Queue (Antrian)

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


He thinks he has to wait in line to get a treat.

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**.

Queue

Elemen Queue tersusun secara **FIFO** (*First In First Out*)

Contoh pemakaian Queue:

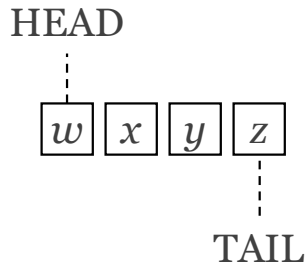
- antrian job yang harus ditangani oleh sistem operasi (*job scheduling*).
- antrian pemrosesan *request* oleh *web server*.
- antrian dalam dunia nyata.

Queue seperti sebuah List dengan batasan lokasi penambahan & penghapusan elemen.

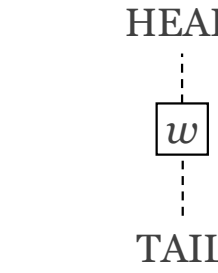
Queue

Secara lojik:

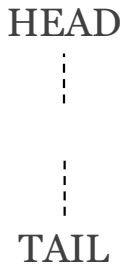
- Elemen
- Head (elemen terdepan)
- Posisi tail (elemen paling belakang)
- Queue kosong



Queue dengan 4 elemen



Queue dengan 1 elemen



Queue kosong

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: $Q \rightarrow$ *integer*	{ Mengirimkan banyaknya elemen Q saat ini }
enqueue: $ElmtQ \times Q \rightarrow Q$	{ Menambahkankan sebuah elemen setelah elemen paling belakang Queue }
dequeue: $Q \rightarrow Q \times ElmtQ$	{ Menghapus kepala Queue, mungkin Q menjadi kosong }
isEmpty: $Q \rightarrow$ *boolean*	{ Tes terhadap Q: true jika Q kosong, false jika Q tidak kosong }

Axiomatic semantics (fungsional)

1) new() returns a queue
2) head(enqueue(*v*, new())) = *v*
3) dequeue(enqueue(*v*, new())) = new()
4) head(enqueue(*v*, enqueue(*w*, *Q*))) = head(enqueue(*w*, *Q*))
5) dequeue(add(*v*, enqueue(*w*, *Q*))) = add(*v*, dequeue(enqueue(*w*, *Q*)))

Di mana *Q* adalah Queue dan *v*, *w* adalah value.

Implementasi Queue dengan array

Memori tempat penyimpan elemen adalah sebuah array dengan indeks 0..CAPACITY-1.

Perlu informasi indeks array yang menyatakan posisi Head dan Tail.

ADT Queue dengan array

```
KAMUS UMUM
constant IDX_UNDEF: integer = -1
constant  CAPACITY: integer = 100

type ElType: integer { elemen Queue }

{ Queue dengan array statik }
type Queue: < buffer: array [0..CAPACITY-1] of ElType, { penyimpanan elemen }
              idxHead: integer, { indeks elemen terdepan }
              idxTail: integer > { indeks elemen terakhir }
```

ADT Queue – Konstruktor, akses, & predikat

```
procedure CreateQueue(output q: Queue)
{ I.S. Sembarang
  F.S. Membuat sebuah Queue q yang kosong berkapasitas CAPACITY
       jadi indeksnya antara 0..CAPACITY-1
       Ciri Queue kosong: idxHead dan idxTail bernilai IDX_UNDEF }

function head(q: Queue) → ElType
{ Prekondisi: q tidak kosong.
  Mengirim elemen terdepan q, yaitu q.buffer[q.idxHead]. }
function length(q: Queue) → integer
{ Mengirim jumlah elemen q saat ini }

function isEmpty(q: Queue) → boolean
{ Mengirim true jika q kosong: lihat definisi di atas }
function isFull(q: Queue) → boolean
{ Mengirim true jika penyimpanan q penuh }
```

ADT Queue - Operasi

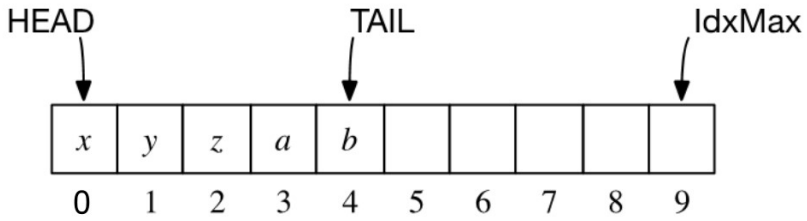
```
procedure enqueue (input/output q: Queue, input val: ElType)
{ Menambahkan val sebagai elemen Queue q.
  I.S. q mungkin kosong, TIDAK penuh
  F.S. q bertambah elemen val sebagai tail yang baru }

procedure dequeue (input/output q: Queue, output val: ElType)
{ Menghapus head dari Queue q.
  I.S. q tidak kosong
  F.S. val berisi nilai head yang lama.
       Jika q tidak menjadi kosong,
         q.idxHead berpindah ke elemen berikutnya pada q.
       Jika q menjadi kosong,
         q.idxHead dan q.idxTail menjadi bernilai IDX_UNDEF. }
```

Implementasi Queue dengan array – alt-1

Jika Queue tidak kosong: idxTail adalah indeks elemen terakhir, idxHead selalu diset = 0.

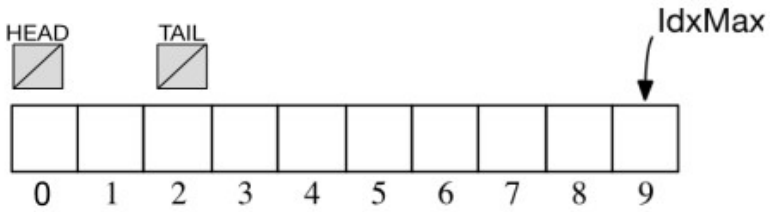
Jika Queue kosong, maka idxHead dan idxTail diset = IDX_UNDEF.

- Ilustrasi Queue tidak kosong dengan 5 elemen:



*dengan IdxMax = CAPACITY-1

- Ilustrasi Queue kosong:



Implementasi Queue dengan array – alt-1

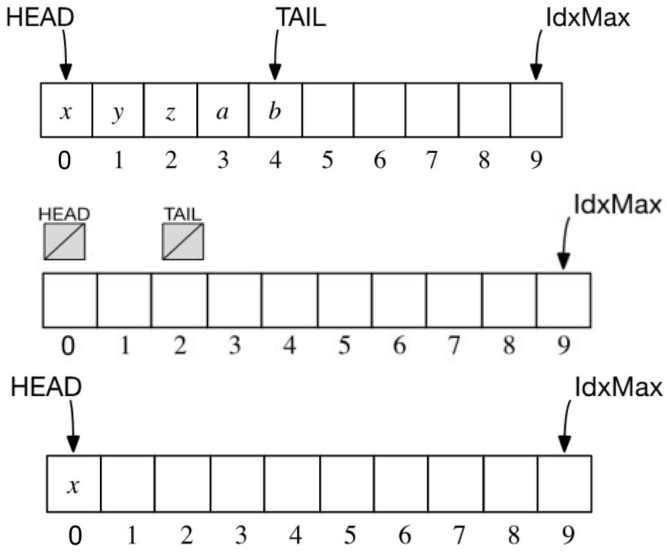
Algoritma **penambahan elemen**:

- **Jika masih ada tempat:** geser TAIL ke kanan.
- **Kasus khusus** (Queue kosong): idxHead dan idxTail diset = 0.

Algoritma paling sederhana dan "naif" untuk **penghapusan elemen**:

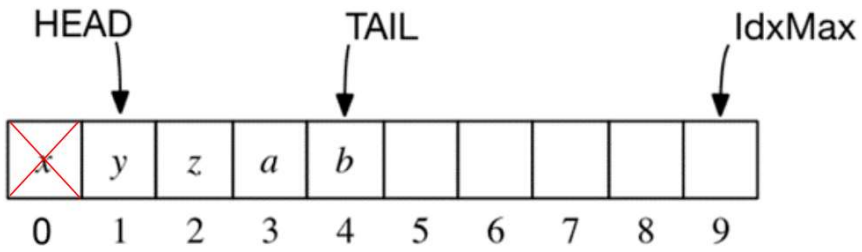
- **Jika Queue tidak kosong:** ambil nilai elemen HEAD, geser semua elemen mulai dari idxHead+1 s.d. idxTail, kemudian geser TAIL ke kiri.
- **Kasus khusus** (Queue berelemen 1): idxHead dan idxTail diset = IDX_UNDEF.

Algoritma ini mencerminkan pergeseran orang yang sedang mengantri di dunia nyata, tapi tidak efisien.

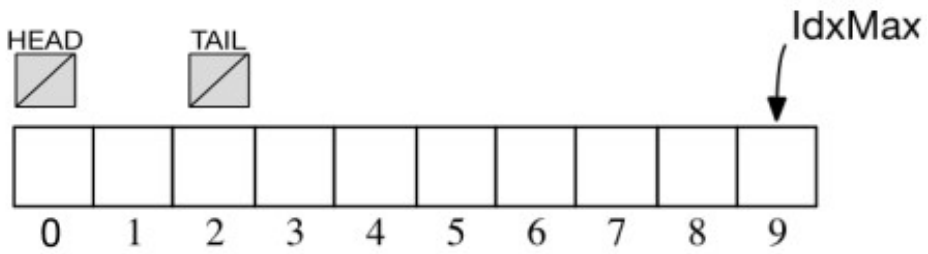


Implementasi Queue dengan array – alt-2

Tabel dengan representasi HEAD dan TAIL yang mana **HEAD bergeser ke kanan** ketika sebuah elemen dihapus.

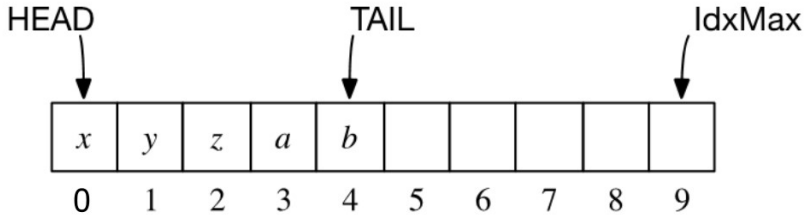


Jika Queue kosong, maka idxHead dan idxTail diset = IDX_UNDEF.

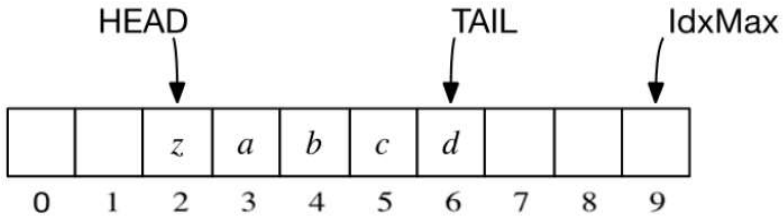


Implementasi Queue dengan array – alt-2

Ilustrasi Queue tidak kosong, dengan 5 elemen, kemungkinan pertama HEAD sedang berada di indeks 0:



Ilustrasi Queue tidak kosong, dengan 5 elemen, kemungkinan lain HEAD tidak berada di indeks 0 (akibat algoritma penghapusan):



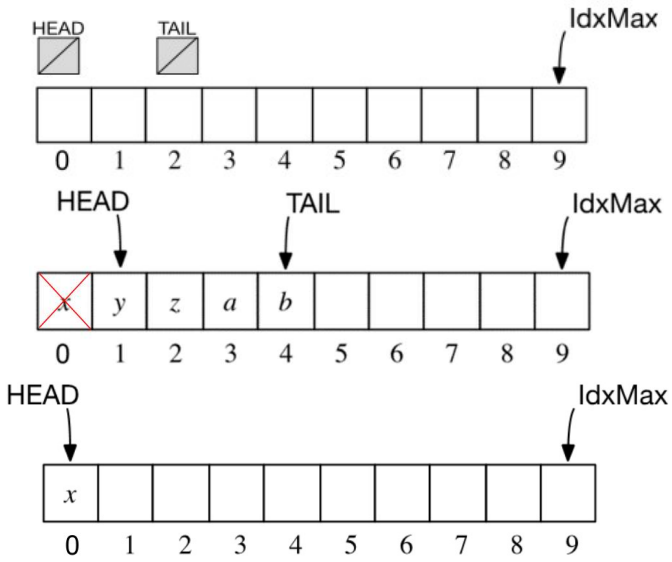
Implementasi Queue dengan array – alt-2

Algoritma **penambahan elemen** sama dengan alt-1, kecuali pada saat "penuh semu" (lihat slide berikutnya.)

Algoritma **penghapusan elemen**:

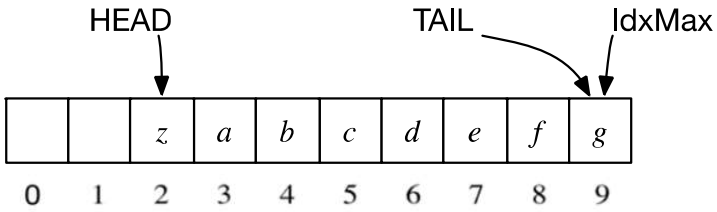
- **Jika Queue tidak kosong:** ambil nilai elemen HEAD, kemudian HEAD digeser ke kanan.
- **Kasus khusus (Queue berelemen 1):** idxHead dan idxTail diset = IDX_UNDEF.

Algoritma ini **TIDAK** mencerminkan pergeseran orang yang sedang mengantri di dunia nyata, tapi **efisien**.



Implementasi Queue dengan array – alt-2

Keadaan Queue penuh tetapi “semu” sebagai berikut:



Harus dilakukan aksi menggeser elemen untuk menciptakan ruangan kosong.

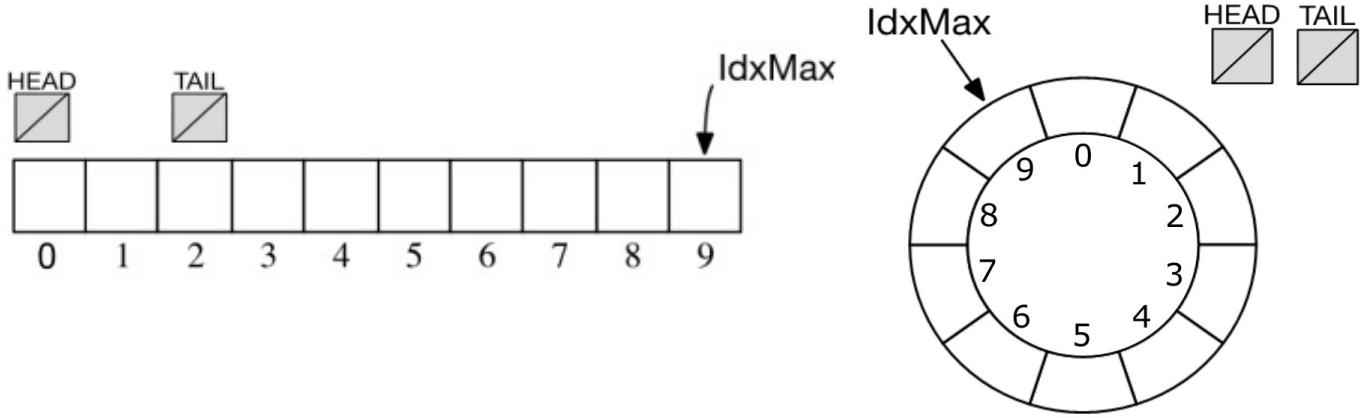
Pergeseran hanya dilakukan jika dan hanya jika idxTail = IdxMax,
i.e., idxTail = CAPACITY-1.

Implementasi Queue dengan array – alt-3

Tabel dengan representasi HEAD dan TAIL yang “berputar” mengelilingi indeks tabel dari awal sampai akhir, kemudian kembali ke awal.

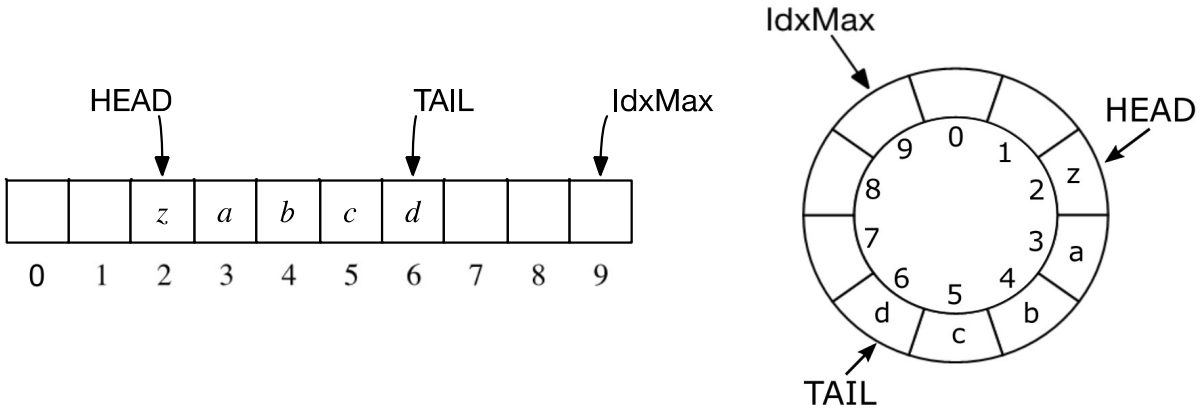
Jika Queue kosong, maka idxHead dan idxTail = IDX_UNDEF.

Representasi ini memungkinkan tidak perlu lagi ada pergeseran yang harus dilakukan seperti pada alt-1 dan alt-2.



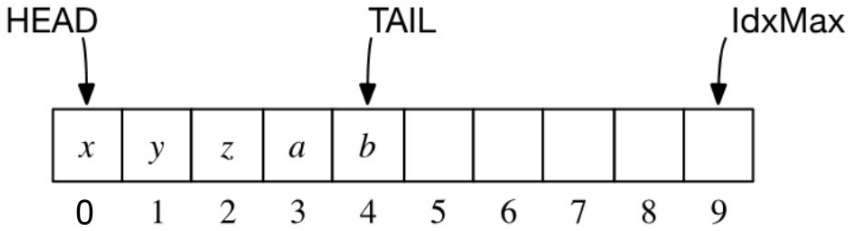
Implementasi Queue dengan array – alt-3

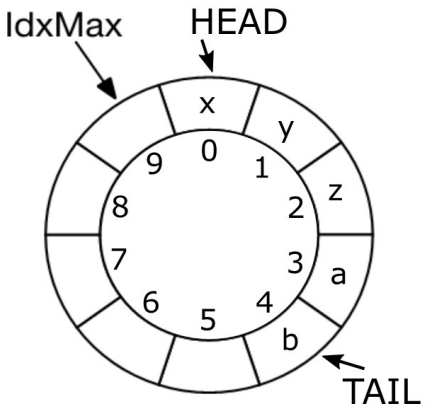
Ilustrasi Queue tidak kosong, dengan 5 elemen, dengan HEAD tidak berada di indeks 0, tetapi **masih "lebih kecil" atau "sebelum" TAIL** (akibat penghapusan/ penambahan):



Implementasi Queue dengan array – alt-3

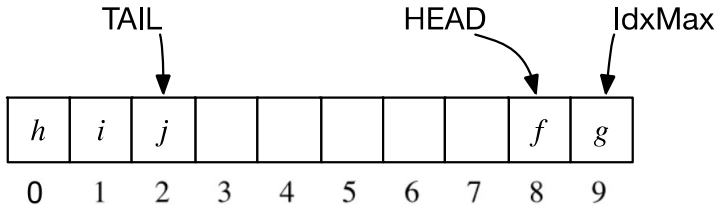
Ilustrasi Queue tidak kosong, dengan 5 elemen, dengan HEAD sedang berada di indeks 0:

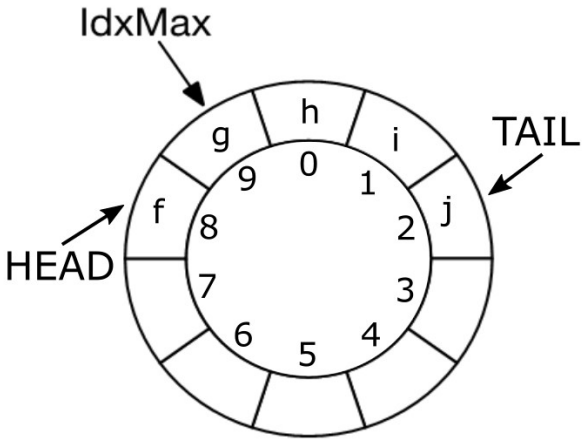




Implementasi Queue dengan array – alt-3

Ilustrasi Queue tidak kosong, dengan 5 elemen, HEAD tidak berada di indeks 0, dan **"lebih besar" atau "sesudah" TAIL** (akibat penghapusan/penambahan):

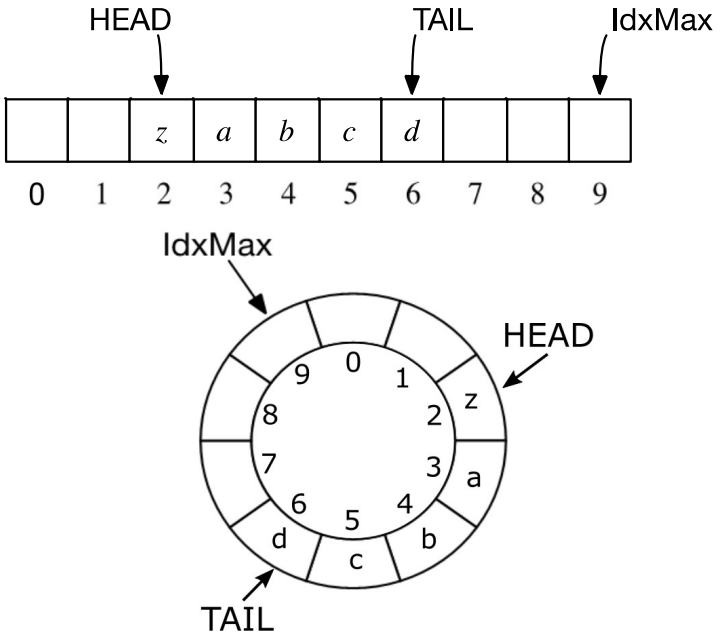




Implementasi Queue dengan array – alt-3

Algoritma **penambahan elemen**:

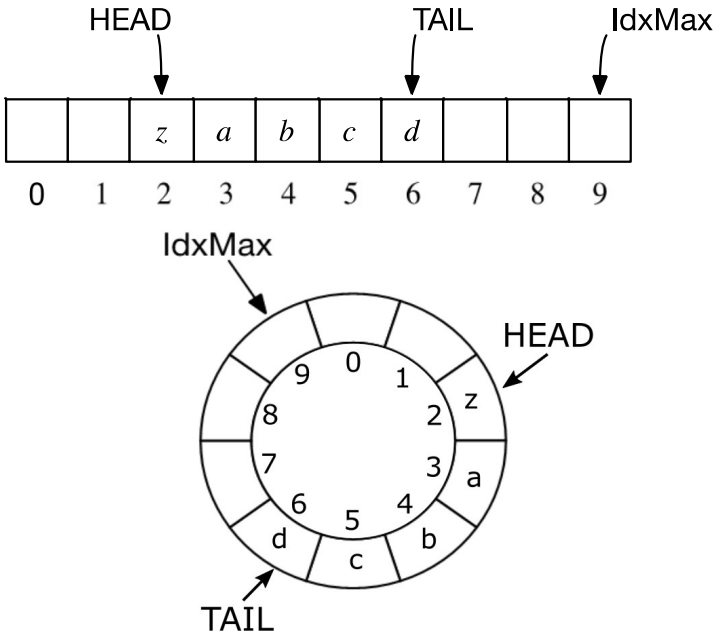
- **Jika masih ada tempat:** geser TAIL
  - **Jika idxTail<IdxMax:** algoritma penambahan elemen sama dengan alt-1 dan alt-2.
  - **Jika idTail=IdxMax:** suksesor dari IdxMax adalah 0 sehingga idxTail yang baru adalah 0.
- **Kasus khusus (Queue kosong):** idxHead dan idxTail diset = 0.



Implementasi Queue dengan array – alt-3

Algoritma **penghapusan elemen**:

- **Jika Queue tidak kosong:**
  - Ambil nilai elemen HEAD, kemudian HEAD digeser ke kanan.
  - **Jika idxHead=IdxMax:** idxHead yang baru adalah 0.
- **Kasus khusus (Queue berelemen 1):** idxHead dan idxTail diset = IDX_UNDEF.



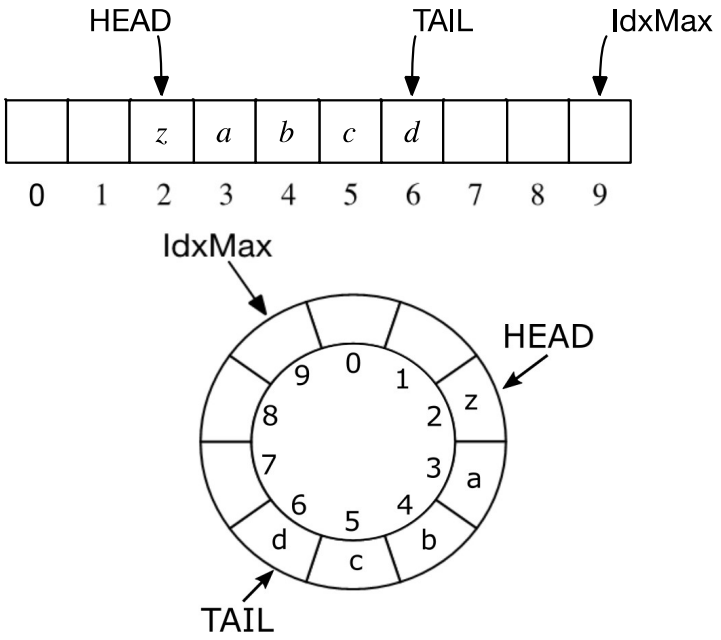
Implementasi Queue dengan array – alt-3

Algoritma ini **efisien** karena tidak perlu pergeseran.

Seringkali strategi pemakaian tabel semacam ini disebut sebagai ***circular buffer***.

Salah satu variasi dari representasi pada alt-3:

Menggantikan representasi TAIL dari "idxTail" menjadi "count" (banyaknya elemen Queue).



Thought exercise

Contoh-contoh sebelumnya menggunakan buffer yang terbatas dan statis.
Di sini “isFull” menjadi relevan meskipun tidak ada di bagian “definisi operasi”.

Renungkan apa yang perlu diubah untuk membuat:

1) ukuran buffer dapat berbeda untuk setiap queue
   (contoh: *q1*, *q2*: Queue; *q1* memiliki kapasitas 100 sedangkan *q2* 150)
2) queue tidak boleh memiliki batas length
   (ukuran buffer bisa ∞ secara teoretis)

Apa konsekuensinya terhadap model alt-1, alt-2, dan alt-3?

Queue dalam Bahasa C

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

ADT Queue dengan C – alt-2 (alokasi memori statik)

```c
/* File: queue.h */
#ifndef QUEUE_H
#define QUEUE_H
#include "boolean.h"
#include <stdlib.h>

#define IDX_UNDEF -1
#define CAPACITY 100

/* Definisi elemen dan address */
typedef int ElType;
/* Contoh struktur type Queue: array statik, indeks head dan indeks tail disimpan */
typedef struct {
  ElType buffer[CAPACITY];
  int idxHead;
  int idxTail;
} Queue;
/* Definisi Queue kosong: idxHead = idxTail = IDX_UNDEF. */
```

NOTE: if you want flexibility, buffer should be an ElType* and you will need malloc() in CreateQueue. You will also have to have a destructor, e.g., DestroyQueue(), where you call free(buffer).

ADT Queue dengan C – alt-2 (alokasi memori statik)

```
/********** AKSES (Selektor) **********/
#define IDX_HEAD(q) (q).idxHead
#define IDX_TAIL(q) (q).idxTail
#define     HEAD(q) (q).buffer[(q).idxHead]
#define     TAIL(q) (q).buffer[(q).idxTail]

/********** Konstruktor **********/
void CreateQueue(Queue *q);
/* I.S. Sembarang
   F.S. Membuat sebuah Queue q yang kosong berkapasitas CAPACITY
        jadi indeksnya antara 0..CAPACITY-1
        Ciri Queue kosong: idxHead dan idxTail bernilai IDX_UNDEF */
```

ADT Queue dengan C – alt-2

```
/********** Operasi: pemeriksaan status Queue **********/
/* catatan: fungsi head(q: Queue) diimplementasikan sebagai macro di halaman
            sebelumnya */

boolean isEmpty(Queue q);
/* Mengirim true jika q kosong: lihat definisi di atas */

boolean isFull(Queue q);
/* Mengirim true jika penyimpanan q penuh */

int length(Queue q);
/* Mengirim jumlah elemen q saat ini */
```

ADT Queue dengan C – alt-2

```
/*** Primitif Add/Delete ***/
void enqueue (Queue *q, ElType val);
/* Proses: Menambahkan val sebagai elemen Queue q.
   I.S. queue mungkin kosong, TIDAK penuh */
   F.S. queue bertambah elemen val sebagai tail yang baru, TAIL bergeser ke kanan */
   Jika IDX_TAIL(queue)=CAPACITY-1, maka geser isi tabel, shg IDX_HEAD(queue)=0 */

void dequeue(Queue *q, ElType* val);
/* Menghapus head dari Queue q.
   I.S. queue tidak kosong
   F.S. val berisi nilai head yang lama.
        Jika queue tidak menjadi kosong,
            queue.idxHead berpindah ke elemen berikutnya pada queue.
        Jika queue menjadi kosong,
            queue.idxHead dan queue.idxTail menjadi bernilai IDX_UNDEF. */

#endif
```

ADT Queue dengan C – alt-2

```c
void CreateQueue(Queue *q) {
/* I.S. ... F.S. ... */
  /* KAMUS LOKAL */
  /* ALGORITMA */
    IDX_HEAD(*q) = IDX_UNDEF;
    IDX_TAIL(*q) = IDX_UNDEF;
}

boolean isEmpty(Queue q) {
/* Mengirim ... */
  /* KAMUS LOKAL */
  /* ALGORITMA */
    return (IDX_HEAD(q) == IDX_UNDEF) && (IDX_TAIL(q) == IDX_UNDEF);
}
```

ADT Queue dengan C – alt-2

```
boolean isFull(Queue q) {
/* Mengirim ... */
  /* KAMUS LOKAL */
  /* ALGORITMA */
    return (IDX_HEAD(q) == 0) && (IDX_TAIL(q) == CAPACITY-1);
}

int length(Queue q) {
/* Mengirim ... */
  /* KAMUS LOKAL */
  /* ALGORITMA */
    if (IDX_HEAD(q)==IDX_UNDEF)
        return 0;
    else
        return (IDX_TAIL(q) - IDX_HEAD(q)) + 1;
}
```

ADT Queue dengan C – alt-2

```c
void enqueue(Queue *q, ElType val) {
/* I.S. ... F.S. ... */
  /* KAMUS LOKAL */
  /* ALGORITMA */
    if (isEmpty(*q)) {
      IDX_HEAD(*q) = 0;
      IDX_TAIL(*q) = 0;
    } else { // *q is not empty
      if (IDX_TAIL(*q)==(CAPACITY-1)) { // elemen mentok kanan, geser dulu
        for (int i=IDX_HEAD(*q); i<=IDX_TAIL(*q); i++) {
          (*q).buffer[i-IDX_HEAD(*q)] = (*q).buffer[i];
        }
        IDX_TAIL(*q) -= IDX_HEAD(*q);
        IDX_HEAD(*q) = 0;
      }
      IDX_TAIL(*q)++;
    }
    TAIL(*q) = val;
}
```

ADT Queue dengan C – alt-2

```c
void dequeue(Queue *q, ElType *val) {
/* I.S. ... F.S. ... */
  /* KAMUS LOKAL */
  /* ALGORITMA */
    *val = HEAD(*q);
    if (IDX_HEAD(*q) == IDX_TAIL(*q)) {
      IDX_HEAD(*q) = IDX_UNDEF;
      IDX_TAIL(*q) = IDX_UNDEF;
    } else {
      IDX_HEAD(*q)++;
    }
}
```

Latihan Soal: ADT Queue

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

Soal 1 – Circular Buffer

a. Definisikan struktur data yang merepresentasi Queue bertipe ElType yang terdiri atas `<id: integer, cost: integer>` dalam bentuk *circular buffer*, dengan alokasi statik maksimum 100 elemen, dan menyimpan informasi indeks head dan count (banyaknya elemen dalam Queue)

b. Buatlah function `isFull`

c. Buatlah procedure `enqueue`

d. Buatlah procedure `dequeue`

Soal 1

```
function isFull (q: Queue) → boolean
{mengirim true jika q penuh}

procedure enqueue (input/output q: Queue, input val: ElType)
{Proses: menambahkan val pada q sebagai Tail baru}
{IS: q mungkin kosong, q tidak penuh}
{FS: val menjadi Tail baru dengan mekanisme circular buffer}

procedure dequeue (input/output q: Queue, output val: ElType)
{Proses: menyimpan nilai Head q ke val dan menghapus Head q}
{IS: q tidak kosong}
{FS: val adalah nilai elemen Head, Head "bergerak" dengan mekanisme
     circular buffer. q mungkin kosong}
```

Soal 2 – Round Robin

Pandanglah Queue pada soal nomor 1 sebagai antrian pekerjaan dengan id adalah nomor identifikasi pekerjaan dan “cost” adalah cost waktu penyelesaian pekerjaan (*time cost*).

Dengan memanfaatkan queue pada soal nomor 1, buatlah procedure **roundRobin** yang memproses Queue secara Round Robin, yaitu memroses dengan waktu terbatas T:

- Jika elemen pada HEAD memiliki cost ≤ T, elemen tersebut dihapus dari Queue.
- Jika elemen pada HEAD memiliki cost > T, maka elemen tersebut dihapus dari Queue **dan** disisipkan kembali sebagai Tail dengan cost yang berkurang sebesar T.

Soal 2 – Round Robin

```
procedure roundRobin (input/output q: Queue, input t: integer)
{Proses: memproses elemen antrian q secara round robin}
{IS: q tidak kosong, t adalah waktu yang tersedia untuk memproses setiap elemen}
{FS: elemen e pada posisi HEAD dihapus dari q.
     Jika cost e ≤ t maka ditampilkan “<id> telah selesai diproses”.
     Jika cost e > t maka e disisipkan kembali sebagai tail q
     dengan cost berkurang sebesar t }
```

Soal 3 – Priority Queue

a. Dengan memodifikasi Queue alternatif 2 pada slide materi kuliah, definisikan (algoritmik) struktur data yang merepresentasi Queue yang menggambarkan antrian pekerjaan (job shop). Setiap elemen Queue bertipe ElType yang terdiri atas ⟨`id: integer, cost: integer`⟩. `id` menunjukkan nomor identifikasi unik dari pekerjaan yang dikelola Queue, dan elemen Queue terurut membesar berdasarkan "`cost`" waktu memproses pekerjaan.

b. Buatlah prosedur `enqueue`

c. Buatlah prosedur `dequeue`

Soal 3 – Priority Queue

```
procedure enqueue (input/output q: Queue, input val: ElType)
{Proses: menambahkan val sebagai elemen baru di q, dengan
         memperhatikan lamanya waktu pekerjaan tsb dapat
         diselesaikan, yaitu pekerjaan yang lebih cost diletakkan
         lebih akhir. Jika ada 2 pekerjaan yang cost waktunya sama,
         pekerjaan terakhir yang baru datang disisipkan lebih
         belakang}
{IS: q mungkin kosong, q tidak penuh}
{FS: val menjadi elemen q yang baru dengan urutan waktu pekerjaan
     membesar}

procedure dequeue (input/output q: Queue, output val: ElType)
{Proses: menyimpan IDX_UNDEFa head q pada val dan menghapus head dari q}
{IS: q tidak kosong}
{FS: elemen pada HEAD dihapus, dan disimpan nilainya pada val}
```