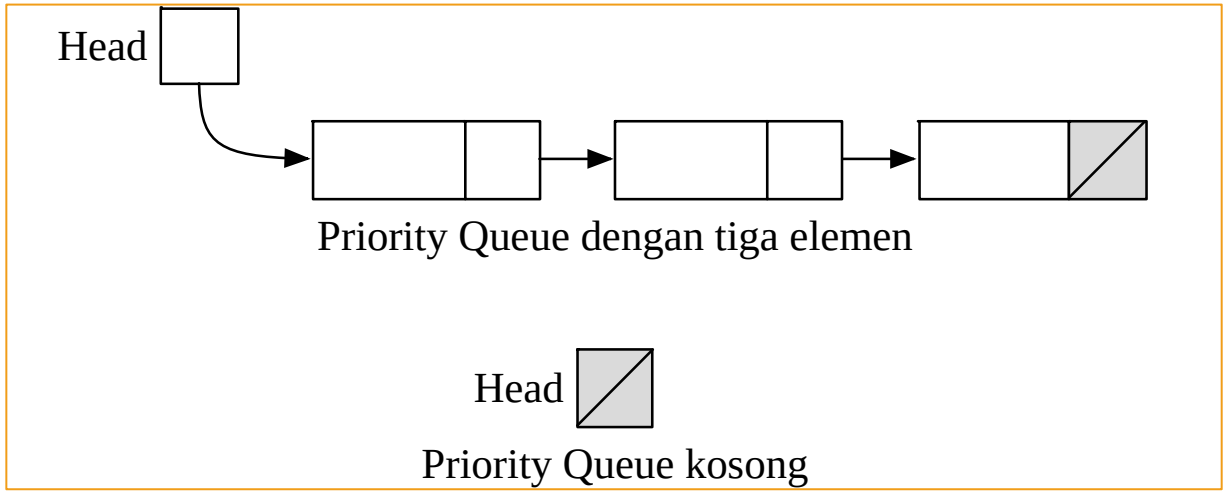
Priority Queue dengan Rep. List

- List Linier yang dicatat first dan last ≍ 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 lojik yang digunakan