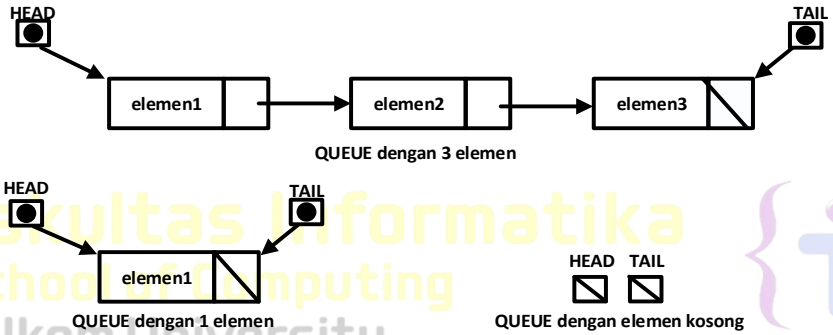
Modul 8 QUEUE

8.1 *Pengertian Queue*

Queue (dibaca : kyu) merupakan struktur data yang dapat diumpamakan seperti sebuah antrian. Misalkan antrian pada loket pembelian tiket Kereta Api. Orang yang akan mendapatkan pelayanan yang pertama adalah orang pertamakali masuk dalam antrian tersebut dan yang terakhir masuk dia akan mendapatkan layanan yang terakhir pula. Jadi prinsip dasar dalam *Queue* adalah **FIFO** (*First in Fisrt out*), proses yang pertama masuk akan diakses terlebih dahulu. Dalam pengimplementasian struktur *Queue* dalam C dapat menggunakan tipe data *array* dan *linked list*.

Dalam praktikum ini hanya akan dibahas pengimplementasian *Queue* dalam bentuk *linked list*. Implementasi *Queue* dalam *linked list* sebenarnya tidak jauh berbeda dengan operasi *list* biasa, malahan lebih sederhana. Karena sesuai dengan sifat FIFO dimana proses *delete* hanya dilakukan pada bagian ***Head*** (depan *list*) dan proses *insert* selalu dilakukan pada bagian ***Tail*** (belakang *list*) atau sebaliknya, tergantung dari persepsi masing-masing. Dalam penerapannya *Queue* dapat diterapkan dalam *single linked list* dan *double linked list*.



Gambar 8-1 *Queue*

Contoh pendeklarasian struktur data *queue*:

```
1   #ifndef queue_H
2   #define queue_H
3   #define Nil NULL
4   #define info(P) (P)->info
5   #define next(P) (P)->next
6   #define head(Q) ((Q).head)
7   #define tail(Q) ((Q).tail)
8
9   typedef int infotype; /*tipe data dalam queue */
10  typedef struct elmQueue *address; /* tipe data pointer untuk elemen queue */
11  struct elmQueue{
12      infotype info;
13      address next;
14  }; /*tipe data elemen queue */
15  /* pendeklarasian tipe data queue */
16  struct queue {
17      address head, tail;
18  };
19  #endif
```

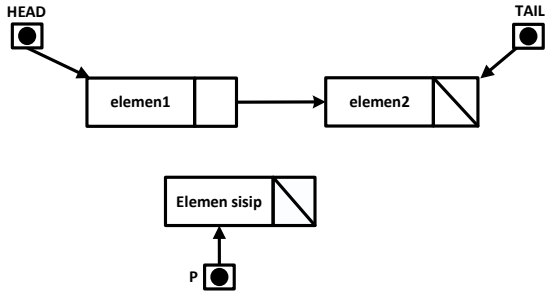
8.2 *Operasi-Operasi dalam Queue*

Dalam *queue* ada dua operasi utama, yaitu operasi penyisipan (*Insert*/*Enqueue*) dan operasi pengambilan (*Delete*/*Dequeue*).

8.2.1 Insert (Enqueue)

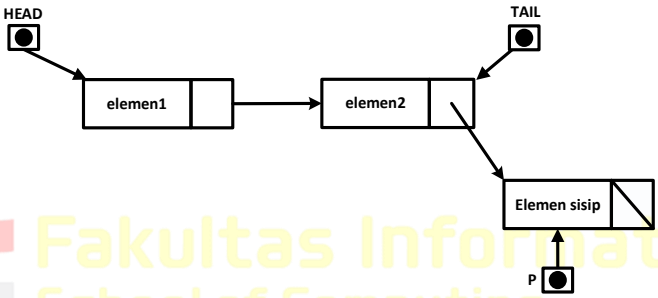
Operasi penyisipan selalu dilakukan pada akhir (*tail*).

Langkah – langkah dalam proses Enqueue:



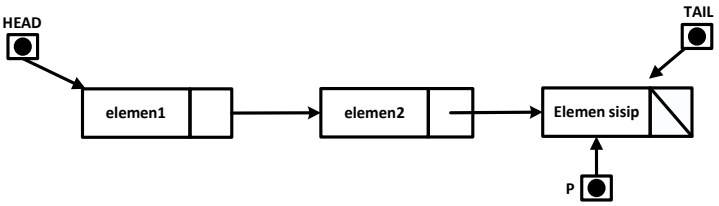

```
P = alokasi(X);
next(P) = Nil;
```

Gambar 8-2 *Queue Insert* 1




```
next(tail(Q)) = P;
```

Gambar 8-3 *Queue Insert* 2



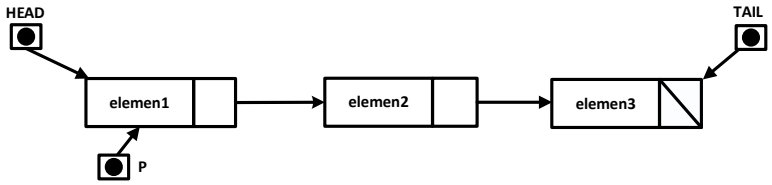

```
tail(Q) = P;
```

Gambar 8-4 *Queue Insert* 3

```
1   /* buat dahulu elemen yang akan disisipkan */
2   address createElm(int x){
3       address p = alokasi(x);
4       next(p) = null;
5       return p;
6   }
7
8   /* contoh sintak queue insert */
9   void queue(address p){
10      next(tail(Q)) = p;
11      tail(Q) = p;
12  }
```

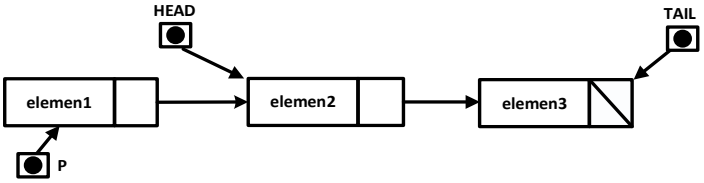
8.2.2 Delete (Dequeue)

Operasi *delete* dilakukan pada awal (*head*).



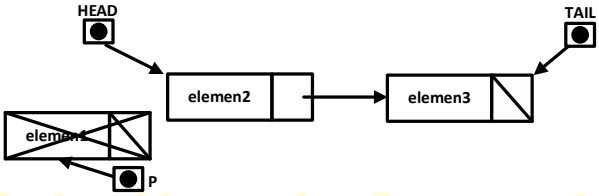

```
P = head(Q);
```

Gambar 8-5 *Queue Delete* 1




```
head(Q) = next(P);
```

Gambar 8-6 *Queue Delete* 2




```
next(P) = Nil;
return P;
```

Gambar 8-7 *Queue Delete* 3


```
1  /*contoh sintak dequeue */
2  address dequeue(address p) {
3      p = head(Q);
4      head(Q) = next(p);
5      next(p) = null;
6      return p;
7  }
```

8.3 *Primitif-Primitif dalam Queue*

Primitif-primitif pada *queue* tersimpan pada ADT *queue*, seperti pada materi sebelumnya, primitif-primitifnya tersimpan pada *file* *.h dan *.c.

File *.h untuk ADT *queue*:

```
1  /*file : queue .h
2   contoh ADT queue  dengan representasi fisik pointer
3   representasi address dengan pointer
4   info tipe adalah integer */
5
6  #ifndef QUEUE_H_INCLUDE
7  #define QUEUE_H_INCLUDE
8  #include <stdio.h>
9
10 #define Nil NULL
11 #define info(P) (P)->info
12 #define next(P) (P)->next
13 #define head(S) ((S).head)
14 #define tail(S) ((S).tail)
15 typedef int infotype;
16 typedef struct elmQ *address;
17
```

```
18  struct elmQ{
19      infotype info;
20      address next;
21  };
22  /* deklarasi tipe data queue, terdiri dari head dan tail, queue kosong jika
23      head = Nil */
24  struct queue {
25      address head, tail;
26  };
27
28  /*prototype*/
29  /********* pengecekan apakah queue kosong **********/
30  boolean isEmpty(queue Q);
31  /*mengembalikan nilai true jika queue kosong*/
32
33  /********* pembuatan queue kosong *********/
34  void CreateQueue(queue &Q);
35  /* I.S. sembarang
36     F.S. terbentuk queue kosong*/
37
38
39  /********* manajemen memori *********/
40  void alokasi(infotype X);
41  /* mengirimkan address dari alokasi elemen
42     jika alokasi berhasil, maka nilai address tidak Nil dan jika gagal, nilai
43  address Nil */
44
45  void dealokasi(address P);
46  /* I.S. P terdefinisi
47     F.S. memori yang digunakan P dikembalikan ke sistem */
48
49  /********* pencarian sebuah elemen list *********/
50  address findElm(queue Q, infotype X);
51  /* mencari apakah ada elemen queue dengan info(P) = X
52     jika ada, mengembalikan address elemen tab tsb, dan Nil jika sebaliknya */
53
54  boolean fFindElm(queue Q address P);
55  /* mencari apakah ada elemen queue dengan alamat P
56     mengembalikan true jika ada dan false jika tidak ada */
57
58  /********* penambahan elemen **********/
59  void insert(queue &Q, address P);
60  /* I.S. sembarang, P sudah dialokasikan
61     F.S. menempatkan elemen beralamat P pada akhir queue*/
62
63  /********* penghapusan sebuah elemen **********/
64  void delete(queue &Q, address &P);
65  /* I.S. queue tidak kosong
66     F.S. menunjuk elemen pertama queue, head dari queue menunjuk pada next
67      elemen head yang lama. Queue mungkin menjadi kosong */
68
69  /********* proses semua elemen queue**********/
70  void printInfo(queue Q);
71  /* I.S. queue mungkin tidak kosong
72     F.S. jika queue tidak kosong, maka menampilkan semua info yang ada pada
73  queue */
74
75  void nbList(queue Q);
76  /* mengembalikan jumlah elemen pada queue */
77
78  void delAll(queue &Q);
79  /* menghapus semua elemen pada queue dan semua elemen di-dealokasi */
80
81  #endif
```

8.4 *Queue (Representasi Tabel)*

8.4.1 Pengertian

Pada dasarnya representasi *queue* menggunakan tabel sama halnya dengan menggunakan *pointer*. Perbedaan yang mendasar adalah pada management memori serta keterbatasan jumlah antriannya. Untuk lebih jelasnya perhatikan perbedaan representasi tabel dan *pointer* pada *queue* dibawah ini.

Tabel 8-1 Perbedaan *Queue* Representasi Table dan *Pointer*

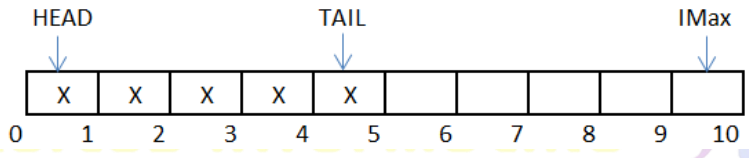
No	Representasi Table	Representasi *Pointer*
1	Jumlah *Queue* terbatas	Jumlah *Queue* tak-terbatas
2	Tidak ada management memori	ada management memori

Ada beberapa macam-macam bentuk *queue*, yaitu:

A. Alternatif 1

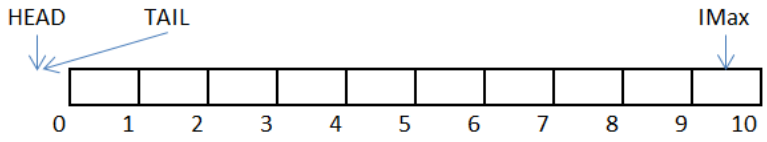
Tabel dengan hanya representasi *TAIL* adalah indeks elemen terakhir, *HEAD* selalu di-*set* sama dengan 1 jika *Queue* tidak kosong. Jika *Queue* kosong, maka *HEAD*=0.

Ilustrasi *Queue* tidak kosong, dengan 5 elemen :



Gambar 8-8 *Queue* 1 Tidak Kosong Representasi Tabel

Ilustrasi *Queue* kosong:

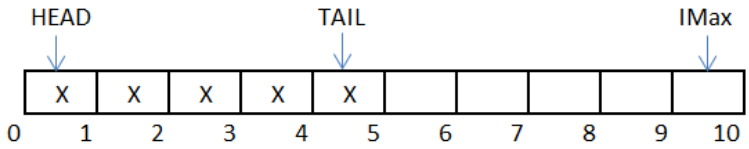


Gambar 8-9 *Queue* 1 Kosong Representasi Tabel

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.

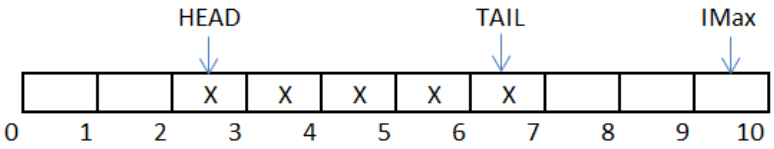
B. Alternatif 2

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:



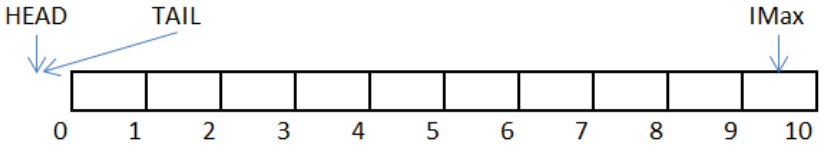
Gambar 8-10 *Queue* 2 Tidak Kosong 1 Representasi Table

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).



Gambar 8-11 *Queue* 2 Tidak Kosong 2 Representasi Tabel

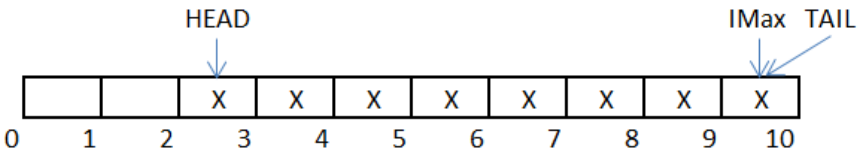
Ilustrasi *Queue* kosong:



Gambar 8-12 *Queue* 2 Kosong Representasi Tabel

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 :

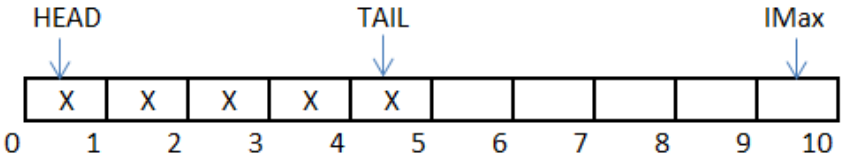


Gambar 8-13 *Queue* 2 Penuh "semu"

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

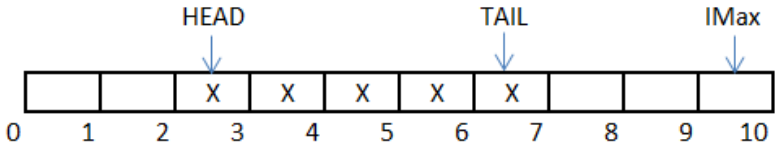
C. Alternatif 3

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:



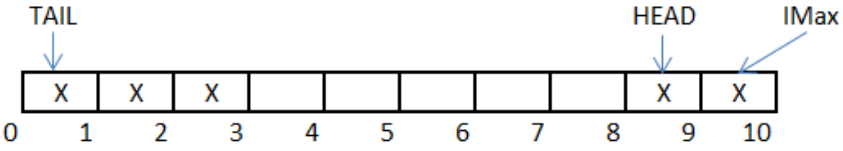
Gambar 8-14 *Queue* 3 Tidak Kosong 1 Representasi Table

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



Gambar 8-15 *Queue* 3 Tidak Kosong 2 Representasi Table

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



Gambar 8-16 *Queue* 3 Tidak Kosong 3 Representasi Table

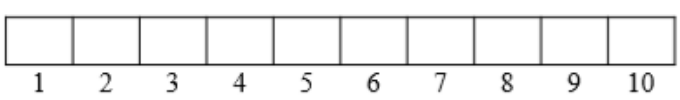
Algoritma untuk penambahan elemen: jika masih ada tempat adalah dengan"memajukan" *TAIL*. Tapi jika *TAIL* sudah mencapai IdxMax, maka *successor* dari IdxMax adalah 1 sehingga *TAIL* yang baru adalah 1. Jika *TAIL* belum mencapai IdxMax, maka algoritma penambahan elemen sama dengan alternatif II. Kasus 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 *successor* dari indeks yang diubah/"maju" dibuat Seperti pada algoritma penambahan elemen: jika *HEAD* mencapai IdxMAx, maka *successor* 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.

Contoh Kasus Alternatif 2

Antrian kosong (*Head*=0, *Tail*=0)



Setelah penambahan elemen 8,3,6,9,15,1 (*Head*=1, *Tail*=6)

8	3	6	9	15	1				
1	2	3	4	5	6	7	8	9	10

Setelah penghapusan sebuah elemen (*Head*=2, *Tail*=6)

	3	6	9	15	1				
1	2	3	4	5	6	7	8	9	10

Algoritma

```
Kamus Umum
constant NMax: integer = 100
constant Nil: integer = 0
type infotype = integer
type   Queue   =   <TabQ:   array[1..NMax]   of
infotype;
Head,Tail: integer>
Function EmptyQ(Q:Queue) → boolean
{True jika Q antrian kosong}
Procedure CreateEmpty(output Q:Queue)
{I.Q. -
 F.Q. Terdefinisi antrian kosong Q}
```

C++

```
const int NMax=100;
const int Nil=0;
typedef int infotype queue {
    infotype TabQ[NMax];
    int head, tail;
}

bool Empty(Queue Q){
    /* True jika Q merupakan antrian
kosong */
}

void CreateEmpty(){
    /* membuat antrian kosong */
}
```

Algoritma

```
function IsFullQ(Q:Queue) → boolean
{True Q penuh}
kamus:
algoritma
    → (Q.Head = 1) and (Q.Tail=NMax)
```

C++

```
bool isFull(Queue Q){
    /* True jika Q penuh */
    if(head(Q)=1 && tail(Q)=Nmax){
        return true;
    }
}
```

Algoritma

```
Procedure AddQ(input/output Q:Queue; input X:infotype)
{I.Q. Q antrian, terdefinisi, mungkin kosong}
{F.Q. X elemen terakhir Q jika antrian tidak penuh}
Kamus
  i:integer
Algoritma
   if IsFull(Q) then
      output('Antrian penuh')
   else
     if Empty(Q) then
        Q.Head ← 1
        Q.Tail ← 1
     else
       if Q.Tail<NMax then
          Q.Tail ← Q.Tail + 1
       else
              for i←Q.Head to Q.Tail do
                Q.TabQ[i-Q.Head+1] ← Q.TabQ[i]
       Q.Tail ← Q.Tail-Q.Head+2
            Q.Head ← 1
        Q.TabQ[Q.Tail] ← X      {menyisipkan X}
```

C++

```
void addQ(Queue Q, infotype x){
 int i;
 if (isFull(Q)){
    cout >> "Antrian Penuh";
 }else{
   if(Empty(Q)){
     head(Q) = 1;
     tail(Q) = 1;
   }else{
     if(tail(Q)>NMax){
       tail(Q)++;
     }else{
       for(i=head(Q);i<=tail(Q);i++){
        TabQ[i+head(Q)+1](Q)=
        TabQ[i](Q);
       }
        tail(Q) = tail(Q)-head(Q)+2;
        head(Q) = 1;
        TabQ[tail(Q)](Q) = x;
        /*menyisipkan x */
        }
     }
  }
}
```

```
Procedure DelQ(input/output Q:Queue; output X:infotype)
{I.Q. Q antrian, terdefinisi, tidak kosong}
{F.Q. X elemen antrian yang dihapus}
Kamus
  i:integer
Algoritma
   X ← Q.TabQ[Q.Head]
  if Q.Head=Q.Tail then
     CreateEmpty(Q)
  else
      Q.Head ← Q.Head+1
```

```
void DelQ(Queue Q, infotype x){
    int i;
    x= TabQ[head(Q)](Q);
    if(head(Q)==tail(Q)){
        CreateEmpty(Q);
    }else{
        head(Q);
    }
}
```

8.4.2 Primitif-Primitif dalam *Queue*

Berikut ini contoh program *queue*.h dalam ADT *queue* menggunakan representasi table.

```
1  /*file : queue .h
2   contoh ADT queue dengan representasi tabel*/
3
4  #ifndef QUEUE_H_INCLUDE
5  #define QUEUE_H_INCLUDE
6  #include <stdio.h>
7  #include <conio.h>
8
9  struct infotype {
10     char id[20];
11     char nama[20];
12 };
13
14 struct Queue {
15     infotype info[3];
16     int head;
17     int tail;
18 };
19
20 /* prototype */
21 /******** pengecekan apakah Queue penuh *********/
22 int isFull(queue Q):
23 /* mengembalikan nilai 0 jika queue penuh */
24
25 /******** pengecekan apakah Queue kosong ********/
26 int isEmpty(queue Q);
27 /* mengembalikan nilai 0 jika queue kosong */
28
29 /********* pembuatan queue *********/
30 void CreateQueue(queue &Q);
31 /* I.S. sembarang
32    F.S. terbentuk queue dengan head = -1 dan tail = -1 */
33
34 /********* penambahan elemen pada queue *********/
35 void enQueue(queue &Q, infotype X);
36 /* I.S. queue mungkin kosong
37    F.S. menambahkan elemen pada stack dengan nilai X */
38
39 /********* penghapusan elemen pada queue *********/
40 void deQueue(queue &Q);
41 /* I.S. queue tidak kosong
42    F.S. head = head + 1 */
43
44 /********* proses semua elemen pada queue *********/
45 void viewQueue(queue Q);
46 /* I.S. queue mungkin kosong
47    F.S. jika queue tidak kosong menampilkan semua info yang ada pada queue */
48
49 #endif
```

8.5 *Latihan Queue*

1. Buatlah ADT *Queue* menggunakan *ARRAY* sebagai berikut di dalam *file* **"*queue*.h"**:

```
Type infotype: integer
Type Queue:  <
     info : array [5] of infotype {index array dalam C++
     dimulai dari 0}
     head, tail : integer
>
prosedur CreateQueue (in/out Q: Queue)
fungsi isEmptyQueue (Q: Queue) → boolean
fungsi isFullQueue (Q: Queue) → boolean
prosedur enqueue (in/out Q: Queue, in x: infotype)
fungsi dequeue (in/out Q: Queue) → infotype
prosedur printInfo (in Q: Queue)
```

Buatlah implementasi ADT *Queue* pada *file* "*queue*.cpp" dengan menerapkan mekanisme *queue* Alternatif 1 (*head* diam, *tail* bergerak).

```
1   int main() {
2       cout << "Hello World" << endl;
3       Queue Q;
4       createQueue(Q);
5
6       cout<<"----------------------"<<endl;
7       cout<<" H - T \t | Queue info"<<endl;
8       cout<<"----------------------"<<endl;
9       printInfo(Q);
10      enqueue(Q,5); printInfo(Q);
11      enqueue(Q,2); printInfo(Q);
12      enqueue(Q,7); printInfo(Q);
13      dequeue(Q); printInfo(Q);
14      enqueue(Q,4); printInfo(Q);
15      dequeue(Q); printInfo(Q);
16      dequeue(Q); printInfo(Q);
17
18      return 0;
19  }
```

Gambar 8-18 *Main Queue*

```
Hello world!
----------------------
 H - T    | Queue Info
----------------------
 -1 - -1  | empty queue
 0 - 0    | 5
 0 - 1    | 5 2
 0 - 2    | 5 2 7
 0 - 1    | 2 7
 0 - 0    | 7
 0 - 1    | 7 4
 0 - 0    | 4
 -1 - -1  | empty queue
```

Gambar 8-17 *Output Queue*

2. Buatlah implementasi ADT *Queue* pada *file* "*queue*.cpp" dengan menerapkan mekanisme *queue* Alternatif 2 (*head* bergerak, *tail* bergerak).
3. Buatlah implementasi ADT *Queue* pada *file* "*queue*.cpp" dengan menerapkan mekanisme *queue* Alternatif 3 (*head* dan *tail* berputar).