MODUL VIII
Struktur Data

Judul	Struktur data Array pada STACK dan QUEUE	
Penyusun	**Distribusi**	**Perkuliahan**
Nixon Erzed	Teknik Informatika Universitas Esa Unggul	Pertemuan – VIII online

Tujuan :

1. Mahasiswa memahami struktur stack dan representasi kontigue stack dengan array
2. Mahasiswa memahami struktur queue dan representasi queue secara linear dengan array

Materi :

Bagian A : Struktur Stack

1. Stack dan spesifikasi stack
2. Single Stack dengan Array
3. Multi-stack dalam satu array
4. Operasi-operasi Dua Stack dengan satu Array

Bagian B : Struktur Queue

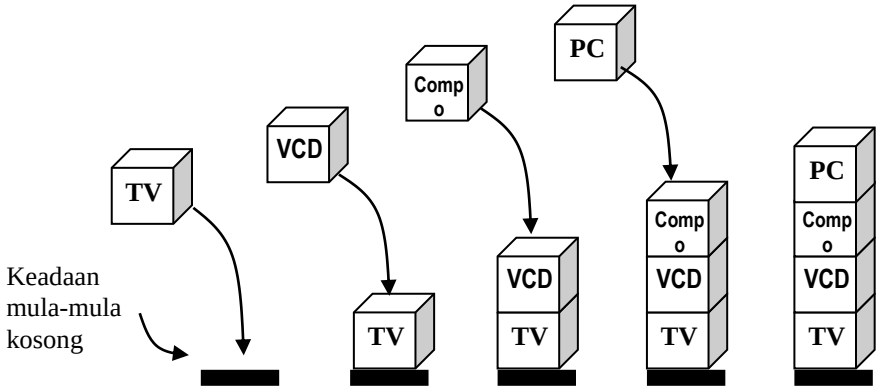
1. Definisi
2. Karakteristik Penting Queue
3. Representasi Queue dengan Array
4. Operasi Dasar
5. Representasi Statik Linear Queue dengan Array

STRUKTUR DATA STACK

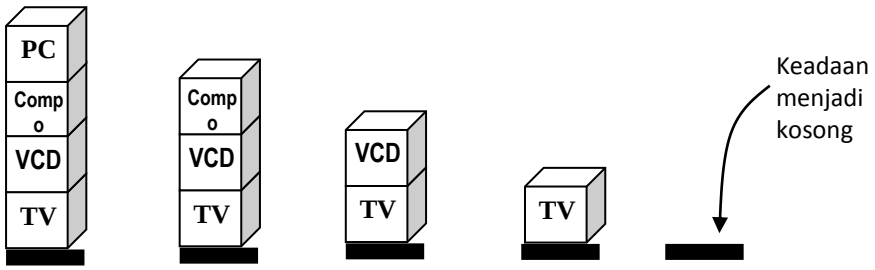
1. Stack dan Spesifikasi Stack

Stack merupakan struktur data yang mengikuti pola tumpukan dari benda. Konsep utamanya adalah LIFO (Last In First Out). Benda terakhir yang masuk kedalam tum-pukan akan menjadi benda pertama yang akan dikeluarkan dari tumpukan.

Ilustrasinya adalah sebagai berikut :



Jika kita ingin mengambil sesuatu dari tumpukan, maka harus diambil dari benda yang paling atas terlebih dahulu yaitu PC Notebook. Misalnya ingin diambil VCD secara langsung maka Compo dan PC akan jatuh.



Dalam implementasi struktur data Stack merupakan kasus khusus dari suatu array ataupun linked list; dimana operasi *penyisipan* dan *–penghapusan* hanya dilakukan disalah satu ujung.
Misalkan *stack S* = (a_1, a_2, …, a_n) maka

1. elemen a_1 adalah elemen terbawah
2. elemen a_i adalah elemen di atas elemen a_{i-1} dimana $1 < i \leq n$

Batasan terhadap stack itu dalam struktur data berimplikasi, jika elemen A, B, C, D, E ditambahkan ke Stack, maka elemen yang pertama dihapus adalah elemen terakhir yang dimasukkan. Yang merupakan implementasi konsep LIFO. Stack sering juga disebut sebagai *Pushdown List.*

Representasi Stack

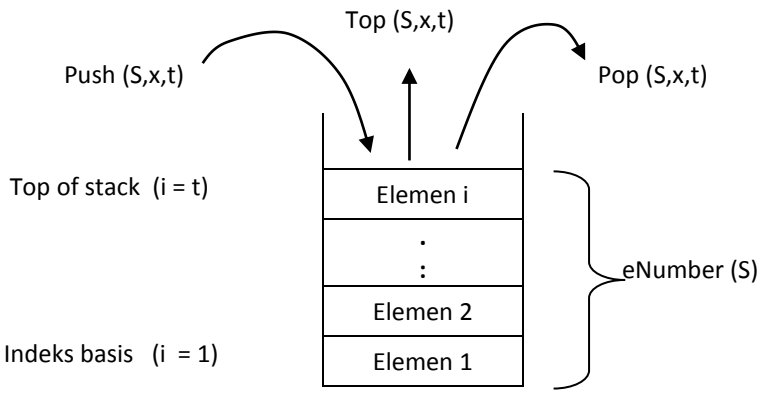
- Stack dengan Array (representasi kontigue)
- Stack dengan Linked List (representasi dinamis)

Kedua represetasi tersebut memiliki keunggulan dan kelemahan, pemilihan metoda representasi bergantung karakteristik aplikasi.

Operasi-operasi dasar pada Stack S

1. Operasi menciptakan S sebagai stack kosong (Inisialisasi S → stack)
2. Operasi menyisipkan elemen x ke stack S untuk memperbaharui stack S (Push (S,x,t) → stack) dimana t adalah indeks puncak (top of stack atau top pointer)
3. Operasi menghilangkan elemen puncak stack S (Pop (S,x,t) → stack)
4. Operasi mengirimkan elemen puncak stack S tanpa memperbaharui/merubah stack S (Top(S,x,t) → item)
5. Operasi memeriksa jika stack S kosong (Empty (S) → boolean (true/false))
6. Operasi memeriksa jika stack S penuh (Full (S) → boolean (true/false))
7. Operasi menghitung jumlah elemen stack S (eNumber (S) → integer)

Implementasi operasi dasar tersebut juga dipengaruhi oleh metoda representasi, misalnya pada representasi dengan array maka operasi eNumber (S) adalah operasi sederhana dengan menghitung selisih indeks top of stack dengan indeks basis



Untuk indeks basis i = 1 dan top of stack i = t, maka eNumber (S) = t

2. Single Stack dengan Array

Representasi stack paling sederhana adalah menggunakan array satu dimensi :

1. Misalnya reporesentasi stack dengan array S [i], elemen pertama disimpan di S [1], elemen ke-2 di S [2] dan seterusnya elemen ke-i di S [i].
2. Untuk implementasi proses, dideklarasikan suatu variabel **t** yang menunjuk elemen puncak (terakhir) pada suatu saat.
3. Sesuai dengan sifat stack, pengambilan/penghapusan elemen dalam stack harus dimulai dari elemen teratas, dalam hal ini S [t].
4. Stack umumnya berukuran kecil/terbatas, karena umumnya lebih banyak digunakan sebagai tools dalam pengelolaan berbagai proses.

Contoh deklarasi Stack dengan Array :

```
Const   max  =  7
Type
      Tipedata      = integer
      Stack         = array [ 1.. max] of tipedata
Var
      t       :  integer
```

Operasi-operasi pada Single Stack :

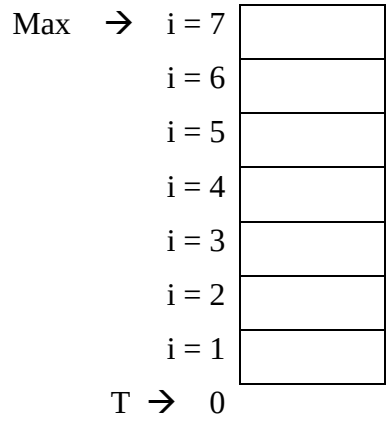
Create S

→ Membuat stack S baru yang masih kosong.

Secara fisik memory sudah dialokasikan ketika deklarasi stack dieksekusi (sesuai dengan sifat array sebagai variabel statis. Secara logika adalah dengan menginisialisasi variabel penunjuk elemen puncak stack (top pointer) t = 0.

```
Procedure Create;
   Begin
          t ← 0;
   end;
```

(Catatan : pada beberapa implementasi, t menunjuk elemen kosong setelah elemen terakhir, sehingga create stack direpresentasikan oleh inisialisasi t = 1)



Full (S)

➔ fungsi untuk memeriksa apakah stack S yang ada sudah penuh

Secara logika stack penuh jika pointer top bernilai sama dengan panjang maksimum stack.

```
Function Full : boolean

    Begin
            Full := false
            if  t = max then Full := true;
    End;
```

Empty (S)

➔ fungsi untuk memeriksa apakah stack kosong atau tidak

Secara logika stack kosong jika pointer top t = 0.

```
    Function Empty : boolean

    Begin
            Empty := false
            if  t = 0 then Empty := true;
    End;
```

Push (S,x,t)

➔ fungsi untuk menempatkan data **x** di stack S.

Operasi Push akan ditolak jika Stack S sudah penuh.

```
Procedure Push ( S : stack;  x : tipedata;  var  t : integer)
Begin
        If  not  Full then
        Begin
             t :=  t + 1
             S [ t ] :=  x
        End-if
End;
```

Pop (S,x,t)

➔ fungsi untuk mengambil elemen teratas dari stack S.

Operasi Pop (S,x,t) akan ditolak jika Stack S kosong

```
Procedure Pop ( S : stack; var x : tipedata;  var  t : integer)
Begin
        If  not  Full then
        Begin
             x  :=  S [ t ]
             t :=  t – 1
        end
End;
```

Clear (S, t)

➔ fungsi untuk mengosongkan seluruh elemen dari stack S.

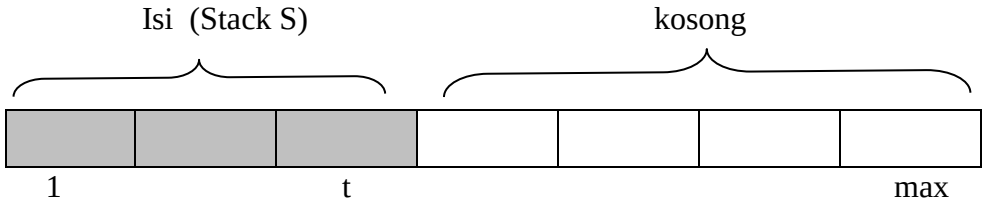
Konsep clear yang diterapkan adalah bahwa seluruh elemen yang berada diatas elemen yang ditunjuk oleh pointer **t** dianggap sudah hilang.

```
Procedure Clear ( S : stack;  var  t : integer)
Begin
        t  :=  0
End;
```

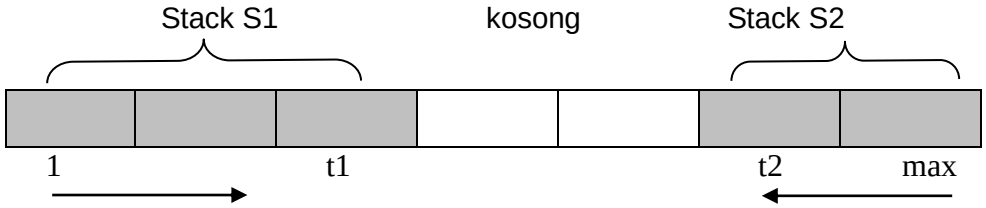
3. Multi Stack Dalam Satu Array

Stack paling sering dipakai dalam kasus manajemen sistem, untuk mendukung pengelolaan proses-proses. Setiap proses user (pemakai) membutuhkan dukungan beberapa stack. Untuk menghemat pemakaian memory, sebuah array dapat digunakan berkali-kali untuk beberapa stack.
Dan dimungkinkan dua buah atau lebih stack dapat diimplementasikan dengan satu array.

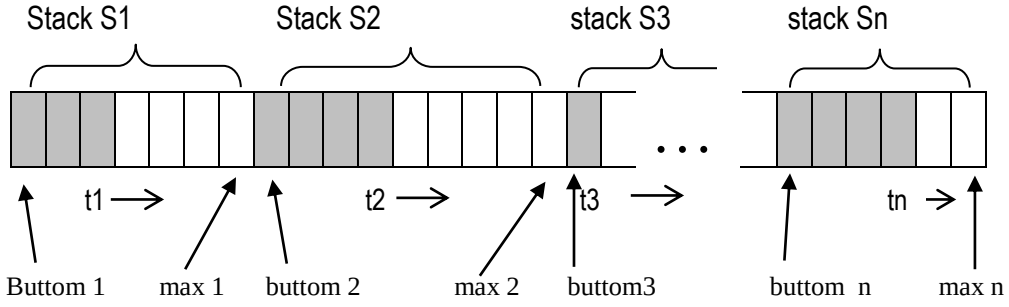
Satu stack dengan satu array :



Dua stack dalam satu array :



Beberapa (n) stack dalam satu array :



4. Operasi-operasi Dua Stack dengan satu Array

Operasi-operasi yang diterapkan pada implementasi dua stack dengan satu array, sama saja dengan operasi pada implementasi satu stack pada satu array, yang harus diperhatikan bahwa stack akan full jika jumlah elemen stack S1 dan stack S2 sama dengan *max*, dan operasi terhadap stack S2 dicacah secara terbalik.

Tujuan implementasi dua stack pada satu array adalah untuk efisiensi alokasi memory. Sehingga panjang array (max) yang dideklarasikan adalah total kebutuhan tertinggi yang paling sering dicapai oleh stack S1 dan stack S2, bukan didasarkan pada total kebutuhan maksimum stack S1 dan stack S2.

$$\textit{panjang array} \leq \textit{max(S1)} + \textit{max(S2)}$$

Contoh deklarasi dua Stack dengan satu Array :

```
Const   max  =  10
Type
      Tipedata     =  integer
      Stack        =  array [ 1.. max] of tipedata
Var
      t1, t2       :  integer
```

Operasi-operasi pada Double Stack :

a. Create S1 dan S2 (inisialisasi)

→ Membuat stack S1 dan S2 baru yang masih kosong.

```
Procedure Create;
 Begin
        t1 ← 0
        t2 ← max+1
 end;
```

Full (S)

➔ fungsi untuk memeriksa apakah stack S1 dan S2 yang ada sudah penuh

Secara logika stack penuh jika jumlah isi Stack S1 dan S2 sama dengan panjang array_S, yaitu:

top_1 + (max + 1 – top_2) = max

```
Function Full : boolean
    Begin
            Full := false
            if  (t1 + max + 1- t2) = max then Full := true;
    End;
```

EmptyS1 EmptyS2

➔ fungsi untuk memeriksa apakah stack kosong atau tidak, dalam hal ini harus diperiksa per stack

➔ Stack S1 kosong jika pointer top t1= 0, dan t2 = max+1 untuk stack S2

```
    Function EmptyS1 : boolean
    Begin
            EmptyS1 := false
            `if  t1 = 0 then EmptyS1 := true;
    End;


    Function EmptyS2 : boolean
    Begin
            EmptyS2 := false
            if  t2 = max+1 then EmptyS2 := true;
    End;
```

Push_S1 (S,x,t1) dan Push_S2 (S,x,t2)

Push_S1 ➔ fungsi untuk menempatkan data **x** di stack S1.
Push_S2 ➔ fungsi untuk menempatkan data **x** di stack S2.
Operasi Push_S1 atau Push_S2 akan ditolak jika array S sudah penuh.

```
Procedure Push_S1 ( S : stack;  x : tipedata;  var  t1 : integer)
Begin
        If  not  Full then
        Begin
             t1 :=  t1 + 1
             S [ t1 ] :=  x
        End-if
End;
```

```
Procedure Push_S2 ( S : stack;  x : tipedata;  var  t2 : integer)
Begin
        If  not  Full then
        Begin
             t2 :=  t2 – 1
             S [ t2 ] :=  x
        End-if
End;
```

Pop_S1 (S,x,t1) dan Pop_S2(S,x,t2)

Pop_S1 ➔ fungsi untuk mengambil elemen teratas dari stack S1.

Pop_S2 ➔ fungsi untuk mengambil elemen teratas dari stack S2.

Operasi Pop_S1 akan ditolak jika Stack S1 kosong, begitu juga dengan Stack S2

```
Procedure Pop_S1 ( S : stack; var x : tipedata;  var  t1 : integer)

Begin
        If  not  empty_S1 then
        Begin
             x  :=  S [ t1 ]
             t1 :=  t1 – 1
        end
End;
```

```
Procedure Pop_S2 ( S : stack; var x : tipedata;  var  t2 : integer)

Begin
        If  not  empty_S2  then
        Begin
             x  :=  S [ t2 ]
             t2 :=  t2 + 1
        end
End;
```

Clear (S, t1, t2)

➔ fungsi untuk mengosongkan seluruh elemen dari stack S.

Konsep clear yang diterapkan adalah bahwa seluruh elemen yang berada diatas elemen yang ditunjuk oleh pointer **t1** dan yang berada dibawah elemen yang ditunjuk pointer **t2** dianggap sudah hilang.

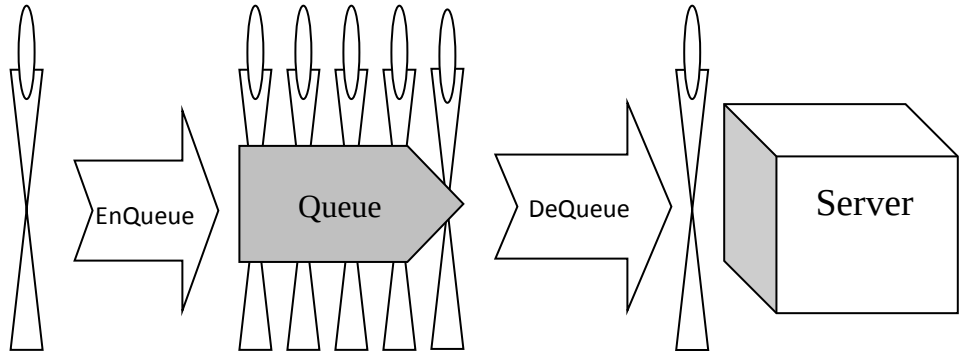
```
Procedure Clear ( S : stack;  var  t1,t2 : integer)

Begin
        t1 ← 0
        t2 ← max+1
End;
```

Q U E U E
(A N T R I A N)

1. Definisi

Queue jika diartikan secara harafiah berarti antrian. Konsep utamanya adalah FIFO atau FIFS (First In First Out atau First In First Serve). Artinya elemen yang datang terlebih dulu akan dilayani lebih dulu.

Contoh antrian dalam bentuk nyata sering kita temui dalam kehidupan sehari-hari, misalnya pada saat anda mengantri di kasir sebuah supermarket atau antri layanan kassa bank.



Tail Front

EnQueue adalah istilah yang sering dipakai apabila seseorang masuk kedalam sebuah antrian. Dalam antrian yang datang terlebih dahulu akan dilayani terlebih dahulu. Seperti diilustrasikan pada gambar diatas. Seseorang yang keluar dari antrian untuk mendapatkan pelayanan sering disebut DeQueue.
Dalam implementasi struktur data, queue adalah struktur data dengan penyisipan di satu ujung dan penghapusan di ujung lainnya. Ujung penyisipan disebut Rear/Tail/ Ekor sedangkan ujung penghapusan disebut Front/Head/Depan.
Ketika elemen terdepan (pertama) dilayani, maka secara logika terjadi pergeseran posisi elemen-elemen dibelakangnya satu langkah, yaitu :

- elemen ke dua akan berpindah ke posisi/menjadi elemen pertama
- elemen ke tiga elemen akan berpindah ke posisi/menjadi elemen ke dua
- elemen ke empat elemen akan berpindah ke posisi/menjadi elemen ke tiga
  …
- elemen ke (n) elemen akan berpindah ke posisi/menjadi elemen ke (n-1)

2. Karakteristik Penting

Hal-hal/karakteristik yang penting dalam pengelolaan antrian adalah :

1. Elemen antrian :
   Item-item data yang terdapat/diorganisasikan pada antrian
2. Front
   Elemen terdepan dari antrian
3. Tail
   Elemen terakhir dari antrian
4. Panjang Antrian
   Yaitu banyaknya (jumlah) elemen pada antrian, ada dua jenis panjang:
   - panjang aktual :
     merepresentasikan banyak item data pada antrian pada suatu saat.
   - Kapasitas antrian :
     panjang maksimum yang diizinkan pada antrian
5. Status

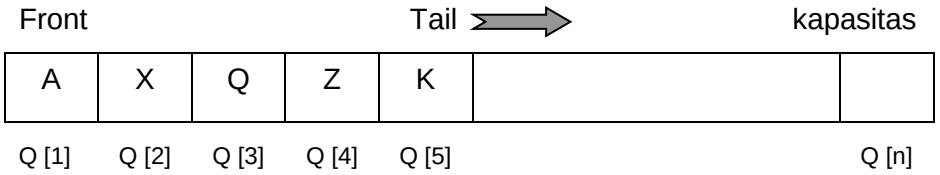
   Merepresentasikan kondisi antrian pada satu saat, terdapat 3 jenis status pada antrian, yaitu :
   - Kosong
     Bila tidak ada elemen pada antrian, pada kondisi ini tidak mungkin dilakukan operasi DeQueue terhadap antrian. DeQueue pada antrian/queue kosong menyebabkan kondisi keselahan Underflow
   - Isi tapi tidak penuh
     Bila paling sedikit terdapat satu elemen pada antrian dan paling banyak (n-1) elemen, dimana n adalah kapasitas antrian.
   - Penuh
     Bila jumlah elemen pada antrian mencapai kapasitas antrian. Pada kondisi ini tidak mungkin dilakukan operasi EnQueue terhadap antrian. Penambahan elemen baru (EnQueue) menyebabkan kondidi overflow.

Untuk implementasi struktur data antrian pada program dapat dilakukan dengan 2 cara, yaitu :

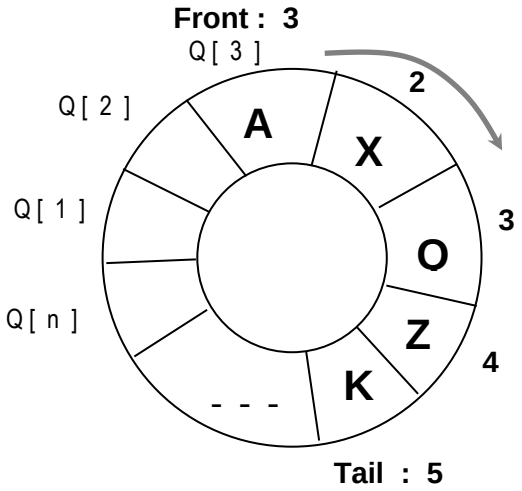
1. Array
2. Linked List

3. Queue dengan Array

a. Representasi Statik Linear



b. Representasi Statik Circular



Pada representasi Circular, indeks array untuk elemen terdepan (front) bersifat dinamis, tidak harus berada pada elemen pertama (Q [1]) dari array tapi bergeser sesuai operasi DeQueue yang diterimanya.

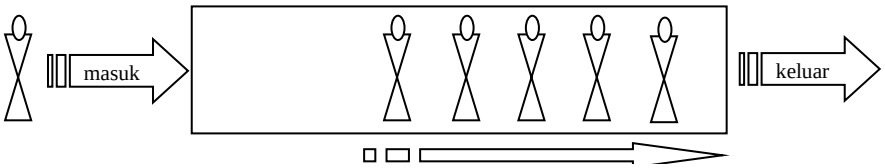
4. Operasi Dasar

Terhadap suatu Queue berlaku operasi-operasi dasar sebagai berikut :

1) Create : menciptakan suatu antrian kosong
2) CekStatus : memeriksa status antrian pada suatu saat, ada dua kondisi penting pada CekStatus : Empty dan Full
3) EnQueue : menyisipkan sebuah elemen baru pada antrian
4) DeQueue : menghapus/mengeluarkan sebuah elemen dari antrian
5) Clear : mengosongkan antrian

5. Representasi Statik Linear Queue Dengan Array

Representasti Queue secara linear (sequens) lebih rumit dibandingkan Stack. Pada representasi linear, array yang digunakan seakan-akan merupakan sebuah lorong lurus dengan satu masuk dan satu pintu keluar. Didalam lorong elemen data bergerak selangkah demi selangkah sesuai dengan operasi DeQueue yang dilakukan pada pintu keluar.



Setiap ada elemen yang keluar antrian, maka elemen yang dibelakangnya harus maju satu langkah (posisi), sehingga elemen ke-2 menempati posisi pertama dst.

Berikut ini adalah contoh deklarasi data untuk suatu Queue :

```
Const
    MaxQueue =   10

Type
    TipeData   = integer;

Var
    Queue          :  array [ 1 ..  MaxQueue ] of TipeData
    Front, Tail, j :  integer
```

Konvensi :

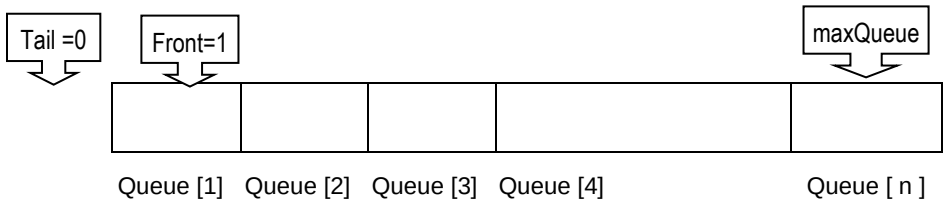
1. Front selalu menunjuk posisi Queue [1]
2. Tail menujuk elemen terakhir pada Queue, sehingga
   → Tail ≤ MaxQueue
3. Queue adalah kosong jika Tail lebih kecil dari Front

Operasi-operasi pada Queue dengan Linear Array

1) Create

Procedure Create berguna untuk menciptakan Queue baru dan Kosong. Pada implementasi Queue dengan Linear Array, pada dasarnya sudah dilakukan alokasi memory ketika struktur data Queue dideklarasikan (deklarasi type dan variabel). Sehingga pengertiaan penciptaan Queue baru adalah melakukan inisialisasi variabel Front dengan 1 dan Tail dengan 0.

```
Procedure  Create;
Begin
        Front :=  1;
        Tail  :=  0;
End;
```



2) CekStatus

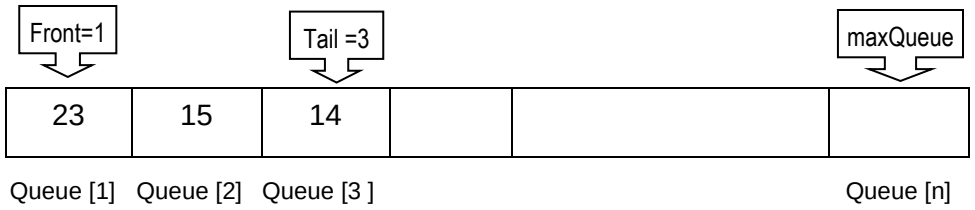
Operasi CekStatus berguna untuk memeriksa apakah Queue :

- Kosong / empty, jika Tail = 0, Function Empty
- Penuh / Full, jika Tail = MaxQueue Function Full

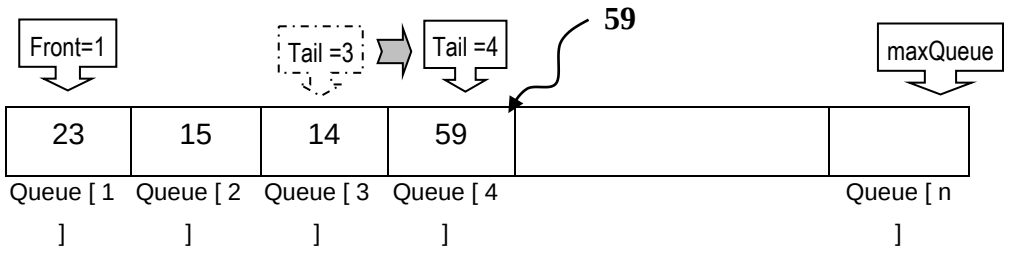
```
Function  Empty : Boolean ;
Begin
     If Tail = 0
     then  Empty := true
      else  Empty  :=  false
end;

Function  Full : Boolean ;
Begin
     If Tail  = MaxQueue
     then  Full := true
      else  Full  :=  false
end;
```

3) EnQueue

Operasi EnQueue berguna untuk memasukkan sebuah elemen baru ke antrian. Operasi ini hanya legal jika Queue tidak Full.



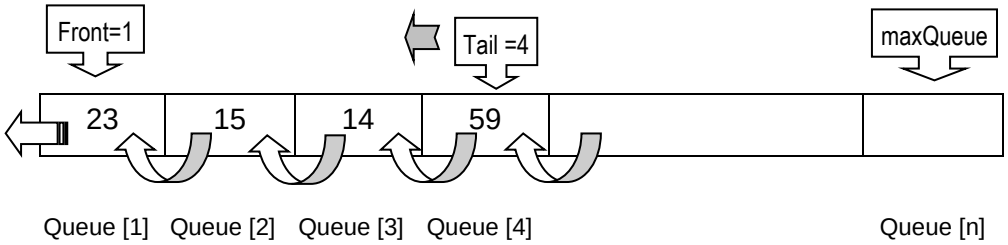
Contoh : ***EnQueue (59)***

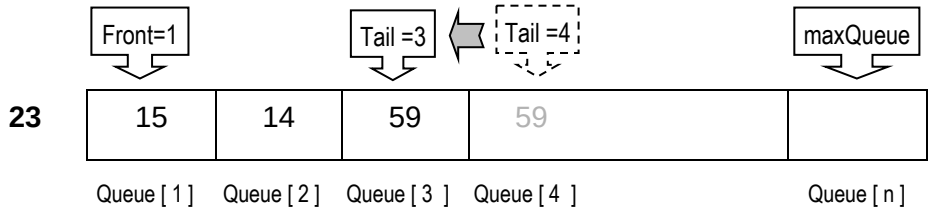



```
Procedure EnQueue ( Databaru:TipeData )
Begin
    If  not Full  then
    begin
        Tail := Tail + 1
        Queue [ Tail ] :=  DataBaru
    End;
End;
```

4) DeQueue

Operasi DeQueue berguna untuk mengambil sebuah elemen dari antrian. Dilakukan terhadap elemen terdepan (front). Instruksi pengambilan harus diikuti dengan instruksi pengeseran elemen-elemen dibelakangnya 1 langkah. Operasi ini legal jika Queue tidak Empty.






```
Procedure DeQueue ( DataServe :TipeData )
Begin
    If not Empty then
    begin
        DataServe := Queue [ Front ]
        For i := 1 to Tail - 1
          Do
              Queue [ i ] := Queue [ i + 1 ]
        Tail := Tail – 1
    End;
End;
```

5) Clear

Mengosongkan antrian (Queue) pada implementasi dengan linear array adalah menginisialisasi ulang variabel Tail dengan 0.

```
Procedure Clear ;
  Begin
       Tail := 0;
End;
```

6) Panjang Antrian

Panjang antrian (Queue) pada linear array sama dengan selisih Tail terhadap Front ditambah 1. Pada kasus Front statis (= 1), maka panjang antrian sama dengan Tail.

PanjangQueue = Tail – Front + 1

Pada kasus Front statis

=> *Front = 1*

sehingga :

PanjangQueue = Tail