QUEUE
Antrian

1. DEFINISI

QUEUE (Antrian) adalah list linier yang :

1. dikenali elemen pertama (HEAD) dan elemen terakhirnya (TAIL),
2. aturan penyisipan dan penghapusan elemennya didefinisikan sebagai berikut:
   - Penyisipan selalu dilakukan setelah elemen terakhir
   - Penghapusan selalu dilakukan pada elemen pertama
3. satu elemen dengan yang lain dapat diakses melalui informasi NEXT

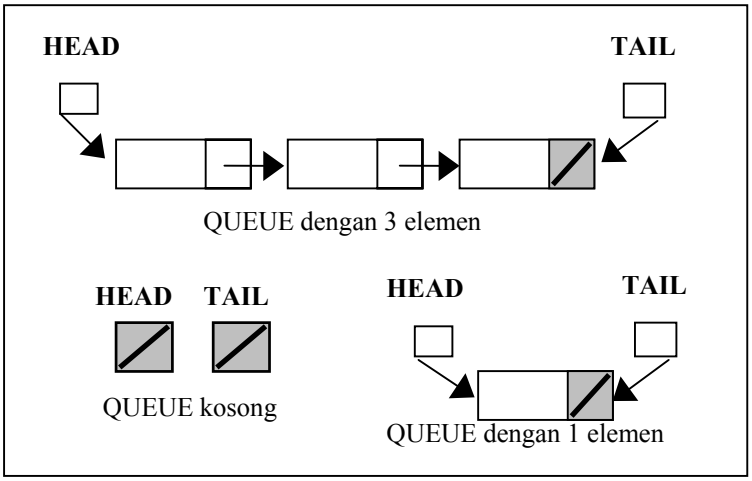
Struktur data ini banyak dipakai dalam informatika, misalnya untuk merepresentasi :
- antrian job yang harus ditangani oleh sistem operasi
- antrian dalam dunia nyata.

Maka secara lojik, sebuah QUEUE dapat digambarkan sebagai list linier yang setiap elemennya adalah

type ElmtQ : < Info : InfoType, Next :address>

dengan InfoType adalah sebuah type terdefinisi yang menentukan informasi yang disimpan pada setiap elemen queue, dan address adalah "alamat" dari elemen
Selain itu alamat elemen pertama (**HEAD**) dan elemen terakhir(**TAIL**) dicatat :
Maka jika Q adalah Queue dan P adalah adaress, penulisan untuk Queue adalah :

Head(Q),Tail(Q), Info(Head(Q)), Info(Tail(Q))



DEFINISI FUNGSIONAL QUEUE

Diberikan Q adalah QUEUE dengan elemen ElmtQ maka definisi fungsional antrian adalah:

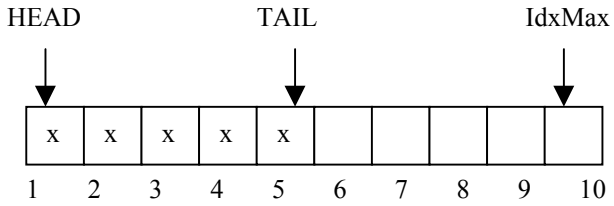
IsEmpty: Q → boolean { Test terhadap Q:
true jika Q kosong, false jika Q tidak kosong}
IsFull : Q → boolean { Test terhadap Q:
true jika memori Q sudah penuh, false jika memori Q tidak penuh}
NBElmt(Q) : Q → integer { Mengirimkan banyaknya elemen Q }

CreateEmpty : → Q {Membuat sebuah antrian kosong }
Add :ElmtQ x Q → Q { Menambahkan sebuah elemen setelah
elemen ekor QUEUE }
Del : Q →Q x ElmtQ { Menghapus kepala QUEUE, }
{ mungkin Queue menjadi kosong}

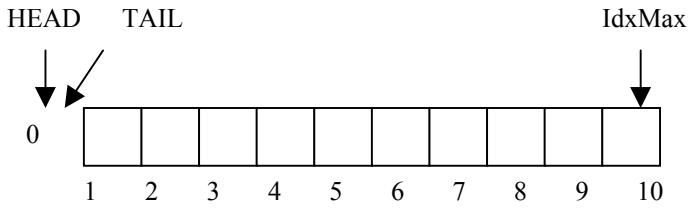
Implementasi QUEUE dengan tabel :
Memori tempat penyimpan elemen adalah sebuah tabel dengan indeks 1.. IdxMax. IdxMax dapat juga "dipetakan" ke kapasitas Queue. Representasi field Next: Jika i adalah "address" sebuah elemen, maka suksesor i adalah Next dari elemen Queue.

Alternatif I :
Tabel dengan hanya representasi TAIL adalah indeks elemen terakhir, HEAD selalu diset sama dengan 1 jika Queue tidak kosong. Jika Queue kosong, maka HEAD=0.

Ilustrasi Queue tidak kosong, dengan 5 elemen :



Ilustrasi Queue kosong:

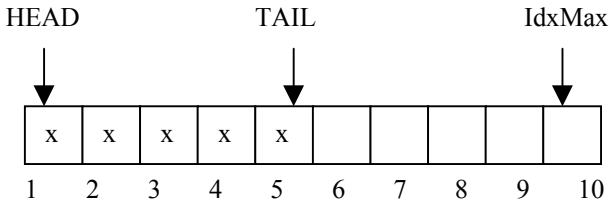


Algoritma paling sederhana untuk **penambahan elemen** jika masih ada tempat adalah dengan "memajukan" TAIL. Kasus khusus untuk Queue kosong karena HEAD harus diset nilainya menjadi 1.

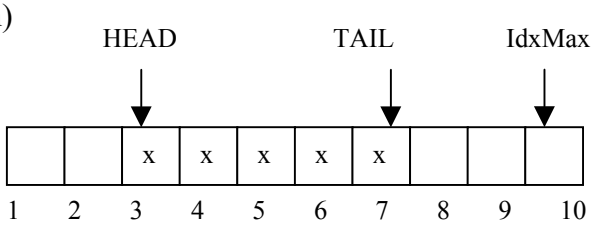
Algoritma paling sederhana dan "naif" untuk **penghapusan elemen** jika Queue tidak kosong: ambil nilai elemen HEAD, geser semua elemen mulai dari HEAD+1 s/d TAIL (jika ada), kemudian TAIL "mundur". Kasus khusus untuk Queue dengan keadaan awal berelemen 1, yaitu menyesuaikan HEAD dan TAIL dengan

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

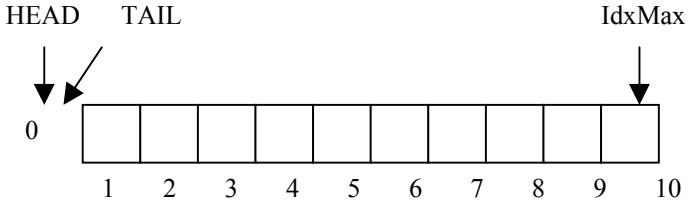
Alternatif II :
Tabel dengan representasi HEAD dan TAIL, HEAD "bergerak" ketika sebuah elemen dihapus jika Queue tidak kosong. Jika Queue kosong, maka HEAD=0. Ilustrasi Queue tidak kosong, dengan 5 elemen, kemungkinan pertama HEAD "sedang berada di posisi awal:



Ilustrasi Queue tidak kosong, dengan 5 elemen, kemungkinan pertama HEAD tidak berada di posisi awal. Hal ini terjadi akibat algoritma penghapusan yang dilakukan (lihat keterangan)

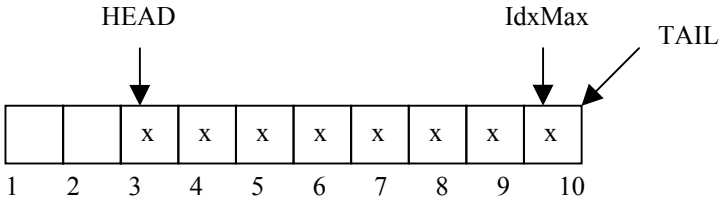


Ilustrasi Queue kosong:



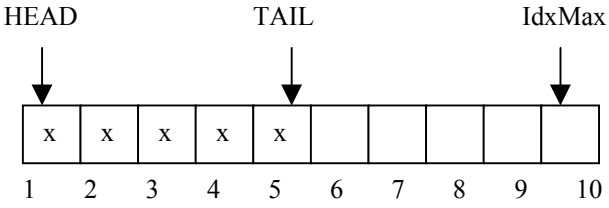
Algoritma untuk **penambahan elemen** sama dengan alternatif I: jika masih ada tempat adalah dengan "memajukan" TAIL. Kasus khusus untuk Queue kosong karena HEAD harus diset nilainya menjadi 1. Algoritmanya sama dengan alternatif I

Algoritma untuk **penghapusan elemen** jika Queue tidak kosong: ambil nilai elemen HEAD, kemudian HEAD "maju". Kasus khusus untuk Queue dengan keadaan awal berelemen 1, yaitu menyesuaikan HEAD dan TAIL dengan DEFINISI. Algoritma ini **TIDAK** mencerminkan pergeseran orang yang sedang mengantri di dunia nyata, tapi **efisien**. Namun suatu saat terjadi keadaan Queue penuh tetapi "semu sebagai berikut :

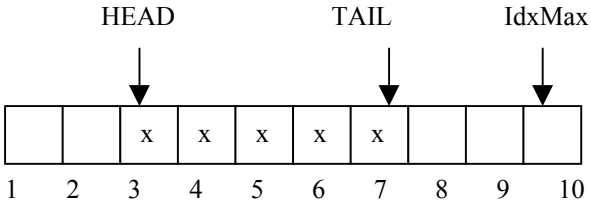


Jika keadaan ini terjadi, haruslah dilakukan aksi menggeser elemen untuk menciptakan ruangan kosong. Pergeseran hanya dilakukan jika dan hanya jika TAIL sudah mencapai IndexMax.

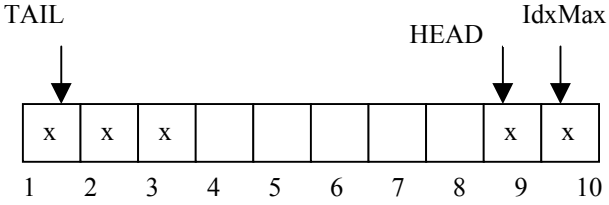
Alternatif III :
Tabel dengan representasi HEAD dan TAIL yang "berputar" mengelilingi indeks tabel dari awal sampai akhir, kemudian kembali ke awal. Jika Queue kosong, maka HEAD=0. Representasi ini memungkinkan tidak perlu lagi ada pergeseran yang harus dilakukan seperti pada alternatif II pada saat penambahan elemen.
Ilustrasi Queue tidak kosong, dengan 5 elemen, dengan HEAD "sedang" berada di posisi awal:



Ilustrasi Queue tidak kosong, dengan 5 elemen, dengan HEAD tidak berada di posisi awal, tetapi masih "lebih kecil" atau "sebelum" TAIL. Hal ini terjadi akibat algoritma penghapusan/Penambahan yang dilakukan (lihat keterangan).



Ilustrasi Queue tidak kosong, dengan 5 elemen, HEAD tidak berada di posisi awal, tetapi "lebih besar" atau "sesudah" TAIL. Hal ini terjadi akibat algoritma penghapusan/Penambahan yang dilakukan (lihat keterangan)



Algoritma untuk **penambahan elemen**: jika masih ada tempat adalah dengan "memajukan" TAIL. Tapi jika TAIL sudah mencapai IdxMax, maka suksesor dari IdxMax adalah 1 sehingga TAIL yang baru adalah 1. Jika TAIL belum mencapai IdxMax, maka algoritma penambahan elemen sama dengan alternatif II. Kasus khusus untuk Queue kosong karena HEAD harus diset nilainya menjadi 1.

Algoritma untuk **penghapusan elemen** jika Queue tidak kosong: ambil nilai elemen HEAD, kemudian HEAD "maju". Penentuan suatu suksesor dari indkes yang diubah/"maju" dibuat Seperti pada algoritma penambahan elemen: jika HEAD mencapai IdxMAx, maka suksesor dari HEAD adalah 1. Kasus khusus untuk Queue dengan keadaan awal berelemen 1, yaitu menyesuaikan HEAD dan TAIL dengan DEFINISI. Algoritma ini **efisien** karena tidak perlu pergeseran, dan seringkali strategi pemakaian tabel semacam ini disebut sebagai "circular buffer", dimana tabel penyimpan elemen dianggap sebagai "buffer".
Salah satu variasi dari representasi pada alternatif III adalah : menggantikan representasi TAIL dengan COUNT, yaitu banyaknya elemen Queue. Dengan

representasi ini, banyaknya elemen diketahui secara eksplisit, tetapi untuk melakukan penambahan elemen harus dilakukan kalkulasi TAIL. Buatlah sebagai latihan.

ADT Queue dalam Bahasa C, REPRESENTASI DENGAN TABEL

```
/* File : queue.h */
/* deklarasi Queue yang diimplementasi dengan tabel kontigu */
/* HEAD dan TAIL adalah alamat elemen pertama dan terakhir  */
/* Queue mampu menampung MaxEl buah elemen */
#ifndef queue_H
#define queue_H
#include "boolean.h"
#include <stdlib.h>
#define Nil 0
/* Definisi elemen dan address */
typedef int infotype;
typedef int address;   /* indeks tabel */
/* Contoh deklarasi variabel bertype Queue : */
/*Versi I: tabel dinamik,Head dan Tail eksplisit, ukuran disimpan */
typedef struct { infotype * T; /* tabel penyimpan elemen */
                   address HEAD;  /* alamat penghapusan */
                   address TAIL;  /* alamat penambahan */
                   int MaxEl;      /* MAx elemen queue */
                  } Queue;
/*Definisi Queue kosong: Head=Nil; TAIL=Nil.*/
/* Catatan implementasi: T[0] tidak pernah dipakai */
/**** AKSES (Selektor)  *****/
/* Jika Q adalah Queue, maka akses elemen : */
#define Head(Q) (Q).HEAD
#define Tail(Q) (Q).TAIL
#define InfoHead(Q) (Q).T[(Q).HEAD]
#define InfoTail(Q) (Q).T[(Q).TAIL]
#define MaxEl(Q) (Q).MaxEl
/* Prototype */
boolean IsEmpty (Queue Q);
/* Mengirim true jika Q kosong: lihat definisi di atas */
boolean IsFull(Queue Q);
/*Mengirim true jika tabel penampung elemen Q sudah penuh */
/* yaitu mengandung MaxEl elemen
int NBElmt(Queue Q);
/* Mengirimkan banyaknya elemen queue. Mengirimkan 0 jika Q kosong */
/**** Kreator ***/
void CreateEmpty(Queue *Q, int Max);
/* I.S. sembarang */
/* F.S. Sebuah Q kosong terbentuk dan salah satu kondisi sbb: */
/* Jika alokasi berhasil, Tabel memori dialokasi berukuran Max */
/* atau : jika alokasi gagal, Q kosong dg Maksimum elemen=0 */
/* Proses : Melakukan alokasi,Membuat sebuah Q kosong */
/**** Destruktor ***/
void DeAlokasi(Queue *Q);
/* Proses: Mengembalikan memori Q */
/* I.S. Q pernah dialokasi */
/* F.S. Q menjadi tidak terdefinisi lagi, MaxEl(Q) diset 0 */
/**** Primitif Add/Delete ****/
void Add (Queue * Q, infotype X);
/* Proses: Menambahkan  X pada Q dengan aturan FIFO */
/* I.S. Q mungkin kosong, tabel penampung elemen Q TIDAK penuh */
/* F.S. X menjadi TAIL yang baru, TAIL "maju" */
/* Jika Tail(Q)=MaxEl+1 maka geser isi tabel, shg Head(Q)=1 */
void Del(Queue * Q, infotype* X);
/* Proses: Menghapus  X pada Q dengan aturan FIFO */
/* I.S. Q tidak mungkin kosong */
/* F.S. X = nilai elemen HEAD pd I.S.,HEAD "maju";Q mungkin kosong */
#endif
```

```
/* File :queues.h */
/* deklarasi Queue yang diimplementasi dengan tabel kontigu */
/* HEAD dan TAIL adalah alamat elemen pertama dan terakhir  */
/* Kapasitas Queue=MaxEl buah elemen, dan indeks dibuat "sirkuler" */
#ifndef queues_H
#define queues_H
#include "boolean.h"
#include <stdlib.h>
#define Nil 0
/* Definisi elemen dan address */
typedef int infotype;
typedef int address;   /* indeks tabel */
/* Contoh deklarasi variabel bertype Queue : */
/*Versi I: tabel dinamik,Head dan Tail eksplisit, ukuran disimpan */
typedef struct { infotype * T; /* tabel penyimpan elemen */
                   address HEAD;  /* alamat penghapusan */
                   address TAIL;  /* alamat penambahan */
                   int MaxEl;      /* MAx elemen queue */
                  } Queue;
/*Definisi Queue kosong: Head=Nil; TAIL=Nil.*/
/* Catatan implementasi: T[0] tidak pernah dipakai */
/**** AKSES (Selektor)  *****/
/* Jika Q adalah Queue, maka akses elemen : */
#define Head(Q) (Q).HEAD
#define Tail(Q) (Q).TAIL
#define InfoHead(Q) (Q).T[(Q).HEAD]
#define InfoTail(Q) (Q).T[(Q).TAIL]
#define MaxEl(Q) (Q).MaxEl
/* Prototype */
boolean IsEmpty (Queue Q);
/* Mengirim true jika Q kosong: lihat definisi di atas */
boolean IsFull(Queue Q);
/*Mengirim true jika tabel penampung elemen Q sudah penuh */
/* yaitu mengandung MaxEl elemen */
int NBElmt(Queue Q);
/* Mengirimkan banyaknya elemen queue. Mengirimkan 0 jika Q kosong */
/**** Kreator ***/
void CreateEmpty(Queue *Q, int Max);
/* I.S. sembarang */
/* F.S. Sebuah Q kosong terbentuk dan salah satu kondisi sbb: */
/* Jika alokasi berhasil, Tabel memori dialokasi berukuran Max */
/* atau : jika alokasi gagal, Q kosong dg Maksimum elemen=0 */
/* Proses : Melakukan alokasi,Membuat sebuah Q kosong */
/**** Destruktor ***/
void DeAlokasi(Queue *Q);
/* Proses: Mengembalikan memori Q */
/* I.S. Q pernah dialokasi */
/* F.S. Q menjadi tidak terdefinisi lagi, MaxEl(Q) diset 0 */
/**** Primitif Add/Delete ****/
void Add (Queue * Q, infotype X);
/* Proses: Menambahkan  X pada Q dengan aturan FIFO */
/* I.S. Q mungkin kosong, tabel penampung elemen Q TIDAK penuh */
/* F.S. X menjadi TAIL yang baru, TAIL "maju" */
/* Jika Tail(Q)=MaxEl+1 maka Tail(Q) diset =1 */
void Del(Queue * Q, infotype* X);
/* Proses: Menghapus  X pada Q dengan aturan FIFO */
/* I.S. Q tidak mungkin kosong */
/* F.S. X = nilai elemen HEAD pd I.S.,Jika Head(Q)=MaxEl+1, */
/* Head(Q) diset=1; Q mungkin kosong */
#endif
```

QUEUE Dengan Representasi Berkait