Nama : Abd. Rafiq Anwar

NIM : H071211029

Kelas : Struktur Data A

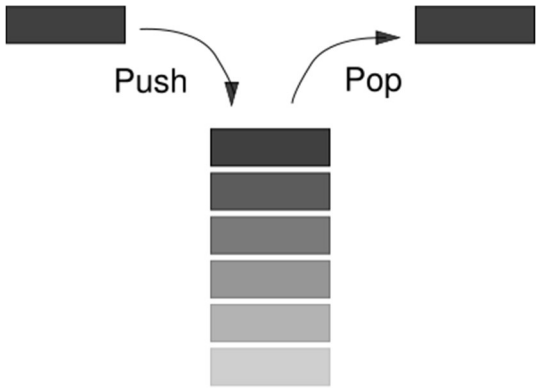
STACK DAN QUEUE

A. Stack

1. Pendahuluan

Stack (tumpukan) dan *queue* (antrian) sebenarnya adalah sebuah cara dalam mengorganisasikan data-data yang dimiliki. Ibarat seseorang yang menyimpan buku-bukunya, ada yang disusun dengan cara ditumpuk, ada juga yang dijejerkan di dalam lemari.

Kaidah utama dalam konsep *stack* adalah LIFO yang merupakan singkatan dari *Last In First Out,* artinya adalah data yang terakhir kali dimasukkan atau disimpan, maka data tersebut adalah yang pertama kali akan diakses atau dikeluarkan. Gambar di bawah ini mengilustrasikan kerja sebuah *stack*.



2. Deklarasi *stack* dalam program

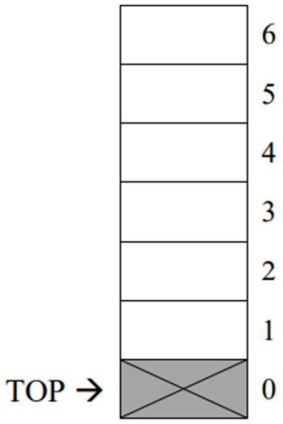
Sebuah *stack* di dalam program komputer dideklarasikan sebagai sebuah tipe bentukan baru, di dalam Bahasa C, biasa disebut *struct*. Sebuah struktur data dari sebuah *stack* setidaknya harus mengandung dua buah variabel, yakni variabel **TOP** yang akan berguna sebagai penanda bagian atas tumpukan dan **ARRAY DATA** dari yang akan menyimpan data-data yang dimasukkan ke dalam *stack* tersebut. Berikut adalah *syntax* untuk mendeklarasikan struktur data dari sebuah *stack* menggunakan Bahasa C:

```
typedef struct
{
      int TOP;
      int data[max+1];
}Stack;
```

dimana, nilai MAX didefinisikan sebagai jumlah tumpukan maksimum yang dapat disimpan dalam *stack*. Setelah strukutr data dari *stack* didefinisikan dengan *syntax* di atas, maka setelah itu dapat dibuat variabel-variabel baru yang mengacu pada tipe data **Stack** di atas, misalkan membuat sebuah variabel bernama **tumpukan** yang bertipe Stack:

```
Stack tumpukan;
```

Dalam tulisan ini, sebuah *stack* didefinisikan dengan *array* berukuran MAX + 1, maksudnya adalah agar elemen *array* ke-0 tidak digunakan untuk menyimpan data, melainkan hanya sebagai tempat „singgah" sementara untuk variabel TOP. Sehingga, jika TOP berada pada elemen *array* ke-0, berarti *stack* tersebut dalam kondisi kosong (tidak ada data yang disimpan). Berikut adalah ilustrasi dari sebuah *stack* kosong dengan ukuran nilai MAX = 6:



3. Operasi-operasi dasar dalam *stack*

Sebuah *stack* setidaknya memiliki lima buah operasi-operasi dasar, yakni:

a. Prosedur createEmpty

Prosedur ini berfungsi untuk mengosongkan *stack* dengan cara meletakkan TOP ke posisi ke-0. Berikut adalah deklarasi prosedur createEmpty dalam Bahasa C:

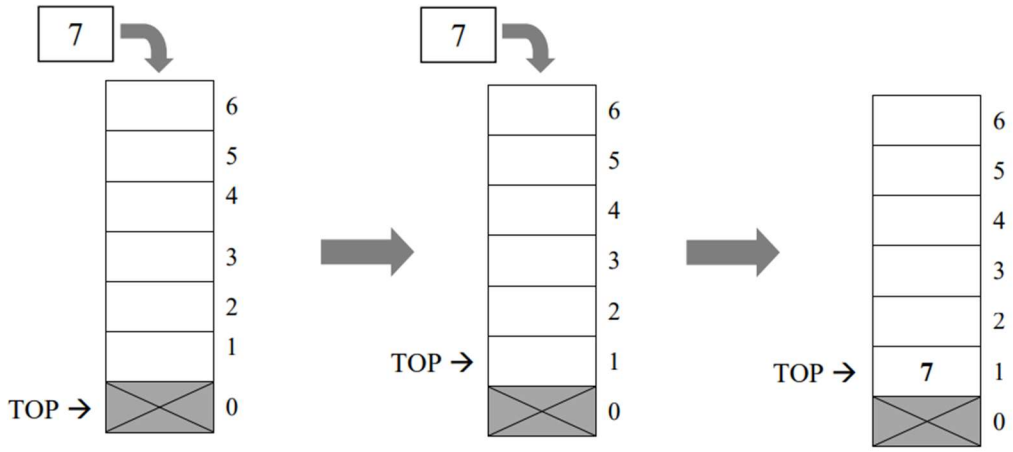
```
void createEmpty()
{
      tumpukan.TOP = 0;
}
```

b. Prosedur push

Prosedur ini berfungsi untuk **memasukkan** sebuah nilai/ data ke dalam *stack*. Sebelum sebuah nilai/ data dimasukkan ke dalam *stack*, prosedur ini terlebih dahulu akan menaikkan posisi TOP satu level ke atas. Misalkan kondisi *stack* masih kosong (TOP = 0), lalu prosedur push akan menaikkan posisi TOP satu level ke atas, yakni ke posisi 1 (TOP = 1), baru setelah itu data dimasukkan ke dalam *array* pada indeks ke-1 (yakni indeks dimana TOP berada). Berikut adalah deklarasi prosedur push dalam Bahasa C:

```
void push(int x)
{
      tumpukan.TOP = tumpukan.TOP + 1;
      tumpukan.data[tumpukan.TOP] = x;
}
```

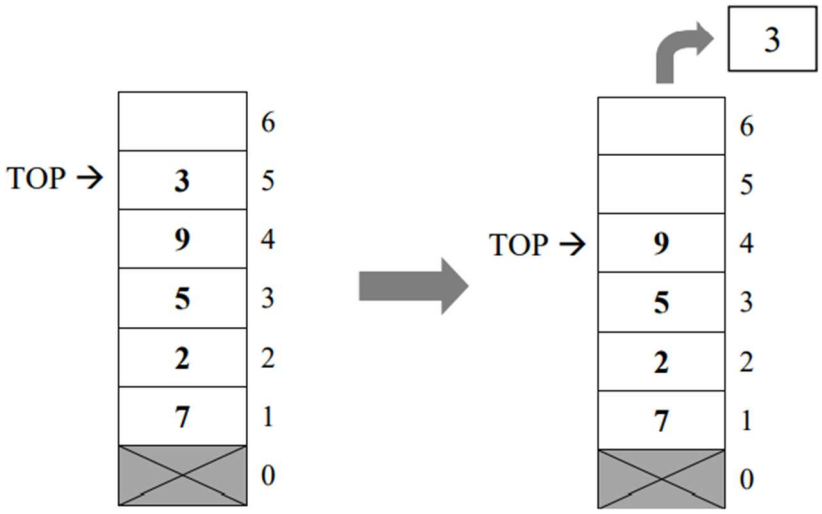
Pada deklarasi prosedur push di atas, prosedur memiliki sebuah parameter formal yang bernama „**x**“ yang bertipe *integer*. Parameter formal „x“ ini berguna untuk menerima kiriman nilai dari program utama (void main()) yakni berupa sebuah bilangan *integer* yang akan dimasukkan ke dalam *stack*. Sebelum nilai pada variabel „x“ dimasukkan ke dalam *stack*, terlebih dahulu posisi TOP dinaikkan satu level, baru setelah itu nilai pada variabel „x“ dimasukkan ke dalam *array* data pada indeks dimana TOP itu berada.



c. Prosedur pop

Prosedur ini berfungsi untuk **mengeluarkan**/ menghapus nilai terakhir (yang berada pada posisi paling atas) dari *stack*, dengan cara menurunkan nilai TOP satu level ke bawah. Misalkan TOP berada pada indeks ke-5, maka ketika akan mengeluarkan/ menghapus data pada posisi paling atas (pada posisi TOP), prosedur ini akan menurunkan posisi TOP ke indeks *array* ke-4. Berikut deklarasi prosedur pop dalam Bahasa C:

```
void pop()
{
      tumpukan.TOP = tumpukan.TOP - 1;
}
```



d. Fungsi IsEmpty

Fungsi ini berfungsi untuk melakukan pengecekan terhadap *stack*, apakah *stack* tersebut **kosong** atau tidak. Jika *stack* tersebut kosong (artinya, TOP berada pada posisi 0), maka fungsi akan mengembalikan nilai 1 (*true*), tetapi jika *stack* tersebut tidak kosong/ berisi (artinya, TOP tidak berada pada posisi 0), maka fungsi akan mengembalikan nilai 0 (*false*). Berikut deklarasi fungsi IsEmpty dalam Bahasa C:

```
int IsEmpty()
{
     if (tumpukan.TOP == 0)
          return 1;
     else
          return 0;
}
```

e. Fungsi IsFull

Fungsi ini berfungsi untuk melakukan pengecekan terhadap *stack*, apakah *stack* tersebut **penuh** atau tidak. Jika *stack* tersebut penuh (artinya, TOP berada pada posisi MAX), maka fungsi akan mengembalikan nilai 1 (*true*), tetapi jika *stack* tersebut tidak penuh (artinya, TOP tidak berada pada posisi MAX), maka fungsi akan mengembalikan nilai 0 (*false*). Berikut deklarasi fungsi IsFull dalam Bahasa C:

```
int IsFull()
{
     if (tumpukan.TOP == MAX)
          return 1;
     else
          return 0;
}
```

4. Contoh program implementasi *stack*

Berikut adalah contoh kode program dalam Bahasa C yang mengimplementasikan konsep *stack*. Pada program ini, *user* disuguhi beberapa menu utama yang akan dipilih oleh *user*. Menu pertama, "Cek kondisi stack" akan melakukan pengecekan terhadap kondisi *stack*. Menu kedua, "Tambah data" akan melakukan pengisian sebuah nilai ke dalam *stack*. Menu ketiga, "Keluarkan isi stack", akan menampilkan semua isi *stack* dan akan mengosongkan *stack*. Menu keempat, "Kosongkan stack", akan melakukan pengosongan *stack*, dan menu kelima, "Keluar", akan menghentikan eksekusi program (selesai menggunakan program).

```
//program implementasi stack
#include <stdio.h>
#include <conio.h>
#define max 5

typedef struct {
    int top;    //untuk mencacah indeks dari stack
    int data[max+1];
}stack;

stack tumpukan;

void createEmpty();
int IsEmpty();
int IsFull();
void push(int x);
void pop();

void main(){
    int lagi;
    int input;
    int pilih;

    createEmpty();

    pilih = 0;
    while (pilih != 5){
    //Menu utama
    system("cls");
    puts("====================================");
    puts("  MENU UTAMA");
    puts("====================================");
    puts("1. Cek kondisi Stack");
```

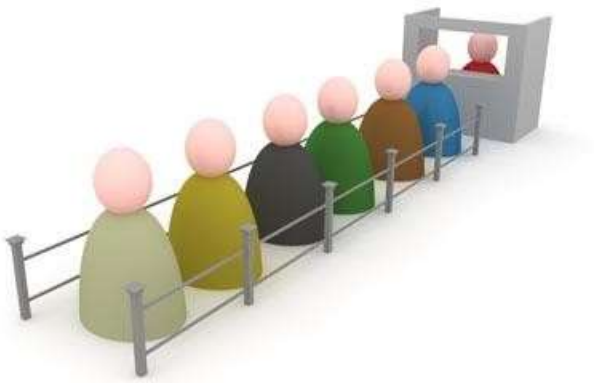
```
    puts("2. Tambah data");
    puts("3. Keluarkan isi stack");
    puts("4. Kosongkan stack");
    puts("5. Keluar");
    printf("Pilihan: ");
    scanf("%d",&pilih);

    switch(pilih){
        case 1: if (IsEmpty() == 1)
                    puts("Stack masih kosong");
                else if ((IsEmpty() == 0) && (IsFull() == 0))
                    puts("Stack sudah terisi, tapi belum penuh");
                else
                    puts("Stack sudah penuh");
                getch();
                break;
        case 2: if (IsFull() == 1)
                    puts("Stack sudah penuh.");
                else{
                    printf("Masukkan data: ");
                    scanf("%d",&input);
                    push(input);
                    printf("%d",tumpukan.top);
                    printf("%d",IsFull());
                    printf("%d",IsEmpty());
                }
                break;
        case 3: while (IsEmpty() == 0){
                    printf("%d \n",tumpukan.data[tumpukan.top]); pop();
                    }
                    getch();
                    break;
        case 4: createEmpty();
                    puts("Stack sudah kosong. Top = 0");
                    getch();
                    break;
        case 5: puts(“Byeee……”);
                    getch();
                    break;
        }
    }
}

//DEKLARASI OPERASI-OPERASI DASAR STACK
void createEmpty(){
    tumpukan.top = 0;
```

```
}

int IsEmpty(){
    if (tumpukan.top == 0)
    return 1;
    else
    return 0;
}

int IsFull(){
    if (tumpukan.top == max)
    return 1;
    else
    return 0;
}

void push(int x){
    tumpukan.top = tumpukan.top + 1;
    tumpukan.data[tumpukan.top] = x;
}

void pop(){
    tumpukan.top = tumpukan.top - 1;
}
```

B. Queue

1. Pendahuluan

Kaidah utama dalam konsep *queue* adalah FIFO yang merupakan singkatan dari *First In First Out,* artinya adalah data yang pertama kali dimasukkan atau disimpan, maka data tersebut adalah yang pertama kali akan diakses atau dikeluarkan. Analoginya sama dengan antrian di sebuah loket pembelian tiket kereta, orang yang datang lebih dahulu, maka akan dilayani terlbih dahulu, dan akan selesai lebih dulu dari orang-orang yang datang setelahnya. Gambar di bawah ini mengilustrasikan kerja sebuah *queue*:



2. Deklarasi *queue* dalam program

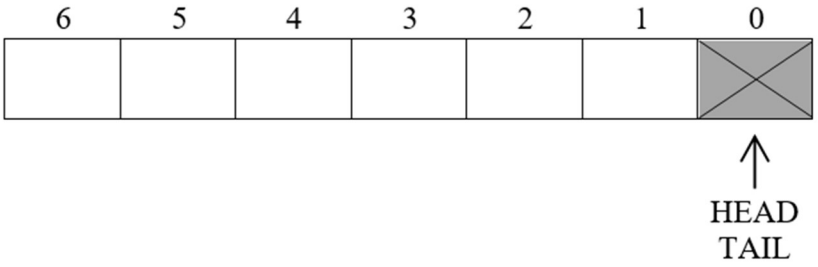
Sebuah *queue* di dalam program komputer dideklarasikan sebagai sebuah tipe bentukan baru, di dalam Bahasa C, biasa disebut *struct*. Sebuah struktur data dari sebuah *queue* setidaknya harus mengandung dua tiga variabel, yakni variabel **HEAD** yang akan berguna sebagai penanda bagian depan antrian, variabel **TAIL** yang akan berguna sebagai penanda bagian belakang antrian dan **ARRAY DATA** dari yang akan menyimpan data-data yang dimasukkan ke dalam *queue* tersebut. Berikut adalah *syntax* untuk mendeklarasikan struktur data dari sebuah *queue* menggunakan Bahasa C:

```
typedef struct
{
     int HEAD, TAIL;
     int data[max+1];
}Queue;
```

dimana, nilai MAX didefinisikan sebagai jumlah tumpukan maksimum yang dapat disimpan dalam *queue*. Setelah strukutr data dari *queue* didefinisikan dengan *syntax* di atas, maka setelah itu dapat dibuat variabel-variabel baru yang mengacu pada tipe data **Queue** di atas, misalkan membuat sebuah variabel bernama **antrian** yang bertipe Queue:

```
Queue antrian;
```

Dalam tulisan ini, sebuah *queue* didefinisikan dengan *array* berukuran MAX + 1, maksudnya adalah agar elemen *array* ke-0 tidak digunakan untuk menyimpan data, melainkan hanya sebagai tempat „singgah" sementara untuk variabel HEAD dan TAIL. Sehingga, jika HEAD dan TAIL berada pada elemen *array* ke-0, berarti *queue* tersebut dalam kondisi kosong (tidak ada data yang disimpan). Berikut adalah ilustrasi dari sebuah *queue* kosong dengan ukuran nilai MAX = 6:



3. Operasi-operasi dasar dalam *queue*

Pada dasarnya, operasi-operasi dasar pada *queue* mirip dengan operasi-operasi dasar pada *stack*. Perbedaannya hanya pada prosedur push dan pop saja. Pada *queue*, prosedur yang berfungsi untuk memasukkan data/ nilai ke dalam antrian adalah **enqueue**, sedangkan prosedur untuk mengeluarkan data/ nilai dari antrian adalah **dequeue**.

a. Prosedur createEmpty

   Sama pada *stack*, prosedur ini berfungsi untuk mengosongkan *queue* dengan cara meletakkan HEAD dan TAIL pada indeks *array* ke-0. Berikut deklarasi prosedur createEmpty pada *queue* dalam Bahasa C:

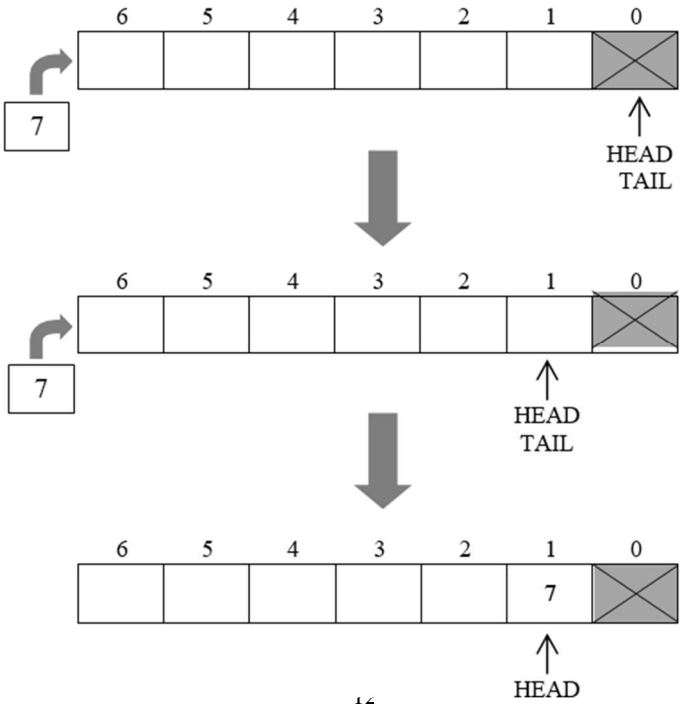
```
void createEmpty()
{
      antrian.HEAD = 0;
      antrian.TAIL = 0;
}
```

b. Prosedur enqueue

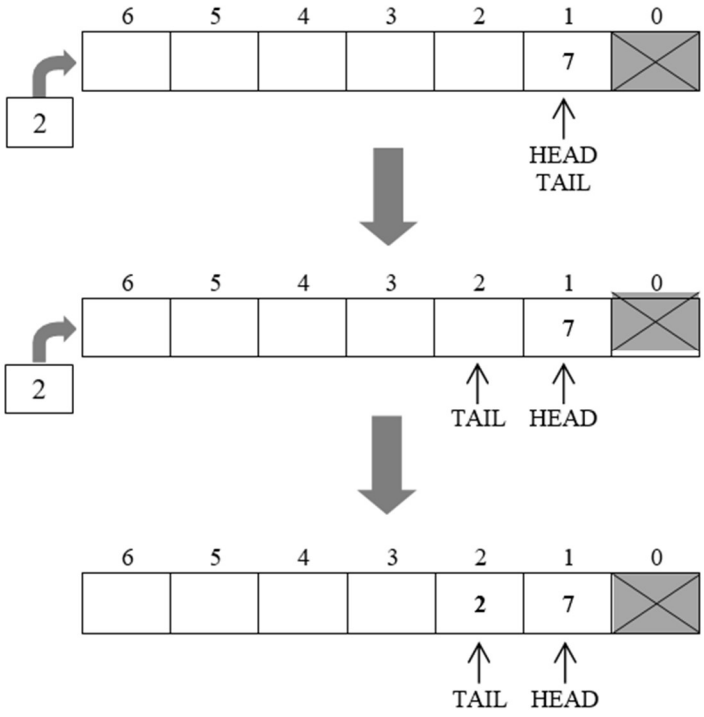
Prosedur ini digunakan untuk **memasukkan** sebuah data/ nilai ke dalam *queue*. Sebelum sebuah data/ nilai dimasukkan ke dalam *queue*, maka prosedur ini terlebih dahulu melakukan pengecekan terhadap posisi HEAD dan TAIL. Jika posisi HEAD dan TAIL masih berada pada indeks ke-0 (artinya *queue* masih kosong), maka prosedur ini akan menempatkan HEAD dan TAIL pada indeks ke-1 terlebih dahulu, baru setelah itu memasukkan data/ nilai ke dalam *array* data *queue*. Namun, jika posisi HEAD dan TAIL tidak berada pada posisi ke-0, maka posisi TAIL yang akan dinaikkan satu level. Jadi, pada proses enqueue, TAIL-lah yang berjalan seiring masuknya data baru ke dalam antrian, sedangkan HEAD akan tetap pada posisi ke-1. Berikut deklarasi prosedur enqueue dalam Bahasa C:

```
void enqueue(int x)
{
    if ((antrian.HEAD == 0) && (antrian.TAIL == 0))
    {
        antrian.HEAD = 1;
        antrian.TAIL = 1;
    }
    else
    {
        antrian.TAIL = antrian.TAIL + 1;
    }
    antrian.data[antrian.TAIL] = x;
}
```

Pada deklarasi prosedur enqueue di atas, prosedur memiliki sebuah parameter formal yang bernama „**x**" yang bertipe *integer*. Parameter formal „x" ini berguna untuk menerima kiriman nilai dari program utama (void main()) yakni berupa sebuah bilangan *integer* yang akan dimasukkan ke dalam *queue*. Gambar di bawah ini mengilustrasikan proses *enqueue* ke dalam sebuah *queue* yang masih kosong.



Berikutnya, gambar di bawah ini akan mengilustrasikan proses *enqueue* ke dalam sebuah *queue* yang telah berisi data (*queue* tidak kosong).



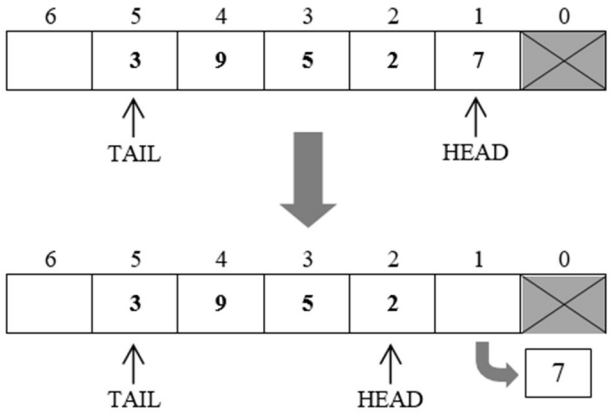
c. Prosedur dequeue

Prosedur ini digunakan untuk **mengeluarkan** atau membuang sebuah data/ nilai yang paling awal masuk (yang berada pada posisi HEAD, yakni yang paling depan dari antrian) ke dalam *queue*. Pekerjaan yang dilakukan oleh prosedur ini adalah menaikkan nilai HEAD satu level. Jadi, setiap satu kali data dikeluarkan, maka posisi HEAD naik bertambah satu level. Misalkan HEAD berada pada indeks ke-1, maka ketika akan mengeluarkan/ menghapus data pada posisi paling depan (pada posisi HEAD), prosedur ini akan menaikkan posisi HEAD ke indeks *array* ke-2. Berikut deklarasi prosedur pop dalam Bahasa C:

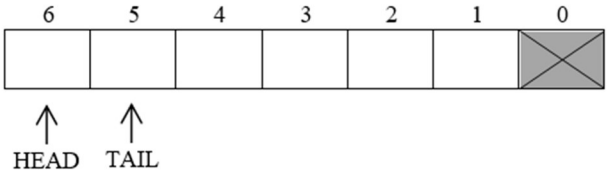
```
void Dequeue(){
    if (q.head > q.tail)     {
        q.head = 0;
        q.tail = 0;
    }
    q.head = q.head + 1;
```

```
}
```

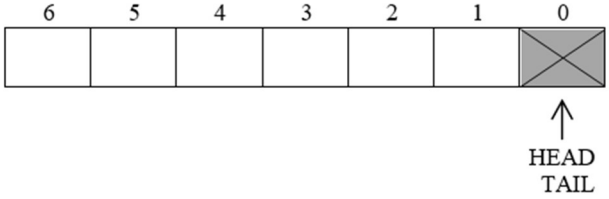
Ketika posisi HEAD sudah melewati posisi TAIL (HEAD > TAIL), berarti sudah tidak ada lagi data/ nilai di dalam *queue* tersebut, maka saat itu terjadi, HEAD dan TAIL dikembalikan ke posisi ke-0. Gambar di bawah ini mengilustrasikan cara kerja prosedur dequeue:



Ketika nilai terakhir (yakni nilai 3) dikeluarkan, maka posisi HEAD dan TAIL akan menjadi seperti ini:



Maka, ketika kondisi HEAD > TAIL terjadi seperti ilustrasi di atas, maka HEAD dan TAIL dikembalikan ke indeks *array* ke-0.



d. Fungsi IsEmpty

Sama seperti fungsinya pada *stack,* fungsi ini berfungsi untuk melakukan pengecekan terhadap *queue*, apakah *queue* tersebut **kosong** atau tidak. Jika *queue* tersebut kosong (artinya, HEAD dan TAIL berada pada posisi 0, atau bisa juga ketika HEAD > TAIL), maka fungsi akan mengembalikan nilai 1 (*true*), tetapi jika *queue* tersebut tidak kosong/ berisi (artinya, HEAD dan TAIL tidak berada pada posisi 0), maka fungsi akan mengembalikan nilai 0 (*false*). Berikut deklarasi fungsi IsEmpty dalam Bahasa C:

```
int IsEmpty()
{
    if ((antrian.HEAD> antrian.TAIL) || (antrian.HEAD == 0) &&
    (antrian.TAIL == 0))
        return 1;
    else
        return 0;
}
```

e. Fungsi IsFull

Fungsi ini berfungsi untuk melakukan pengecekan terhadap *queue*, apakah *queue* tersebut **penuh** atau tidak. Jika *queue* tersebut penuh (artinya, TAIL berada pada posisi MAX), maka fungsi akan mengembalikan nilai 1 (*true*), tetapi jika *queue* tersebut tidak penuh (artinya, TAIL tidak berada pada posisi MAX), maka fungsi akan mengembalikan nilai 0 (*false*). Berikut deklarasi fungsi IsFull dalam Bahasa C:

```
int IsFull()
{
     if (antrian.TAIL == max)
               return 1;
     else
               return 0;
}
```

4. Contoh program implementasi *queue*

Berikut adalah contoh kode program dalam Bahasa C yang mengimplementasikan konsep *queue*. Pada program ini, *user* akan menginputkan data satu per satu, kemudian setelah selesai menginputkan data ke dalam *queue*, maka program akan menampilkan semua isi *queue*.

```
//program implementasi queue
#include <stdio.h>
#define max 5

typedef struct{
    int HEAD, TAIL; //untuk mencacah indeks dari stack
    int data[max+1];
}Queue;

Queue antrian;

void createEmpty();
int IsEmpty();
int IsFull();
void Enqueue(int x);
void Dequeue();

void main(){
    int lagi;
    int input;
    createEmpty();

    //mengisi queue lagi = 1;
    while (lagi == 1){
        if ((IsEmpty() == 1) || (IsFull() == 0)){
            system("cls");
            printf("Antrian masih tersedia.\n");
            printf("Masukkan nilai: ");
            scanf("%d",&input);
            Enqueue(input);

            printf("Tambah data (1/0)?");
            scanf("%d",&lagi);
        }
        else if (IsFull() == 1){
            printf("Antrian sudah penuh.\n"); lagi = 0;
        }
    }
    //menampilkan isi stack
    while (IsEmpty() == 0){
```

```
        printf("%d ",antrian.data[antrian.HEAD]);
        Dequeue();
    }
    return 0;
}

void createEmpty(){
    antrian.HEAD= 0;
    antrian.TAIL = 0;
}

int IsEmpty(){
    if ((antrian.HEAD> antrian.TAIL) || (antrian.HEAD == 0) && (antrian.TAIL ==0))
        return 1;
    else
        return 0;
}

int IsFull(){
    if (antrian.TAIL == max)
        return 1;
    else
        return 0;
}

void Enqueue(int x){
    if ((antrian.HEAD == 0) && (antrian.TAIL == 0)){
        antrian.HEAD = 1;
        antrian.TAIL = 1;
    }
    else{
        antrian.TAIL = antrian.TAIL + 1;
    }
    antrian.data[antrian.TAIL] = x;
}

void Dequeue(){
    if (antrian.HEAD > antrian.TAIL){
        antrian.HEAD = 0;
        antrian.TAIL = 0;
    }
    antrian.HEAD = antrian.HEAD + 1;
}
```