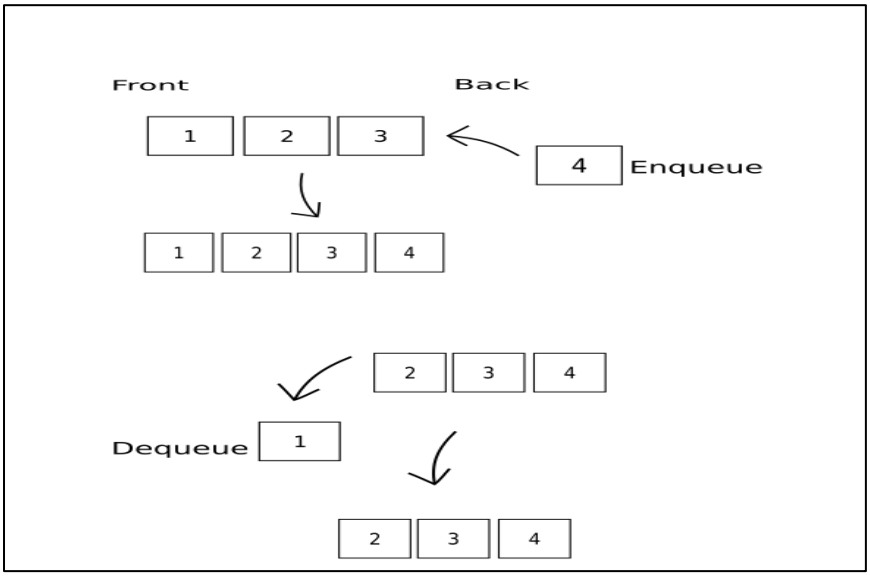
**QUEUE**

Using:

a) **Single Linked List**
b) **Array Alternative 1**
c) **Array Alternative 2**
d) **Array Alternative 3**

## a. Single Linked List




Pertama buat kelas Node, yang dengannya kita dapat membuat elemen dengan data tertentu untuk diantrekan dalam Antrean. Di kelas Node, kita akan mengatur data dan next_node (yaitu pointer) sama dengan None sebagai parameter dalam metode init sehingga jika kita tidak mengirim data dan pointer ke node berikutnya, ia akan mengembalikan None. Kemudian metode get_data dan get_next akan mengembalikan data dan node berikutnya masing-masing. Kami akan menambahkan metode lain set_next yang akan mengambil new_node sebagai parameter dan akan mengatur pointer dari Node sebelumnya menuju Node ini.

Setelah itu buat kelas lain bernama Queue, dan atur head (elemen 1st Queue) = None sebagai parameter dalam metode init. Kemudian kita akan memasukkan elemen pada Queue pertama menggunakan metode enqueue. Metode enqueue mengambil data sebagai parameter dan membuat Node menggunakannya. Sekarang, kami akan memeriksa apakah ada kepala di Antrian! Jika belum ada, kita setel elemen Node (new_item) / yang dibuat sebelumnya sebagai head. Jika sudah ada head yaitu ada elemen di Queue. Jadi, kita akan pergi ke akhir Queue menggunakan while loop dan ketika kita menemukan bahwa pointer dari node sebelumnya adalah null, kita mengatur

pointer dari Node / elemen terakhir ke Node yang kita buat. Sekarang saatnya menambahkan metode dequeue untuk menghapus elemen pertama dari Queue. Pertama kita cek apakah Queue tersebut kosong atau tidak, jika tidak kosong maka kita setel elemen kedua Queue sebagai head dan elemen pertama (self.head) yang semula disimpan pada saat ini akan dihapus. Sekali lagi, jika Antrian kosong, kita cukup mencetak "Antrian kosong.".

**a. Array Alternative 1**

Sebelum melakukan fungsi dibawah, buat terlebih dahulu fungsi antrian

```
Procedure CreateQueue( Q : queue )

  Kamus

  Algoritma

        Q.head ← -1 Q.tail ← -1
```

1. IsEmpty()

Untuk memeriksa apakah Antrian sudah penuh atau belum Dengan cara memeriksa nilai Tail, jika Tail = -1 maka empty Kita tidak memeriksa Head, karena Head adalah tanda untuk kepala antrian (elemen pertama dalam antrian) yang tidak akan berubah-ubah Pergerakan pada Antrian terjadi dengan penambahan elemen Antrian kebelakang, yaitu menggunakan nilai Tail.

```
Function IsEmptyAlt1( Q : Queue ) → boolean

 Kamus

 Algoritma

  → ( Q.head = -1 and Q.tail = -1 )
```

2. IsFull()

Untuk mengecek apakah Antrian sudah penuh atau belum Dengan cara mengecek nilai Tail, jika Tail >= MAX-1 (karena MAX-1 adalah batas elemen array pada C) berarti sudah penuh.

```
Function IsFullAlt1( Q : Queue ) → boolean

 Kamus

 Algoritma

  → (Q.head = 0 and Q.tail = NMax-1 )
```

3. EnQueue()

Untuk menambahkan elemen ke dalam Antrian, penambahan elemen selalu ditambahkan di elemen paling belakang Penambahan elemen selalu menggerakan variabel Tail dengan cara increment counter Tail terlebih dahulu.

```
Procedure EnQueueAlt1( input/output Q:Queue, input X:integer )
{
  IS. Queue mungkin kosong atau penuh, dan P berisi data
  FS. X di-enqueue kedalam Queue dengan Alternative 1, tampilkan penuh apabila queue penuh }
  Kamus
  Algoritma
    if (Q.head = 0 and Q.tail = NMax-1 ) then       // queue penuh
      output('penuh')
    else
      if ( Q.head = -1 and Q.tail = -1 ) then
        Q.head ← Q.head + 1                  // queue kosong, head nambah 1
      {end if}
      Q.tail ← Q.tail + 1                    // queue kosong dan tidak kosong, tail nambah 1
      Q.info [Q.tail] ← X                    // data di tambahkan
    {end if}
```

4. DeQueue()

Digunakan untuk menghapus elemen terdepan/pertama (head) dari Antrian Dengan cara menggeser semua elemen antrian kedepan dan mengurangi Tail dgn 1 Penggeseran dilakukan dengan menggunakan looping.

```
Procedure DeQueueAlt1( input/output Q: queue)
{ IS. Queue mungkin kosong atau penuh
  FS. Queue di-dequeue dengan Alternative 1 }
  Kamus
       i : integer
  Algoritma
     if ( Q.head = -1 and Q.tail = -1 ) then       // kondisi kosong
        output('Stack kosong')
     else                                  // tidak kosong
        i ← 0                               // yang ngantri dimajukan satu langkah kedepan
        while ( i < Q.Tail ) do
           Q.info[i] ← Q[i+1]
           i++
        {end while}
        Q.tail ← Q.tail - 1                 // update tailnya, yaitu PASTI berkurang 1
        if ( Q.Tail = -1 ) then
          Q.Head -1
         {end if}
        {end if}
```

**b. Array Alternative 2**

Sebelum melakukan fungsi dibawah, buat terlebih dahulu fungsi antrian

```
Procedure CreateQueue( Q : queue )

  Kamus

  Algoritma

      Q.head ← -1 Q.tail ← -1
```

1. IsEmpty()

Untuk memeriksa apakah Antrian sudah penuh atau belum Dengan cara memeriksa nilai Tail, jika Tail = -1 maka empty Kita tidak memeriksa Head, karena Head adalah tanda untuk kepala antrian (elemen pertama dalam antrian) yang tidak akan berubah-ubah Pergerakan pada Antrian terjadi dengan penambahan elemen Antrian kebelakang, yaitu menggunakan nilai Tail.

```
Function IsEmptyAlt2( Q:Queue ) → boolean

Kamus

Alghoritma

  → ( Q.head = -1 and Q.tail = -1 )
```

2. IsFull()

Untuk mengecek apakah Antrian sudah penuh atau belum Dengan cara mengecek nilai Tail, jika Tail >= MAX-1 (karena MAX-1 adalah batas elemen array pada C) berarti sudah penuh.

```
Function IsFullAlt2( Q:Queue ) → boolean

Kamus

Algoritma

  → ( Q.head = 0 and Q.tail = NMax-1 )
```

3. EnQueue()

Untuk menambahkan elemen ke dalam Antrian, penambahan elemen selalu ditambahkan di elemen paling belakang Penambahan elemen selalu menggerakan variabel Tail dengan cara increment counter Tail terlebih dahulu.

```
Procedure EnQueueAlt2(input/output Q: Queue, input P : infotype)
{ IS. Queue mungkin kosong atau penuh, dan P berisi data
  FS. P di-enqueue kedalam Queue, dengan Alternative 2 }
Kamus
        i,j : integer
Algoritma
        if ( Q.head = 0 and Q.tail = NMax-1 ) then   // Queue Penuh
                output ('Queue Penuh')
        else  if ( Q.head = -1 and Q.Tail = -1 ) then // Queue Kosong
                        Q.head ← Q.head + 1
                        Q.tail ← Q.tail + 1
                        Q.info[Q.tail] <- P
        else if ( Q.tail = NMax-1 ) then   // Kondisi Khusus, jika tail ada di ujung kanan, dan tidak penuh
                        i ← Q.head      // maka seluruh antrian dimajukan ke posisi paling depan semua
                        j ← 0
```

4. DeQueue()

Digunakan untuk menghapus elemen terdepan/pertama (head) dari Antrian Dengan cara menggeser semua elemen antrian kedepan dan mengurangi Tail dgn 1 Penggeseran dilakukan dengan menggunakan looping.

**Procedure DeQueueAlt2(input/output Q : queue)**

{ **IS**. Queue mungkin kosong atau penuh

**FS**. Queue di-dequeue , dengan Alternative 2 }

**Kamus**

**Algoritma**

```
        if ( Q.head = -1 and Q.Tail = -1 ) then      //queue kosong
                output('Stack kosong')
        else                                         // queue tidak kosong
           p <- Q.info[q.head]                                  // data yang di head disimpan di p
           if ( Q.head = Q.tail ) then     // jika 1 elemen dan posisi dimanapun dan tidak kosong
              Q.head <- -1            //maka head dan tail pasti jadi -1 semua} Q.tail   -1
           else                       // kondisi biasa, headnya nambah 1
             Q.head ++
          {end if}
          {end if}
```

**c. Array Alternative 3**

Sebelum melakukan fungsi dibawah, buat terlebih dahulu fungsi antrian

```
Procedure CreateQueue( Q : queue )

  Kamus

  Algoritma

      Q.head ← -1 Q.tail ← -1
```

1. IsEmpty()

Untuk memeriksa apakah Antrian sudah penuh atau belum Dengan cara memeriksa nilai Tail, jika Tail = -1 maka empty Kita tidak memeriksa Head, karena Head adalah tanda untuk kepala antrian (elemen pertama dalam antrian) yang tidak akan berubah-ubah Pergerakan pada Antrian terjadi dengan penambahan elemen Antrian kebelakang, yaitu menggunakan nilai Tail.

```
Function IsEmptyAlt3( Q:Queue ) → boolean

Kamus

Alghoritma

  → ( Q.head = -1 and Q.tail = -1 )
```

2. IsFull()

Untuk mengecek apakah Antrian sudah penuh atau belum Dengan cara mengecek nilai Tail, jika Tail >= MAX-1 (karena MAX-1 adalah batas elemen array pada C) berarti sudah penuh.

```
Function IsFullAlt3( Q:Queue ) → boolean

Kamus

Algoritma

  → (Q.head = 0 and Q.Tail = NMax-1) or (Q.head = ( Q.tail + 1))
```

3. EnQueue()

Untuk menambahkan elemen ke dalam Antrian, penambahan elemen selalu ditambahkan di elemen paling belakang Penambahan elemen selalu menggerakan variabel Tail dengan cara increment counter Tail terlebih dahulu.

**Procedure EnQueueAlt3(input/output Q:queue, input p : infotype)**

{ **IS**. Queue mungkin kosong atau penuh, dan P berisi data

**FS**. P di-enqueue kedalam Queue, dengan Alternative 3}

**Kamus**

**Algoritma**

**if (Q.head = 0 and Q.Tail = NMax-1) or (Q.head = ( Q.tail + 1)) then** // Queue penuh

output('Queue penuh')

**else if ( Q.head = -1 and Q.Tail = -1 ) then** // Queue kosong

**Q.head ← Q.head + 1**

**Q.tail ← Q.tail + 1**

**if ( Q.tail = NMax-1 ) then** // Kondisi khusus, tail sudah di ujung kanan dan tidak penuh

**Q.tail ← 0**

**else** // Kondisi biasa

**Q.tail ← Q.tail + 1**

{**end if**}

4. DeQueue()

Digunakan untuk menghapus elemen terdepan/pertama (head) dari Antrian Dengan cara menggeser semua elemen antrian kedepan dan mengurangi Tail dgn 1 Penggeseran dilakukan dengan menggunakan looping.

**Procedure DeQueueAlt3(input/output Q:queue)**

{ **IS**. Queue mungkin kosong atau penuh

**FS**. Queue di-dequeue dengan Alternative 3}

**Kamus**

**Algoritma**

**if ( Q.head = -1 and Q.Tail = -1 ) then** // Queue kosong

output('Stack kosong')

**else**

**if Q.Tail = Q.Head then** // data cuman 1 elemenm berada dimanapun

**Q.head ← -1**

**Q.tail ← -1**

**else if ( Q.Head <- nMax-1 and Q.tail <> Q.head ) then** // Kondisi khusus, head ada di ujung kanan dan tidak penuh

**Q.head <- 0**

**else** // Kondisi biasa

**Q.head ← Q.head + 1**

{**end if**}

{**end if**}