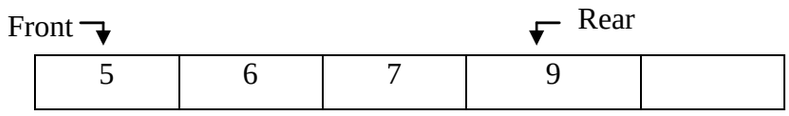
QUEUE (ANTRIAN)

Queue (antrian) adalah barisan elemen yang apabila elemen ditambah maka penambahannya berada di posisi **belakang** (rear) dan jika dilakukan pengambilan elemen dilakukan di elemen paling **depan** (front). Oleh karena itu, queue bersifat FIFO (first in first out).

Contoh :



Operasi-operasi dasar dari sebuah queue adalah :

1. Enqueue : proses penambahan atau memasukkan satu elemen di belakang
2. Dequeue : proses pengambilan atau mengeluarkan satu elemen di posisi depan

Representasi Queue :

1. Menggunakan Array Statis

a. **Deklarasi** secara umum :

```
Const
    MaksQueue =…….
Type
    TypeQueue = array [1..MaksQueue] of typedata

VarQueue : TypeQueue
Front, Rear : integer
```

b. Penciptaan Queue (Create queue)

Proses pemberian nilai 0 untuk variabel penunjuk depan(Front) dan variabel penunjuk belakang(Rear) dari queue, yaitu:

Front ←0

Rear ←0

c. Fungsi Kosong

Fungsi kosong digunakan untuk memeriksa apakah keadaan queue tidak memiliki elemen. Fungsi kosong didapatkan dengan memeriksa penunjuk Rear dari queue. Jika penunjuk Rear bernilai nol (0), maka berarti queue kosong dan jika tidak nol, maka berarti queue mempunyai elemen.

```
Function  Kosong (Input Rear : integer) → Boolean
{I.S : penunjuk Rear pada queue sudah terdefinisi}
{F.S : menghasilkan fungsi kosong}
Kamus:

Algoritma :
        Kosong←false
        If  (Rear =0) then
            Kosong←True
        EndIf
EndFunction
```

d. Fungsi Penuh

Fungsi penuh berguna untuk memeriksa apakah suatu queue telah penuh atau belum. Fungsi ini diperlukan ketika proses enqueue. Fungsi ini akan bernilai benar (true) jika penunjuk Rear sama dengan nilai Maksimum Queue jika tidak sama berarti queue belum penuh.

```
Function Penuh (Input  Rear : Integer) → Boolean
{I.S : penunjuk Rear pada queue sudah terdefinisi}
{F.S : menghasilkan fungsi penuh}
Kamus:

Algoritma :
        Penuh←false
        If  (Rear =MaksQueue) Then
            Penuh←True
        EndIf
EndFunction
```

e. Proses enqueue (dengan memeriksa Queue kosong/tidak dan Queue penuh/tidak)

- Proses enqueue adalah proses untuk penambahan di posisi Rear.
- Penambahan ini dilakukan jika kondisi queue tidak penuh.
- Jika keadaan masih kosong, maka posisi Front dan Rear bernilai 1 tetapi jika sudah mempunyai elemen, maka nilai Rear harus bertambah 1.
- Kemudian data baru disimpan di queue pada posisi Rear.

```
Procedure  Enqueue (I/O Front,Rear : integer, Input elemen : tipedata)
{I.S : penunjuk Front dan Rear serta data yang akan dimasukkan ke
       queue (elemen) sudah terdefinisi}
{F.S : menghasilkan queue yang sudah bertambah satu data}
Kamus:

Algoritma :
   If  (Kosong(Rear))  Then
       Front←1
       Rear←1
   Else
       If  (Not Penuh(Rear)) Then
           Rear ← Rear + 1
       Endif
   EndIf
   Queue(Rear) ←  elemen
EndProcedure
```

f. Proses Dequeue

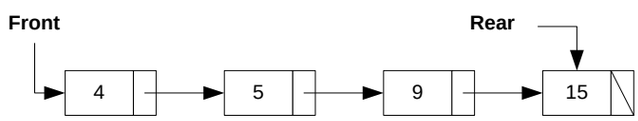
- Operasi dequeue adalah proses pengambilan satu elemen dari queue. Tentunya elemen yang diambil selalu dari elemen pertama (1).
- Setelah elemen pertama diambil, maka akan terjadi proses pergeseran elemen data setelah elemen data yang diambil (dari posisi ke-2 sampai posisi paling belakang), dan kemudian posisi Rear akan berkurang 1 karena ada data yang diambil.

```
Procedure  Dequeue (I/O Front,Rear : integer, Output elemen : tipedata)
{I.S : penunjuk Front dan Rear sudah terdefinisi}
{F.S : menghasilkan queue yang sudah berkurang satu data}
Kamus:
        i : integer
Algoritma :
        If  (Not  Kosong(Rear)) Then
            elemen ← Queue(Front)
            For  i ←  Front  to  (Rear-1) do
                 Queue(i)← Queue(i+1)
            Endfor
            Rear ← Rear - 1
        EndIf
EndProcedure
```

2. Menggunakan Array Dinamis (Single Linked List)

Proses penyimpanan elemen queue dalam linked list mirip dengan operasi pada single linked list yang menggunakan penyisipan di akhir/belakang dan penghapusan di awal/depan.

Contoh :



Operasi-operasi yang dapat dilakukan dalam queue yang menggunakan representasi linked list adalah :

2.1. Pendeklarasian sebuah queue

Setiap queue memiliki penunjuk berupa pointer Front, pointer Rear, elemen antrian, dan pointer penunjuk ke elemen berikutnya. Berikut ini adalah pendeklarasian queue yang disimpan dalam bentuk linked list.

```
Type
   NamaPointer = ↑Simpul
   Simpul = Record
        Info : Tipedata
        Next : NamaPointer
   EndRecord

    Front, Rear : NamaPointer
```

2.2. Penciptaan Queue

Proses inisialisasi queue yang disimpan dalam bentuk linked list adalah dengan memberikan nilai **nil** ke pointer Front dan Rear yang menandakan bahwa queue masih kosong.

```
Front ← Nil
Rear  ← Nil
```

2.3. Fungsi Kosong

Fungsi ini berguna untuk memeriksa apakah suatu queue dalam keadaan kosong. Fungsi ini berguna ketika proses dequeue yaitu ketika sebuah elemen akan diambil, maka harus diperiksa dulu apakah memiliki data atau tidak. Fungsi ini akan mempunyai nilai benar jika Front atau Rear bernilai nil.

```
Function Kosong(Input  Front:NamaPointer) → Boolean
{I.S : penunjuk Front pada queue sudah terdefinisi}
{F.S : menghasilkan fungsi kosong}
Kamus:

Algoritma :
        Kosong←false
        If  (Front = Nil) Then
            Kosong←True
        EndIf
EndFunction
```

2.4. Operasi Enqueue

- Proses enqueue dalam queue linked list adalah dengan menambahkan elemen baru ke posisi paling belakang.
- Setelah itu, pointer penunjuk Rear harus berpindah ke posisi baru tersebut.
- Proses ini seperti proses penyisipan di belakang/akhir pada single linked list.

Procedure Enqueue (I/O Front,Rear:NamaPointer, Input elemen:tipedata)
{I.S : penunjuk Front dan Rear serta data yang akan dimasukkan ke queue (elemen) sudah terdefinisi}
{F.S : menghasilkan queue yang sudah bertambah satu data}
Kamus:
Baru : NamaPointer
Algoritma :
Alloc (Baru)
Baru↑.Info ← elemen
If (Kosong(Front)) Then
Front←Baru
Else
Rear↑.Next←Baru
Endif
Rear←Baru
Baru↑.Next←Nil
EndProcedure

2.5. Operasi Dequeue

- Proses dequeue untuk queue linked list adalah dengan mengambil data yang ditunjuk pointer Front.
- Pointer Front harus berpindah ke elemen antrian berikutnya. Proses tersebut dilakukan hanya jika linked list tidak kosong. Proses ini mirip dengan proses penghapusan di awal/depan pada single linked list.

```
Procedure Dequeue(I/O Front,Rear : NamaPointer, Output elemen : tipedata)
{I.S  : penunjuk Front dan Rear sudah terdefinisi}
{F.S : menghasilkan queue yang sudah berkurang satu data}
Kamus:
        Phapus : NamaPointer
Algoritma :
        If  (Not  Kosong(Rear)) Then
          Phapus ← Front
          Elemen ← Front↑.Info
          If (Front = Rear) Then
                Front ← Nil
                Rear ← Nil
          Else
                Front←Front↑.Next
          EndIf
          Dealloc(Phapus)
        Endif
EndProcedure
```

QUEUE DENGAN CIRCULAR ARRAY

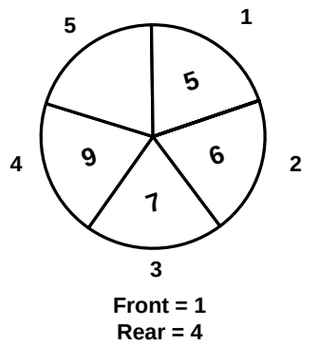
Jika kita menggunakan array untuk queue seperti di atas, maka ketika ada proses pengambilan (dequeue) ada proses pergeseran data. Proses pergeseran data ini pasti memerlukan waktu apalagi jika elemen queue-nya banyak. Oleh karena itu solusi agar proses pergeseran dihilangkan adalah dengan metode circular array.

Queue dengan circular array dapat dibayangkan sebagai berikut :

Front = 1 Rear = 4

5	6	7	9	

Atau agar lebih jelas, array di atas dapat dibayangkan ke dalam bentuk seperti lingkaran di bawah ini.



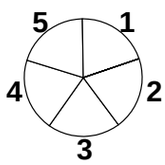
Aturan-aturan dalam queue yang menggunakan circular array adalah :

1. Proses penghapusan dilakukan dengan cara nilai depan (Front) ditambah 1 :
   Front = Front + 1.
2. Proses penambahan elemen sama dengan linear array yaitu nilai belakang(Rear) ditambah 1 : Rear = Rear + 1.
3. Jika Front = MaksQueue dan ada elemen yang akan dihapus, maka nilai Front = 1.
4. Jika Rear = MaksQueue dan Front tidak 1 maka jika ada elemen yang akan ditambahkan, nilai belakang=1
5. Jika hanya tinggal 1 elemen di queue (Front = Rear), dan akan dihapus, maka Front diisi 0 dan Rear diisi dengan 0 (queue kosong).

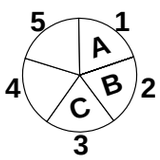
Contoh :

Untuk mempermudah, elemen dari queue bertipe karakter.

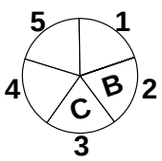
No	Operasi	Status	Queue				
			1	2	3	4	5
1.	Inisialisasi	Front = 0 Rear = 0					
2.	Enqueue A, B, C	Front = 1 Rear = 3	**A**	**B**	**C**		
3.	Dequeue	Front = 2 Rear = 3		**B**	**C**		
4.	Enqueue D, E	Front = 2 Rear = 5		**B**	**C**	**D**	**E**
5.	Dequeue 2 kali	Front = 4 Rear = 5				**D**	**E**
6.	Enqueue F	Front = 4 Rear = 1	**F**			**D**	**E**
7.	Enqueue G, H	Front = 4 Rear = 3	**F**	**G**	**H**	**D**	**E**
8.	Dequeue	Front = 5 Rear = 4	**F**	**G**	**H**		**E**
9.	Dequeue	Front = 1 Rear = 3	**F**	**G**	**H**		
10.	Dequeue 2 kali	Front = 3 Rear = 3			**H**		
11.	Dequeue	Front = 0 Rear = 0					



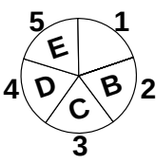
Operasi 1



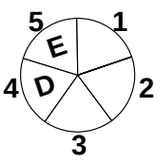
Operasi 2



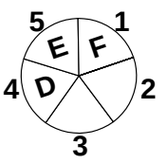
Operasi 3



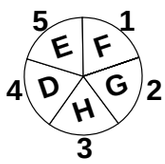
Operasi 4



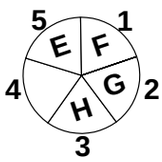
Operasi 5



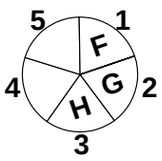
Operasi 6



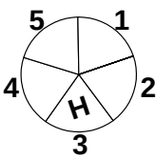
Operasi 7



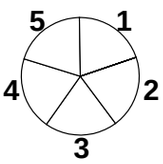
Operasi 8



Operasi 9



Operasi 10



Operasi 11

Operasi-operasi yang dapat terjadi dalam queue yang menggunakan circular array adalah :

1. **Penciptaan queue**

   Penciptaan queue adalah proses pemberian nilai 0 untuk penunjuk Front dan penunjuk Rear dari queue

   Front ← 0

   Rear ← 0

2. **Fungsi Kosong**

   Suatu queue yang menggunakan circular array dapat dikatakan kosong jika nilai dari posisi Front atau Rear mempunyai nilai 0.

```
Function Kosong(Input Front : Integer) → Boolean
{I.S. : Penunjuk Front sudah terdefinisi}
{F.S. : Menghasilkan fungsi kosong}
Kamus:

Algoritma :
    Kosong ← false
    If (Front = 0) Then
        Kosong← true
    EndIf
EndFunction
```

3. **Fungsi Penuh**

   Suatu queue akan disebut penuh jika terjadi 2 hal yaitu

   - Jika Front ada diposisi 1 dan Rear ada diposisi MaksQueue
   - Atau jika Front bernilai sama dengan posisi Rear +1.

```
Function Penuh (Input Front, Rear : Integer) → Boolean
{I.S. : Penunjuk Front dan Rear sudah terdefinisi}
{F.S. : Menghasilkan fungsi penuh}
Kamus:

Algoritma :
    Penuh←False
    If ((Front = 1) and (Rear = maks)) or (Front = Rear+1) Then
        Penuh←True
    EndIf
EndFunction
```

5. Operasi Enqueue

Proses enqueue hanya bisa dilakukan jika queue tidak kosong. Ada beberapa hal yang harus diperhatikan ketika akan melakukan enqueue pada circular array, diantaranya adalah :

- Kondisi ketika queue masih kosong. Jika ini terjadi, maka posisi Front dan Rear bernilai 0.
- Ketika nilai Rear sama dengan MaksQueue, maka posisi Rear bernilai 1
- Ketika nilai Rear masih lebih kecil dari MaksQueue, maka posisi Rear ditambah 1 : Rear = Rear + 1

```
Procedure Enqueue (Input elemen : tipedata, I/O Front,Rear : Integer)
{I.S.  : Data yang akan dimasukkan (elemen), penunjuk Front dan penunjuk
         Rear sudah terdefinisi}
{F.S. : Menghasilkan Queue yang sudah bertambah satu data}
Kamus:

Algoritma :
    If (Kosong(Front)) Then
        Front ← 1
        Rear  ← 1
    Else
    If  (Not Penuh(Front,Rear))  Then
        If  (Rear = MaksQueue)  Then
            Rear ← 1
        Else
            Rear ← Rear + 1
        EndIf
        Queue(Rear) ← elemen
   EndIf
EndProcedure
```

4. Operasi Dequeue

Proses dequeue hanya bisa dilakukan jika queue dalam keadaan tidak kosong. Ada beberapa kondisi yang harus diperhatikan ketika dequeue elemen queue yaitu :

- Kondisi ketika posisi Front sama dengan posisi Rear (queue hanya memiliki 1 elemen), maka nilai Front dan Rear bernilai 0 (queue kosong).
- Jika posisi Front sama dengan MaksQueue maka posisi Rear menjadi 1.
- Selain itu, posisi Front ditambah dengan 1 : Front = Front + 1

```
Procedure  Dequeue (Output elemen : tipedata, I/O Front, Rear : Integer)
{I.S.  : Penunjuk Front dan penunjuk Rear sudah terdefinisi}
{F.S. : Menghasilkan Queue yang sudah berkurang satu data}
Kamus:

Algoritma :
    If  (Not Kosong(Front))  Then
       If  (Front = Rear)  Then
             Front←0
             Rear← 0
       Else
         If  (Front = MaksQueue)  Then
              Front ← 1
         Else
              Front ← Front + 1
         EndIf
    EndIf
EndProcedure
```