บทที่ 9

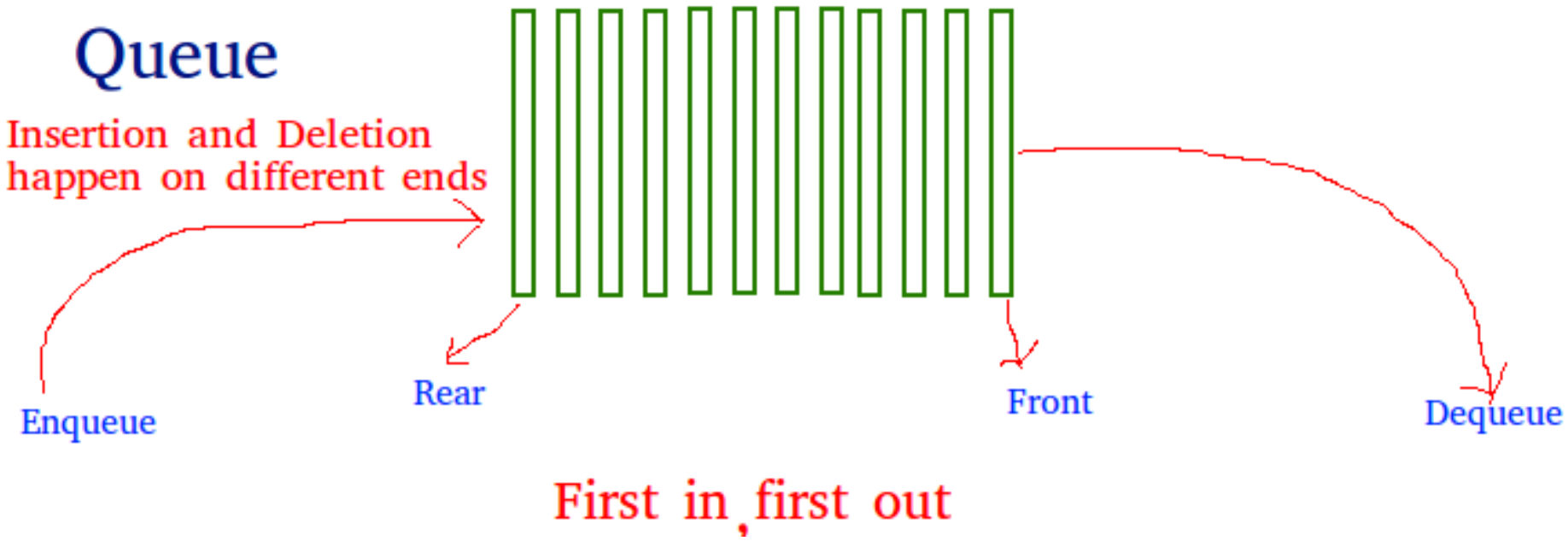
Queue

สุนทรี คุ้มไพโรจน์

QUEUE

FIFO : First In First Out

โครงสร้างข้อมูลแบบคิว



เนื้อหา

- คิว
- โอเปอร์เรชั่นของคิว
  - การเพิ่มข้อมูลเข้าไปในคิว (enqueue)
  - การนำข้อมูลออกจากคิว (dequeue)
- คิวแบบวงกลม
- การประยุกต์ใช้งานคิว

คิว : Queue

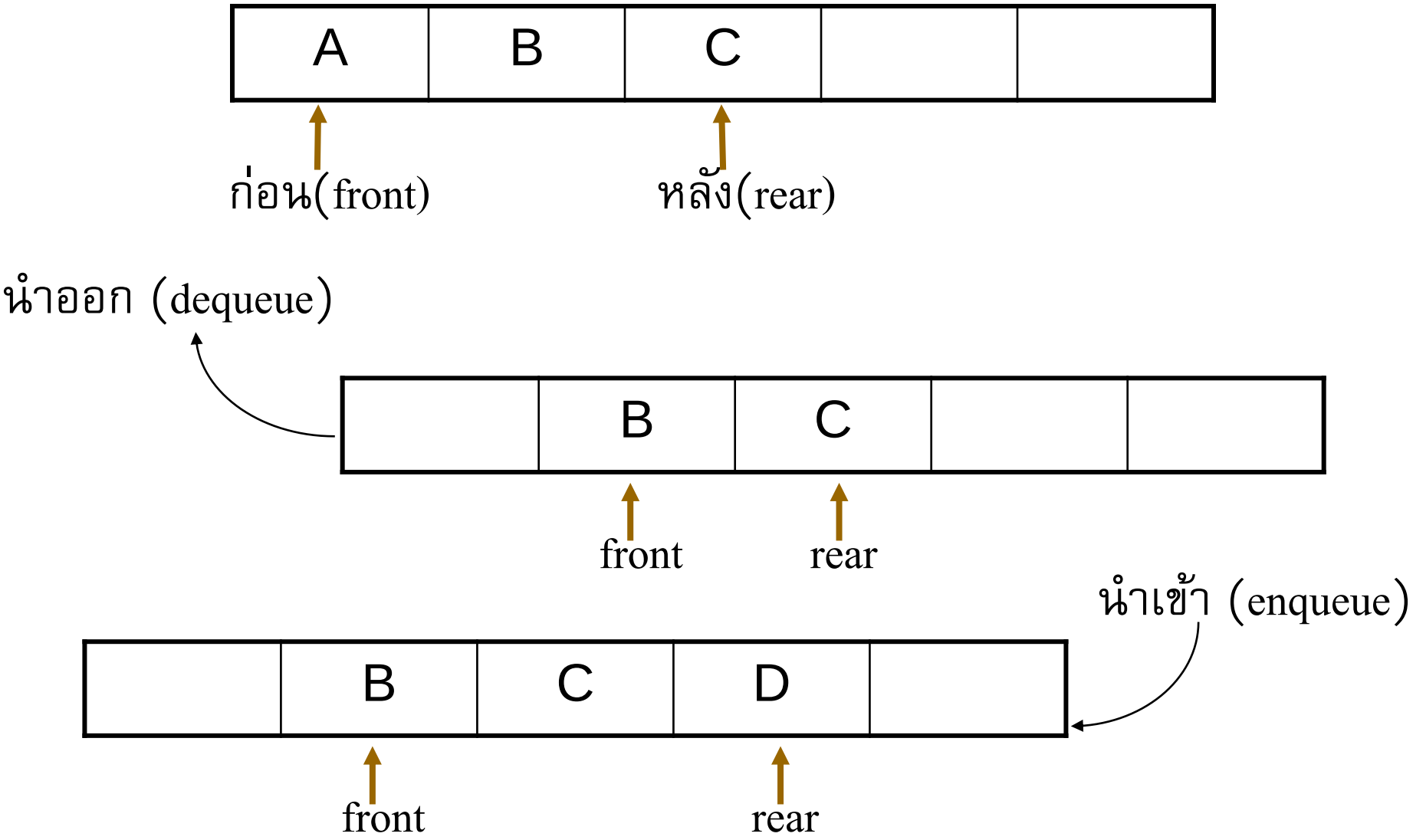
- โครงสร้างข้อมูลที่ประกอบด้วยสมาชิกที่เรียงติดต่อกันเป็นแถว

  เมื่อมีสมาชิกใหม่เข้าไปเสริมในคิวจะต้องเสริมจากทางด้านหลัง (rear)

  กรณีที่นำสมาชิกออกจากคิวจะต้องนำออกจากด้านหน้า (front)
- เป็นลิสต์แบบเชิงเส้น เช่นเดียวกับสแตก
- มีตัวชี้ 2 ตัว คือ front และ rear สำหรับการใส่ข้อมูลเข้าและนำข้อมูลออ
- มีกระบวนการทำงานแบบ First In First Out : FIFO หรือ

  First Come First Serve : FCFS

ตัวอย่างของคิว

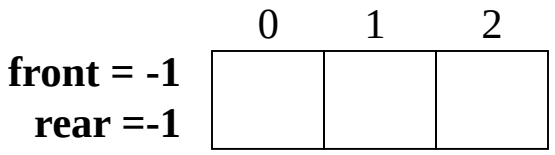


โอเปอร์เรชั่นของคิว

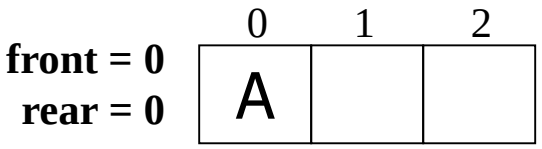
- การเพิ่มข้อมูลเข้าไปในคิว :Enqueue
- การนำข้อมูลออกจากคิว : Dequeue

โดยใช้การสร้างคิวด้วยโครงสร้างอาร์เรย์

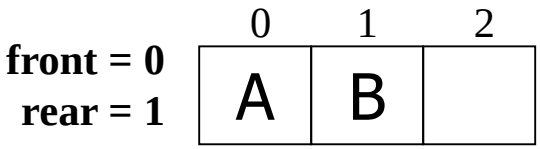
สถานะของคิว



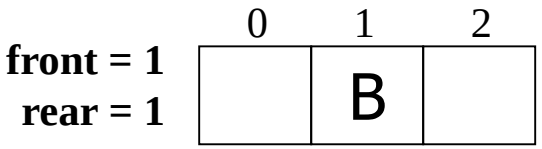
a) empty queue



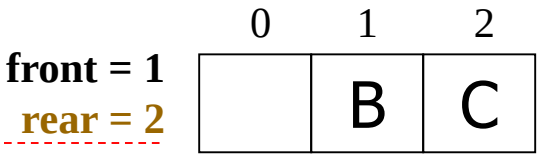
b) enqueue A



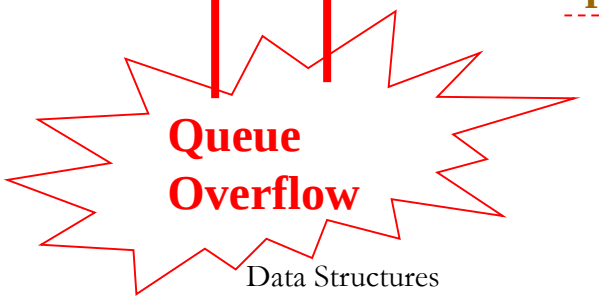
c) enqueue B

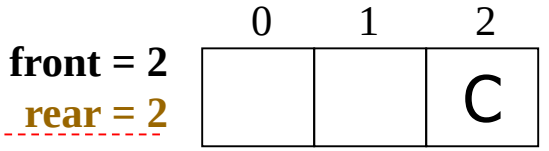


d) dequeue A



e) enqueue C





f) dequeue B

การเพิ่มข้อมูลเข้าไปในคิว Enqueue

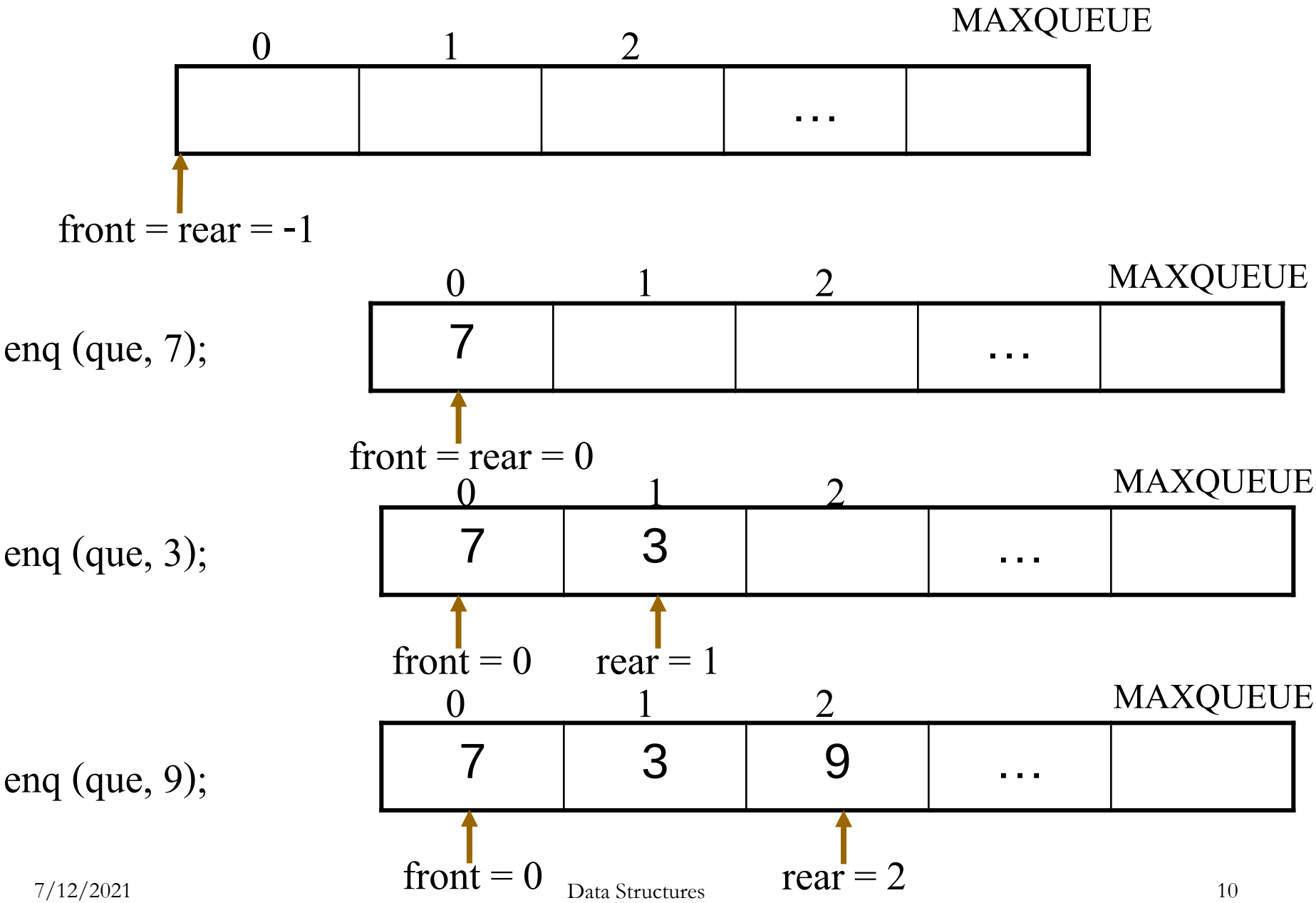
- ก่อนทำการ enqueue เพื่อเพิ่มข้อมูลเข้าที่ตำแหน่ง rear จะมีการตรวจสอบก่อนว่าคิวเต็มหรือไม่

  โดยตรวจสอบจากค่า rear = MAXQUEUE หรือไม่
- ถ้าคิวเต็มแล้ว ให้แสดงข้อความเตือนว่าคิวเต็ม

  ถ้ายังไม่เต็มให้ทำการเพิ่มข้อมูลเข้าไปในคิว

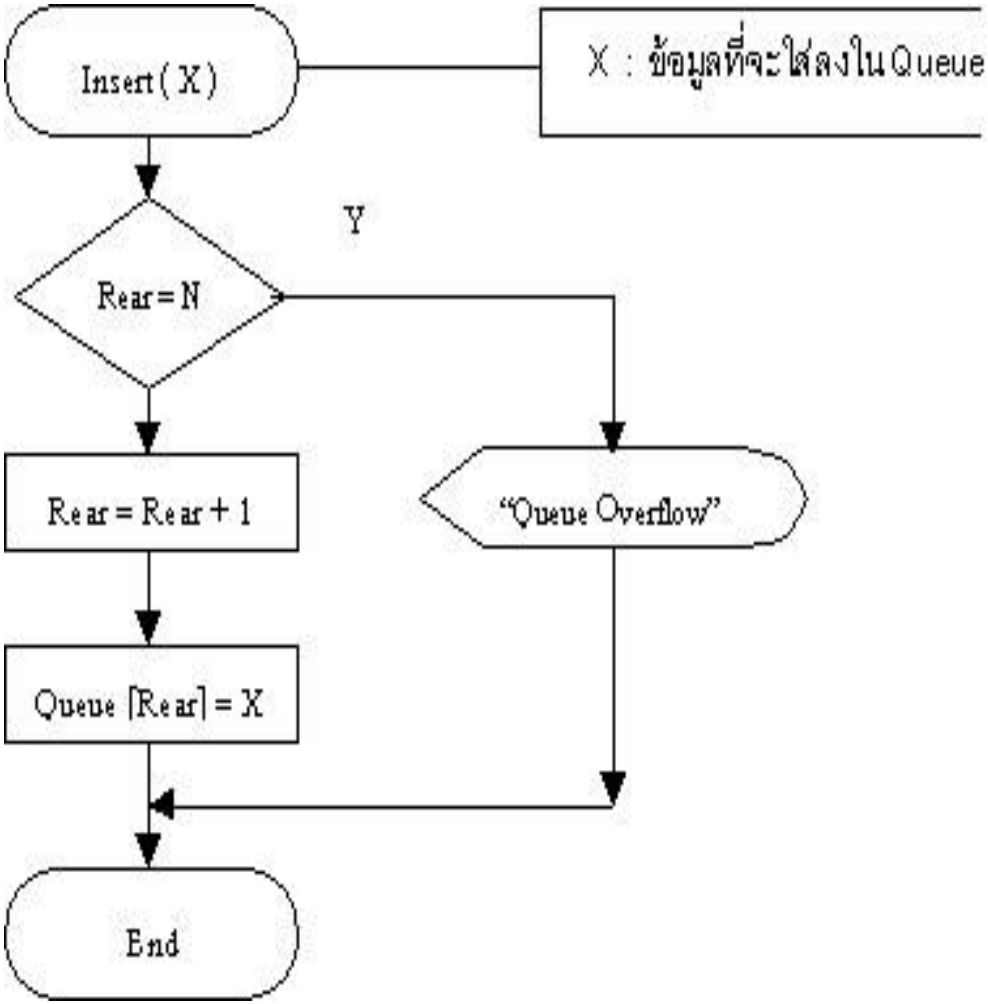
  และเพิ่มค่า rear ทีละ 1 ก่อนจะนำสมาชิกใหม่เข้ามา และทำลักษณะเช่นนี้จน กว่าคิวจะเต็ม

แสดงการ enqueue



Enqueue

```
addq ( int data )  {
    if ( rear == MAXQUEUE -1)  {
            printf ( "\nQueue is full" );
            return;
    }
    else  {
            rear++ ;
            arr[rear]  = data;
            if ( front == -1 )
                front = 0;
}
```



Algorithm

1. ตรวจสอบว่า Queue เต็ม ? (Rear = N)

   - ถ้า Queue เต็ม ให้แสดงข้อความว่า "Queue Overflow" แล้วเลิกงาน

   - ถ้า Queue ไม่เต็ม ให้ทำงานข้อที่ 2 และ 3

2. ให้เพิ่มค่าของ Rear อีก 1

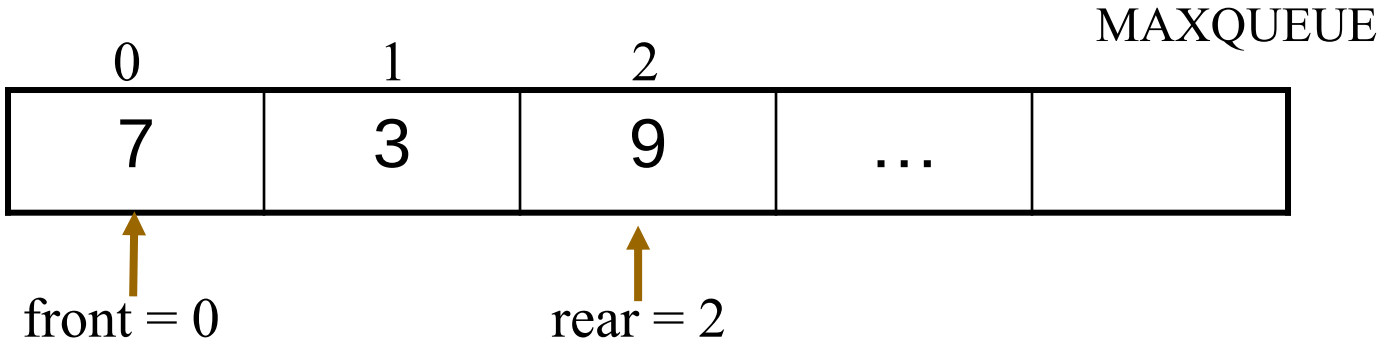
3. ใส่ข้อมูลลงใน Queue ในตำแหน่งของตัวแปร Rear

การนำข้อมูลออกจากคิว Dequeue

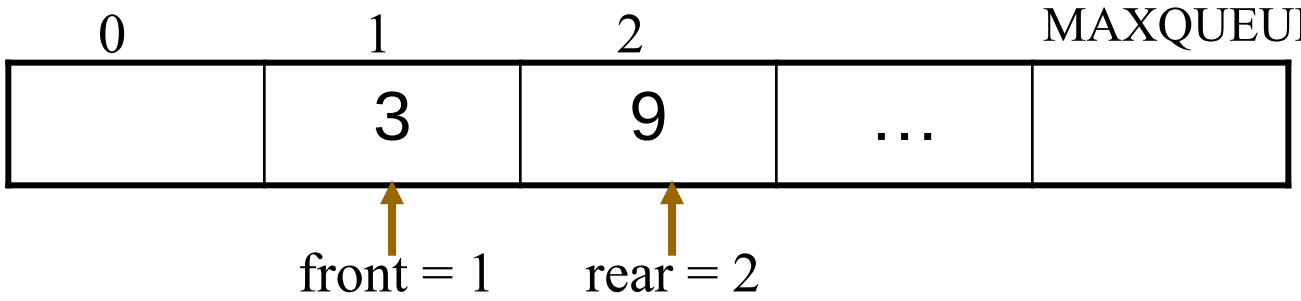
- ก่อนทำการ dequeue เพื่อนำข้อมูลออกที่ตำแหน่ง front จะมีการตรวจสอบก่อนว่าคิวว่างหรือไม่
  โดยตรวจสอบจาก front = rear = -1 หรือไม่

- ถ้าคิวว่าง ให้แสดงข้อความเตือนว่าคิวว่าง
  ถ้ายังไม่ว่าง(แสดงว่ามีข้อมูล)ให้ทำการนำข้อมูล ณ ตำแหน่งที่ front ชี้อยู่ออกจากคิว และเพิ่มค่า front ขึ้นทีละ 1 ทุกครั้งหลังจากนำข้อมูลออกจากคิวแล้ว ทำจนกระทั่ง front = rear โดยกำหนดให้ค่าของ front=-1 และ rear=-1 ซึ่งแสดงว่าคิวว่าง

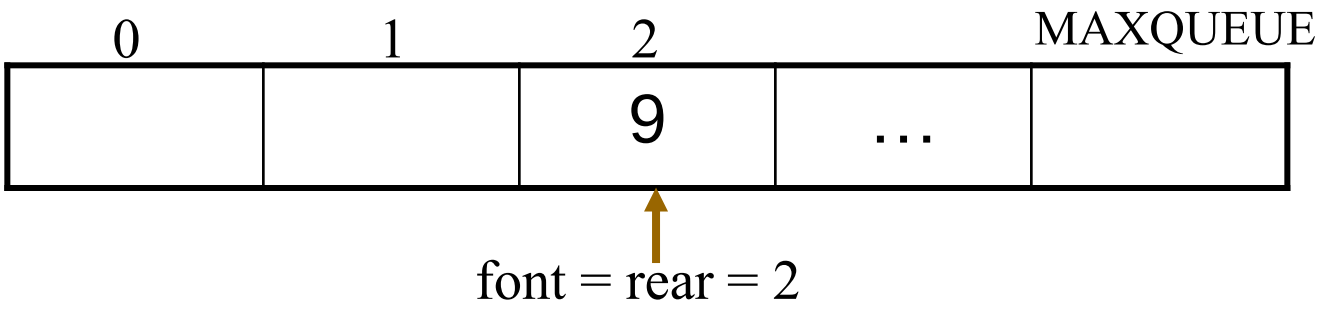
แสดงการ dequeue



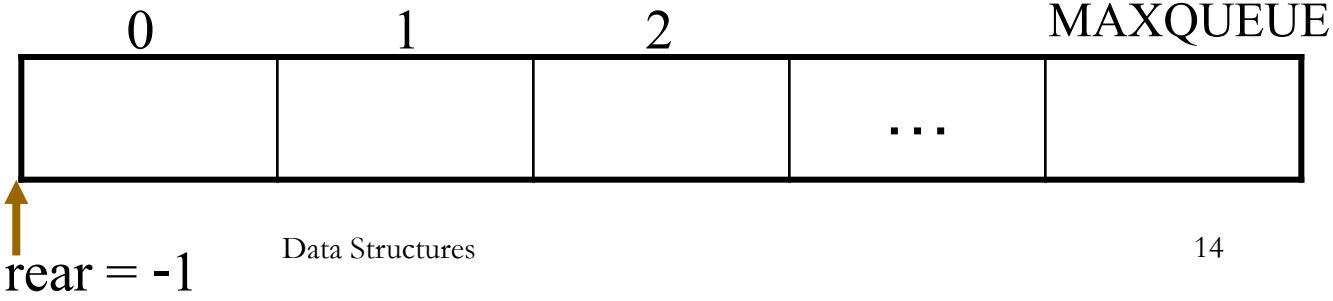
deq (que,data);



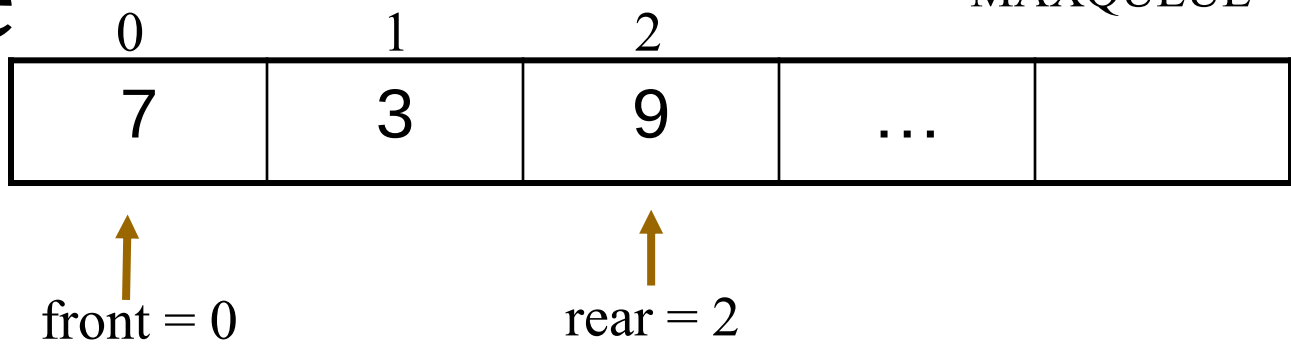
deq (que, data);



deq (que, data);

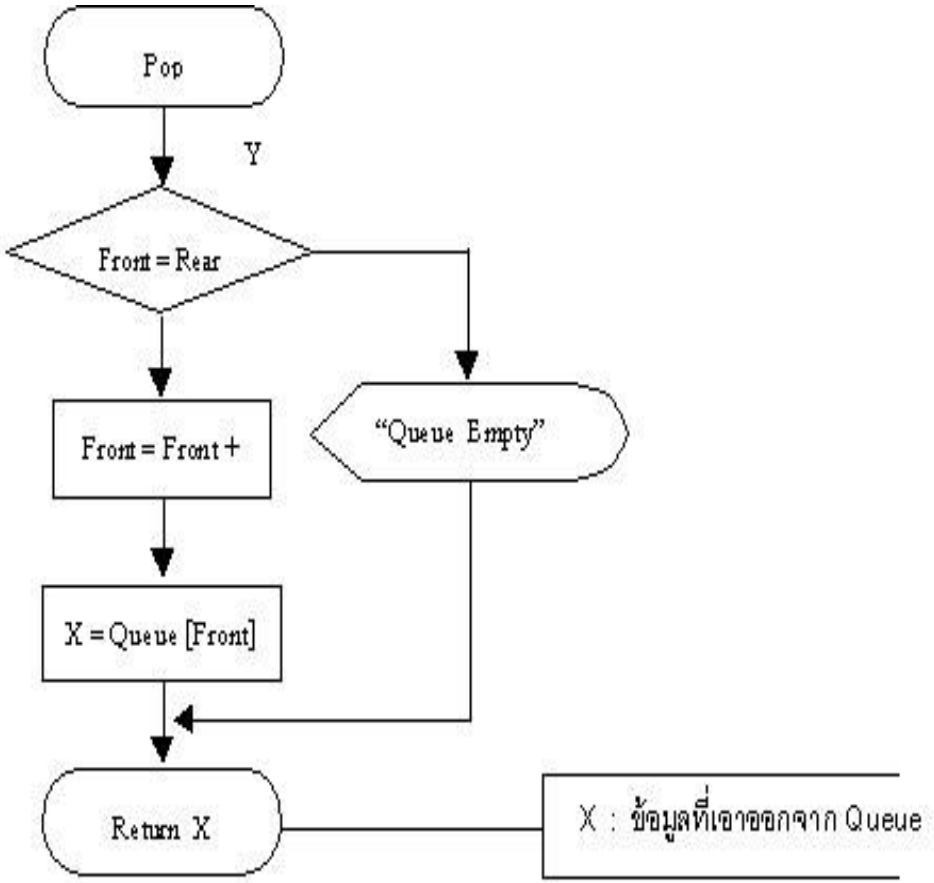


Dequeue




```
delq( )  {
    int data ;
    if ( front == -1 )  {
            printf ( "\nQueue is Empty" ) ;
             return NULL ;
    }
    data = arr[front] ;
    if ( front == rear )
            front = rear = -1 ;
    else
            front++ ;
    return data ;
}
```

กรณีลบคิวที่เหลือเพียงตัวเดียว



Algorithm

1. ตรวจสอบว่า Queue ว่าง ? (โดยการตรวจสอบว่า Front = Rear)

   - ถ้า Queue ว่าง ให้แสดงข้อความว่า "Queue Empty" แล้วเลิกงาน

   - ถ้า Queue ยังมีข้อมูล ให้ทำงานข้อที่ 2 และ 3

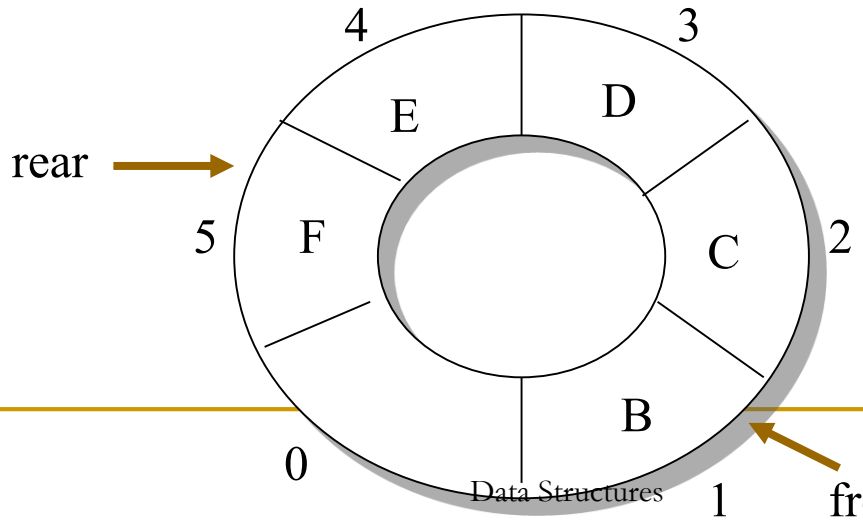
2. เพิ่มค่าของตัวแปร Front

3. ให้นำข้อมูลในตำแหน่งที่ Front ออกจาก Queue

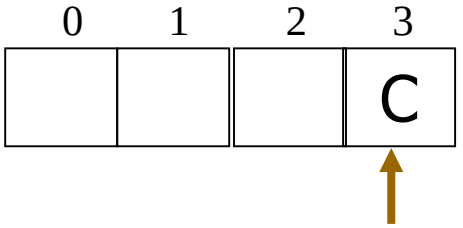
คิววงกลม : Circular Queue

- เป็นการออกแบบมาเพื่อใช้ในการแก้ปัญหา ในกรณีที่ยังมีพื้นที่เหลือในการเก็บข้อมูล
- คิววงกลมสามารถกลับไปใช้พื้นที่ด้านหน้าได้ในกรณีที่ใช้พื้นที่มาถึงปลายสุดของคิวแล้ว

คิววงกลม : Circular Queue



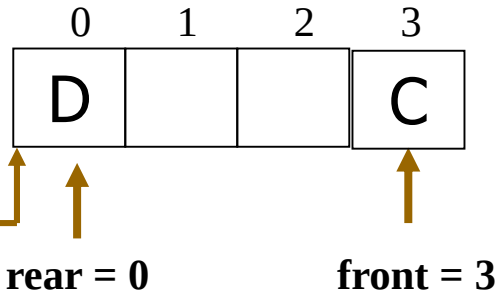
คิวแบบวงกลม



“queue overflow”

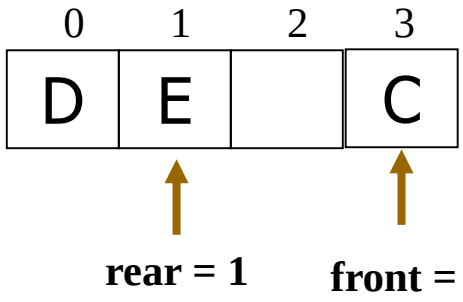
enqueue (D)

กรณีคิวเต็ม กำหนดให้ rear ไปชี้ที่จุดเริ่มต้นใหม่ (0) เพื่อสามารถเพิ่ม D เข้าไปได้

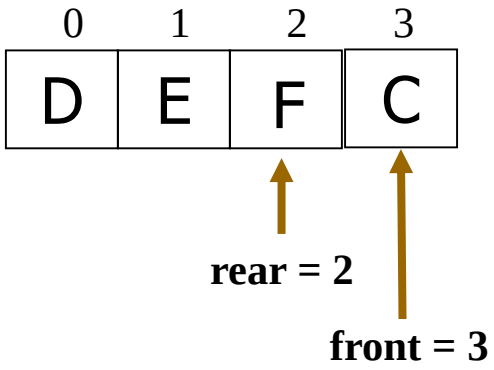


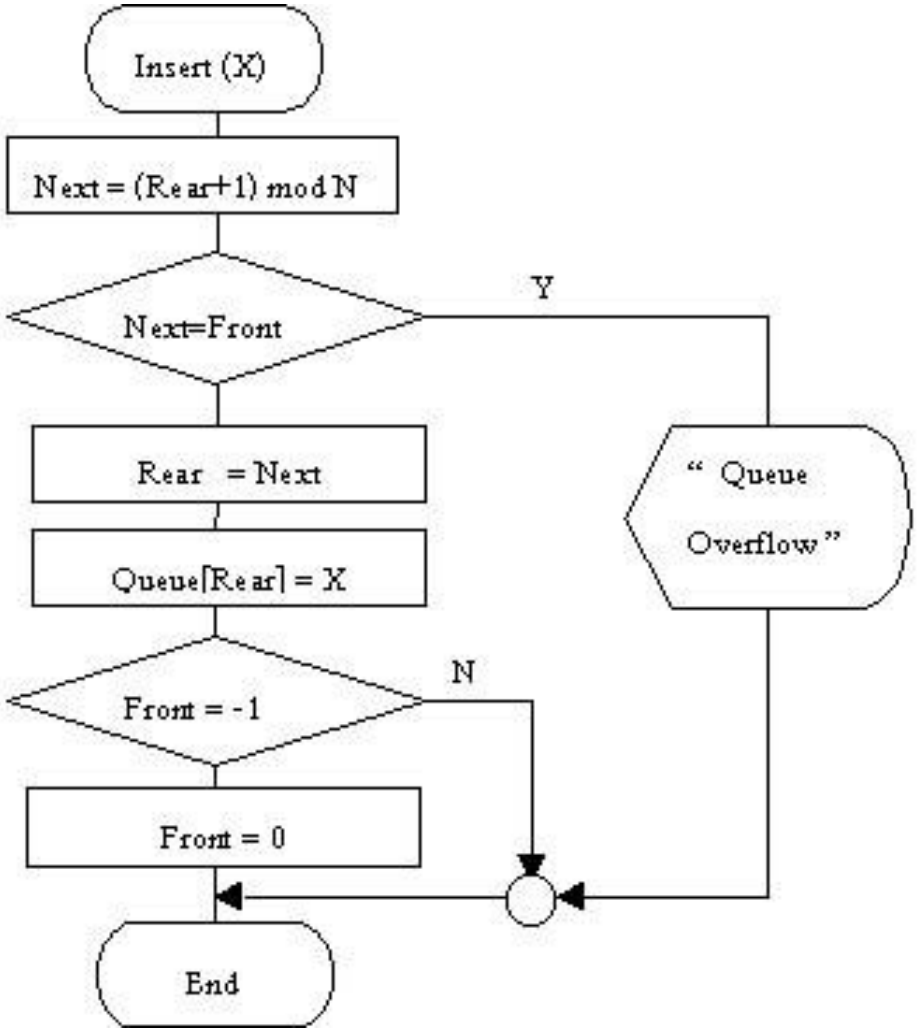
เพิ่มข้อมูลในคิวแบบวงกลม

enqueue (E)



enqueue (F)





Algorithm Insert

1. ตรวจสอบว่าคิวเต็ม ?

(ตำแหน่งต่อไปของ Rear = Front ?)

- ถ้าคิวเต็มให้แสดงข้อความว่า "Queue Overflow" แล้วเลิกการทำงาน

2. เลื่อน Rear ไปคิวต่อไป

3. นำข้อมูลใส่ลงในคิว

4. ตรวจสอบว่า Front เป็น -1 หรือไม่ (กรณีที่เป็นการใส่ข้อมูลเป็นตัวแรกของคิว)

- ถ้าใช่ ให้ Front = 0

(Front รอที่ตำแหน่งแรกของข้อมูลในคิว)

Enqueue of Circular Queue

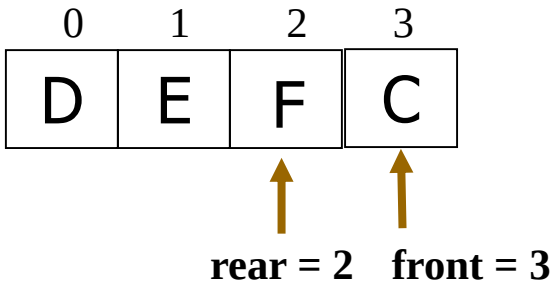
```
addq ( int item )  {
    if ( ( rear == MAX -1 && front == 0 ) || ( rear + 1 == front ) )  {
            printf ( "\nQueue is full" ) ;
            return ;
    }
    if ( rear == MAX - 1 )
            rear = 0 ;
    else
            rear++ ;
    arr[rear] = item ;
    if ( front == -1 )
            front = 0 ;
}
```

แสดงข้อความว่า "คิวเต็ม"

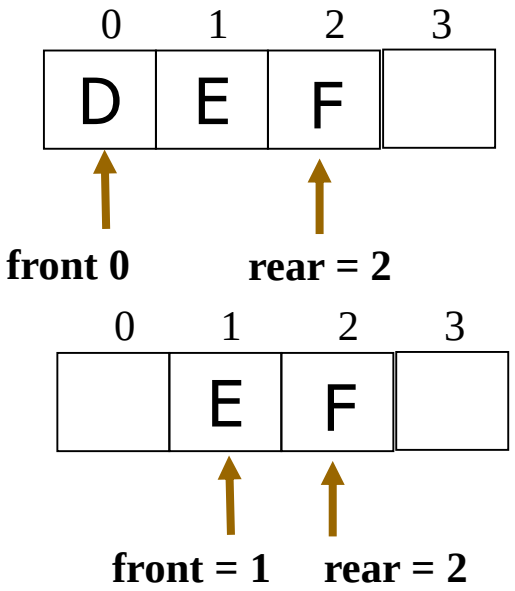
เมื่อต้องการเพิ่มข้อมูลเข้าไปอีก ให้กำหนด rear ไปชี้ที่จุดเริ่มต้นใหม่

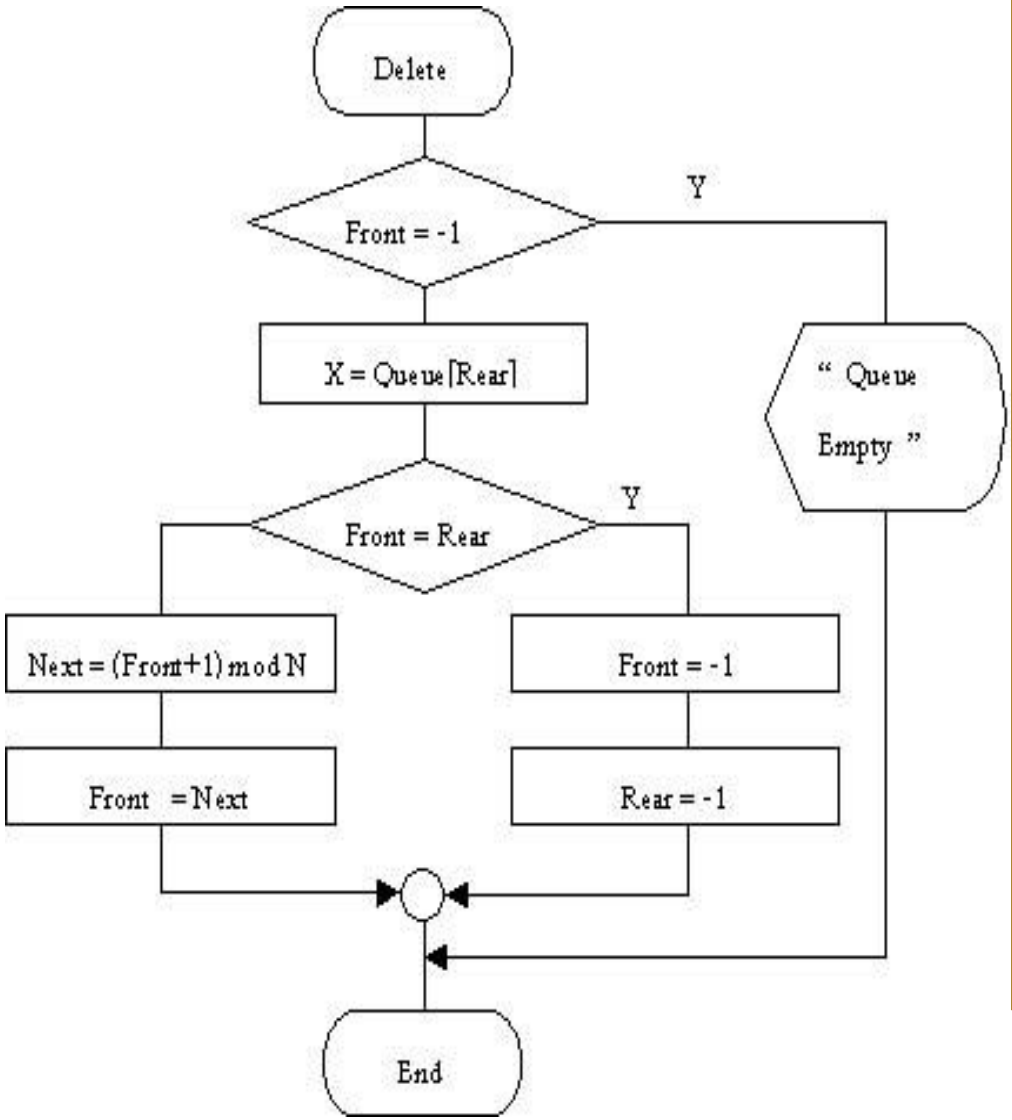
ลบข้อมูลในคิวแบบวงกลม

deq (que,data);



deq (que,data);





Algorithm Delete

1. ตรวจสอบว่าคิวว่างหรือไม่ ?

   (Front = -1 ?)

2. นำข้อมูลออกจากคิว
3. ตรวจสอบว่าเป็นข้อมูลตัวสุดท้ายของคิวหรือไม่

   (Front = Rear ?)

   - ถ้าใช่ ให้ Front และ Rear เป็น -1

     (กำหนดให้ไม่มีข้อมูลในคิวเลย)

   - ถ้าไม่ใช่ ให้เลื่อน Front ไปรอในคิวต่อไป

Dequeue of Circular Queue

```
delq( )  {
     int data ;
     if ( front == -1 )  {
               printf ( "\nQueue is Empty" ) ;
          return NULL ;
     }
     else  {
               data = arr[front] ;
          if ( front == rear )
                front = rear = -1 ;
          else  {
                 if ( front == MAX - 1 )
                      front = 0 ;
                else
                      front++ ;
          }
     }
     return data ;
}
```

ลบคิวที่เหลือเพียงตัวเดียว

ลบคิวที่ช่องความจำสุดท้าย กำหนดให้ front วนไปชี้ที่จุดเริ่มต้นใหม่

ตัวอย่างโปรแกรม Queue

```c
#include <stdio.h>
#define MAX_QUEUE_SIZE 4
#define TRUE 1
#define FALSE 0
typedef struct int_queue_type {
    int item[MAX_QUEUE_SIZE];
    int front, back, count;
}  Queue;

void initQueue(Queue *queue){
    queue->count = 0;
    queue->front = queue->back = 0;
}

int enQueue(Queue *queue, int x){
    if (queue->count >= MAX_QUEUE_SIZE)
        return FALSE;
    else {
        queue->count++;
        queue->item[queue->back] = x;
        queue->back = (queue->back+1)% MAX_QUEUE_SIZE;
        return TRUE;
    }
}
int deQueue(Queue *queue, int *x){
    if (queue->count <=0)
        return FALSE;
    else {
        queue->count--;
        *x = queue->item[queue->front];
        queue->front = (queue->front + 1) % MAX_QUEUE_SIZE;
        return TRUE;
    }
}

void printQueue(Queue queue){
    int i;
    int pos = queue.front;

    printf("Queue:");
    for (i=0; i< queue.count; i++){
        printf("%d", queue.item[pos]);
        pos = (pos + 1) % MAX_QUEUE_SIZE;
    }
    printf("\n");
}
```

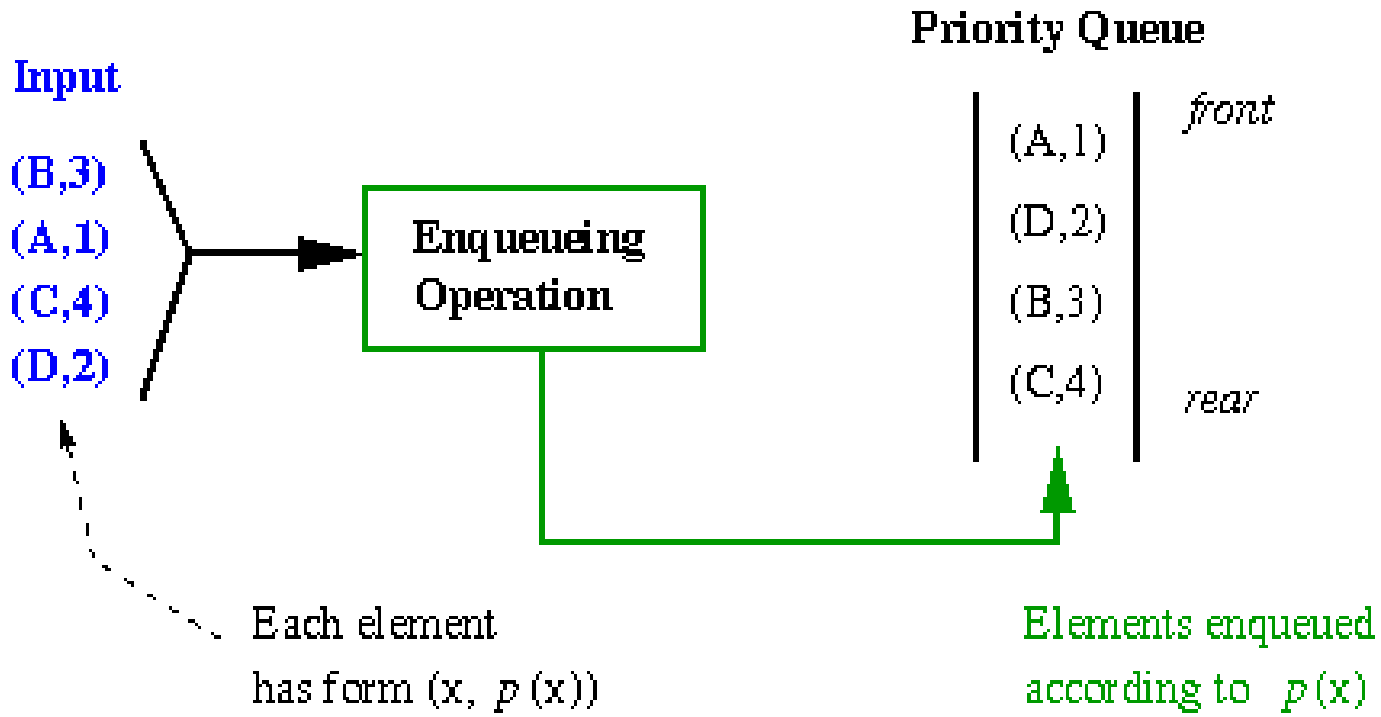
```c
void main(){
    Queue q;
    int choice;
    int cont = TRUE;
    int x;

    initQueue(&q);
    while (cont == TRUE){
        printf("Please select [1:enQueue 2:deQ
        scanf("%d", &choice);
        switch(choice){
        case 1:
            printf("Please enter a number to b
            scanf("%d", &x);
            if (!enqueue(&q,x))
                printf(" Error putting into th
            break;
        case 2:
            if (deQueue(&q,&x))
                printf("The number from the qu
            else
                printf(" Error getting an item
            break;
        case 3:
            printQueue(q);
            break;
        case 0:
            cont = FALSE;
            break;
        }
    }
}
```

คิวลำดับความสำคัญ หรือ แถวคอยเชิงบุริมภาพ (Priority Queue)

- ในคิวปกติ ข้อมูลที่เข้ามาก่อนจะมีสิทธิ์ออกก่อน (First In First Out:FIFO) อย่างไรก็ตาม มีบางครั้งที่เราต้องยกให้สมาชิกบางประเภทได้ทำงานก่อนทั้งที่มาทีหลัง เช่นการให้คิวงานที่เล็กกว่าได้ทำก่อน หรือ การให้สิทธิพิเศษแก่การทำงานบางประเภท
- **คิวลำดับความสำคัญ**ทำให้เราสามารถประยุกต์ใช้คิวได้ดีขึ้น เนื่องจากเพิ่มการให้ความสำคัญของสมาชิกที่แตกต่างกัน ส่งผลให้เราสามารถจัดเรียงคิวได้ใหม่ให้เหมาะสมกับการทำงานได้ เราใช้**คิวลำดับความสำคัญ**ในการจัดการทำงาน การตรวจนับ

คิวลำดับความสำคัญ หรือ แถวคอยเชิงบุริมภาพ (Priority Queue)



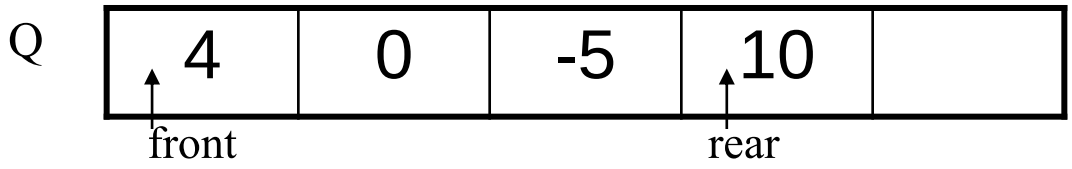
Exercise

เขียนโปรแกรมมีเงื่อนไขว่า ใครที่อายุ(Age)น้อยที่สุด จะถูกเรียกชื่อก่อนเสมอ โดยหากมีหลายคนที่อายุเท่ากัน คนที่ถูกเพิ่มเข้าไปในคิวภายหลังจากถูกเรียกก่อนเสมอ

https://th.wikipedia.org/wiki/%E0%B9%81%E0%B8%96%E0%B8%A7%E0%B8%84%E0%B8%AD%E0%B8%A2%E0%B8%A5%E0%B8%B3%E0%B8%94%E0%B8%B1%E0%B8%9A%E0%B8%84%E0%B8%A7%E0%B8%B2%E0%B8%A1%E0%B8%AA%E0%B8%B3%E0%B8%84%E0%B8%B1%E0%B8%8D

แบบฝึกหัด

1. จงอธิบายถึงสิ่งที่เหมือนกัน และสิ่งที่แตกต่างกันระหว่าง สแตก กับ คิว
2. จงอธิบายถึงสิ่งที่เหมือนกัน และสิ่งที่แตกต่างกันระหว่าง คิวเส้นตรง กับ คิววงกลม
3. จงเขียนแผนภาพการดำเนินการตามขั้นตอนซึ่งมีข้อมูลในคิววงกลมดังนี้



การดำเนินการ

1. deq (Q,x)
2. enq (Q,2)
3. deq (Q,x)
4. deq (Q,x)
5. enq (Q,-3)
6. deq (Q,x)

แบบฝึกหัด (ต่อ)

4. คิวขนาด 7 ค่า มีรายการต่อไปนี้

front = 2 , rear = 5

Queue[] = {null, null, “ขนุน”, “ส้มโอ”,”แตงโม”, “ฝรั่ง”, null}

จงแสดงผลของคิวแบบวงกลม โดยมีค่า front และ rear ตามที่กำหนดข้างบน เมื่อมีการกระทำต่อไปนี้

ก. เพิ่ม “มะม่วง” ข. นำ 2 รายการออกจากคิว

ค. เพิ่ม “ทุเรียน” ง.เพิ่ม “องุ่น”

จ. นำ 2 รายการออกจากคิว ฉ. เพิ่ม “ละมุด”

แบบฝึกหัด (ต่อ)

5. จงยกตัวอย่าง การใช้คิวในชีวิตประจำวัน หรือในระบบคอมพิวเตอร์ มา 5 ตัวอย่าง