Pemahaman Modul

Praktikum Struktur Data.

Kode_Asisten_PJ_:_MTN Kelompok = A11.4301U

NIM :A11.2012.06758 NIM : A11.2012.06889

Nama :Muhammad Adhi D Nama : Agung Madra Jaya

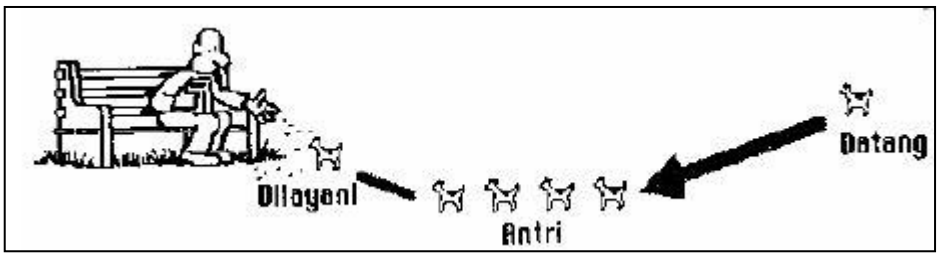
PSDA – 02 (Queue).

A. Pendahuluan Mengenai Queue

Stack merupakan "List Linier" yang dikenali elemen pertamanya (HEAD), dan elemen terakhirnya (TAIL).Aturan penambahan dan penghapusan elemen yaitu :

- Penambahan selalu dilakukan pada TAIL.
- Penghapusan selalu dilakukan pada HEAD.

Elemen stack tersusun secara FIFO (*First In First Out*).Queue dapat digambarkan sebagai sebuah lorong lurus, dimana terdapat 2 pintu, sebagai tempat masuk dan keluar dari elemen (seperti dalam antrian).



Beberapa contoh aplikasi Queue :

1. Antrian Job yang harus ditangani oleh OS.

```
typedef int addres;
typedef int infotype;
typedef struct
{
    infotype *T;
    addres HEAD;
    addres TAIL;
    int MAX;
}Queue;
```

- Tipe data bentukan diberi nama "addres" dan "infotype" bertype integer.
- ADT Queue dengan beberapa komponen penyusunnya adalah variabel pointer "*T" sebagai Queue-nya bertype integer.Dimana alokasi memori dilakukan pada saat program berjalan.
- Variabel "HEAD" dan "TAIL" bertype addres (int), sebagai index dari suatu queue yang dapat menunjukan alamat sebuah queue.
- Variabel "MAX" digunakan untuk memberikan informasi mengenai jumlah elemen yang digunakan pada queue.
- ADT diberi nama "Queue".dimana ADT ini bersifat global, artinya dapat di manipulasi dan diakses informasinya dari fungsi manapun.

B. Implementasi Queue dengan Alternatif 1.

B.1 Selektor.

```
#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).MAX
```

Definisi selektor , "**Q**" merupakan sebuah Queue dengan ,

- ✓ **HEAD(Q)** adalah alamat dari elemen HEAD, dimana operasi penghapusan dilakukan.
- ✓ **TAIL(Q)** adalah alamat dari elemen TAIL, dimana operasi penambahan dilakukan.
- ✓ **InfoHead(Q)** adalah nilai dari elemen suatu queue dengan alamat "HEAD".
- ✓ **InfoTail(Q)** adalah nilai dari elemen suatu queue dengan alamat "TAIL".
- ✓ **MaxEl(Q)** adalah informasi dari jumlah elemen maksimum suatu queue.

B.2 Definisi Fungsional.

```
boolean IsMax (Queue Q);
boolean IsEmpty (Queue Q);
void CreateEmpty (Queue *Q, int max);
int NbElmt (Queue Q);
void Dealokasi (Queue *Q);
void Add (Queue *Q,int val);
void Del (Queue *Q, int *val);
void Dealokasi (Queue *Q);
```

a. Int NbElmt.

Fungsi yang digunakan untuk memberikan informasi jumlah elemen yang ada pada kondisi tertentu.

b. Void CreateEmpty.

Membuat stack kosong, dengan parameter pointer (*Q),bertype Queue dan berkapasitas MaxEl, dengan index antara 1~MaxEl, indeks 0 tidak digunakan , dengan ciri Queue kosong adalah nilai dari HEAD(Q) dan TAIL(Q) bernilai NIL, karena memori bersifat dinamis, maka digunakan keyword "malloc".

{**I.S** = Sembarang kondisi.}

{**F.S** = membuat Queue kosong berkapasitas MaxEl.}

```
(*Q).T = (int *) malloc((max+1) * sizeof(int));
   HEAD(*Q) = NIL;
   TAIL(*Q) = NIL;
   MaxEl(*Q) = max;
```

c. Boolean IsEmpty.

Mengecek kondisi dari Queue Q, apakah Queue tersebut kosong?.

Algoritma.

- Jika "HEAD(Q) dan TAIL(Q)" bernilai NIL atau 0, maka **"TRUE",**selain itu **"FALSE".**

  Return ((TAIL(Q)==NIL) AND (HEAD(Q)==NIL)).

d. Boolean IsFull.

Mengecek kondisi dari Queue Q, apakah queue tersebut Penuh?.

Algoritma.

- Jika TAIL(Q) berada di posisi index terakhir DAN HEAD(Q) ada di posisi awal, maka bernilai **"TRUE",** selain itu **"FALSE".**

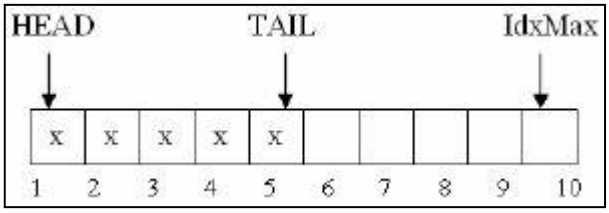
  Return ((TAIL(Q)==MaxEl(Q)) AND (HEAD(Q)==1)).

e. Void Dealokasi.

```
MaxEl(*Q) = NIL;
free((*Q).T);
```

Prosedure ini berfungsi sebagai dealokasi seluruh table memori sekaligus. Ditandai dengan keyword "free". Ukuran queue di set menjadi NIL atau 0, dan semua data pada queue akan di hapus.

f. Void Add (Alternatif 1).



Menambahkan elemen pada queue,dengan penambahakan dilakukan pada **TAIL**. Nilai **HEAD** akan teteap pada posisinya , sedangkan setiap dilakukan penambahan elemen maka nilai dari **TAIL** akan ters bertambah 1.Jika **TAIL** sudah mencapai pada IdxMax , makan artinya queue penuh, dan perlu dilakukan penghapusan agar dapat ditambahkan elemen lagi.

Algoritma

{**I.S** = Slot queue masih ada/ kondisi tidak penuh.}

{**F.S** = menambah 1 buah elemen pada queue.}

- Posisikan HEAD pada elemen pertama, sebagai awalan untuk penambahan elemen.
- Jika masih ada slot kosong, maka majukan TAIL sebesar 1.kemudian isikan elemen yang akan ditambahkan ke dalam queue.
- Jika queue sudah penuh, (TAIL berada pada posisi indexmax), maka tidak dapat dilakukan penambahan, dan harus dilakukan penghapusan elemen.

g. Void Del (Alternatif 1).

Menghapus sebuah elemen dari queue. Penghapusan dilakukan pada elemen HEAD.Ilustrasi penghapusan pada alternatif 1 ini yaitu pada saat orang sedang mengantri, dimana jika orang terdepan telah bergerak maju, orang yang di belakang juga akan ikut bergerak maju satu per satu, hingga elemennya kosong.

Algoritma

{**I.S** = Elemen queue masih ada/ kondisi tidak kosong.}

{**F.S** = 1 eleme pada queue akan terhapus.}

- Jika elemen tidak kosong, maka ambil nilai dari head.dan geser mundur semua komponen elemen mendekati nilai head. Mulai dari index HEAD+1, sampai dengan index ke TAIL.Jika elemen kosong maka penghapusan tidak dapat dilakukan.
- Kemudian pada saat itu juga mundurkan TAIL 1 langkah ke belakang.
- Jika TAIL sudah berada pada NIL (0), maka posisikan juga nilai HEAD pada ke 0. Dengan kata lain queue telah kosong.

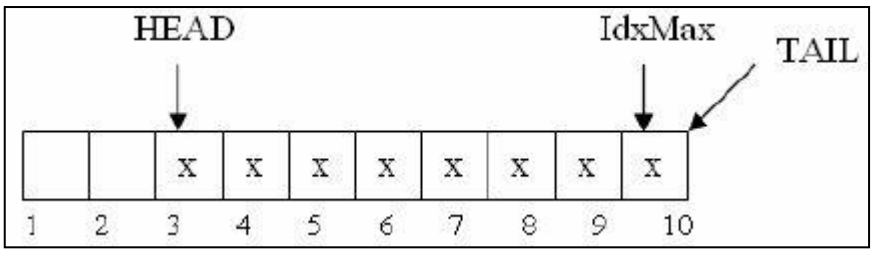
C. Implementasi Queue dengan Alternatif 2.

C.1 Void Add.

Menambahkan elemen ke dalam queue dengan alternatif ke 2 caranya hampir sama dengan kondisi pertama yaitu penambahan dilakukan pada TAIL. Dan seiring dengan penambahannya nilai TAIL akan terus pertambah 1 langkah.

Namun pada kondisi tertentu, didapat jika queue tersebut penuh, namun bersifat "Semu", artinya, posisi TAIL ada pada index elemen terakhir, namun posisi HEAD, tidak pada posisi ke 1 akibat dari aksi penghapusan.

Ilustrasi Semu.



Jika kondisi tersebut terjadi maka perlu dilakukan aksi penggesran tiap elemen mundur 1 langkah ke belakang mendekati head.Sehingga akan tercipta ruangan kosong.

Algoritma.

{**I.S** = Kondisi Queue tidak penuh.}

{**F.S** = 1 elemen ditambahkan ke queue.}

- Jika elemen tidak penuh, maka majukan TAIL sebanyak 1 langkah.
- Jika kondisi semu tercapai, maka geser semua elemen mulai dari HEAD sampai ke TAIL, dengan posisi HEAD akan berpindah pada posisi ke 1.
- Jika elemen sudah penuh, maka perlu dilakukan aksi penghapusan elemen.

C.2 Void Delet.

Menghapus sebuah elemen pada queue. Pada Alternatif 2 ini, jika dilakukan penghapusan elemen, maka HEAD akan bergerak maju ke depan sebesar 1.Pada kondisi tertentu terdapat jika HEAD sama posisinya dengan TAIL.Jika kondisi tersebut terpenuhi maka harus dilakukan penambahan elemen dahulu.

Algoritma.

{**I.S** = Kondisi Queue tidak kosong.}

{**F.S** = 1 elemen dihapus dari queue.}

- Jika elemen tidak kosong, maka ambil nilai dari alamat HEAD, kemudian majukan HEAD sebesar 1.
- Jika kondisi posisi HEAD sama dengan TAIL maka, perlu dilakukan penambahan elemen.

D. Implementasi Queue dengan Alternatif 3

Tabel dengan representasi HEAD dan TAIL yang "berputar" mengelilingi indeks table dari awal sampai ahkir, kemudian kembali ke awal. Jika Queue kosong, maka HEAD=0. Representasi ini memungkinkan tidak perlu lagi ada pergeseran yang harus dilakukan seperti pada alternative II pada saat penambahan elemen.

D.1 Void Add

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 alternative II. Kasus khusus untuk queue kosong karena HEAD harus di set nilainya menjasi 1.

D.2 Void Del

Algoritma untuk penghapusan elemen jika queue tidak kosong : ambil nilai elemen HEAD, kemudian HEAD "maju". Penentuan suatu suksesor dari indeks 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. Alogitma ini efisien karena tidak perlu pergeseran, dan seringkali strategi pemakaian table semacam ini disebut "circular buffer".