

บทที่ 9

Queue

สุนทรี คุ่มไพโรจน์

QUEUE

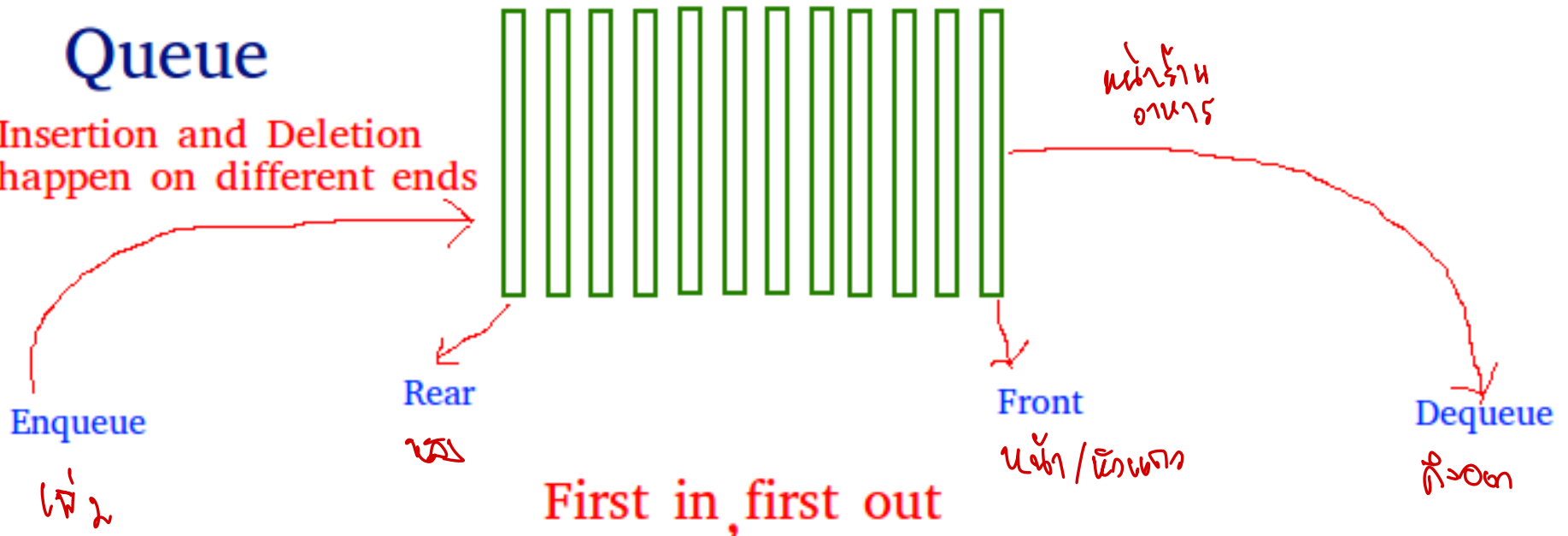
FIFO : First In First Out



โครงสร้างข้อมูลแบบคิว

Queue

Insertion and Deletion happen on different ends



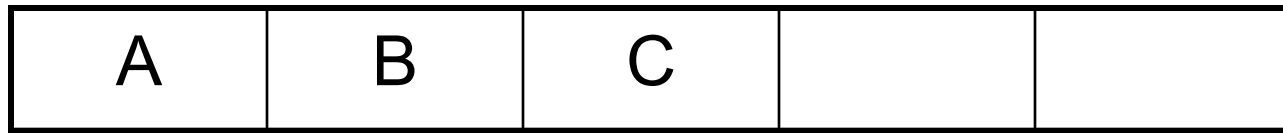
เนื้อหา

- คิว
- โอเปอเรชันของคิว
 - การเพิ่มข้อมูลเข้าไปในคิว (enqueue)
 - การนำข้อมูลออกจากคิว (dequeue)
- คิวแบบวงกลม *Priority Queue*
- การประยุกต์ใช้งานคิว

คิว : Queue

- โครงสร้างข้อมูลที่ประกอบด้วยสมาชิกที่เรียงติดต่อกันเป็นแถว
เมื่อมีสมาชิกใหม่เข้าไปเสริมในคิวจะต้องเสริมจากทางด้านหลัง (rear)
กรณีที่นำสมาชิกออกจากคิวจะต้องนำออกจากด้านหน้า (front)
- เป็นลิสต์แบบเชิงเส้น เช่นเดียวกับสแต็ก
- มีตัวชี้ 2 ตัว คือ front และ rear สำหรับการใส่ข้อมูลเข้าและนำข้อมูลออก
- มีกระบวนการทำงานแบบ First In First Out : FIFO หรือ
First Come First Serve : FCFS

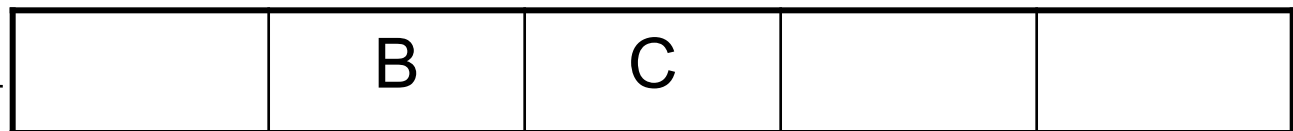
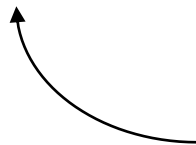
ตัวอย่างของคิว



↑
ก่อกน(front)

↑
หลัง(rear)

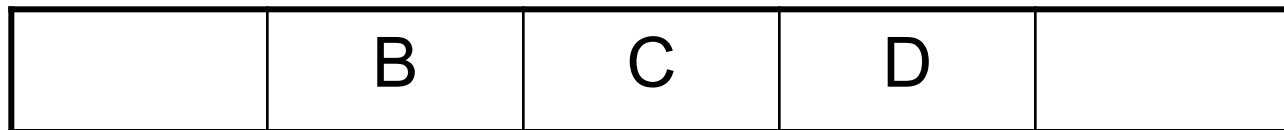
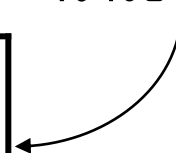
นำออก (dequeue)



↑
front

↑
rear

นำเข้า (enqueue)



↑
front

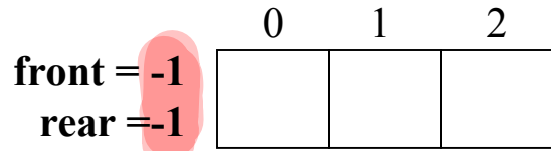
↑
rear

โอเพอร์เรชันของคิว

- การเพิ่มข้อมูลเข้าไปในคิว : Enqueue
- การนำข้อมูลออกจากคิว : Dequeue

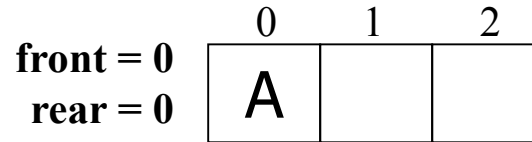
โดยใช้การสร้างคิวด้วยโครงสร้างอาร์เรย์

สถานะของคิว

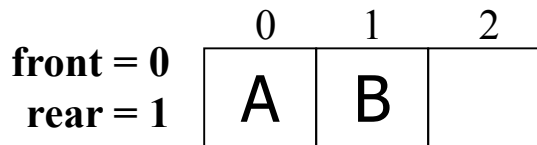


a) empty queue

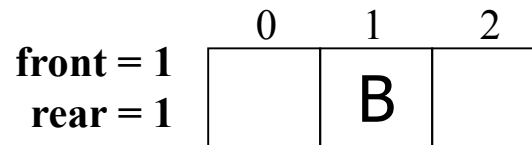
array = 3



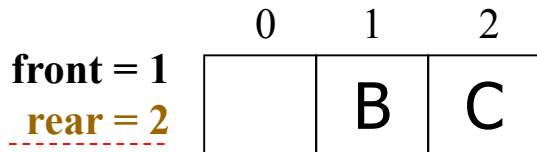
b) enqueue A



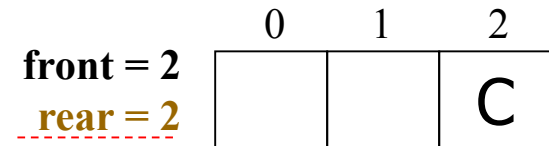
c) enqueue B



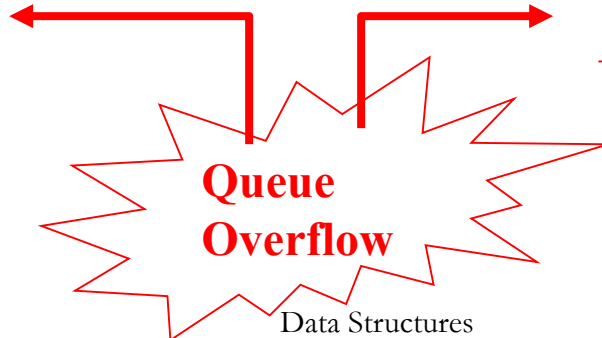
d) dequeue A



e) enqueue C



f) dequeue B



การเพิ่มข้อมูลเข้าไปในคิว Enqueue

- ก่อนทำการ **enqueue** เพื่อเพิ่มข้อมูลเข้าที่ตำแหน่ง **rear** จะมีการตรวจสอบก่อนว่าคิวเต็มหรือไม่

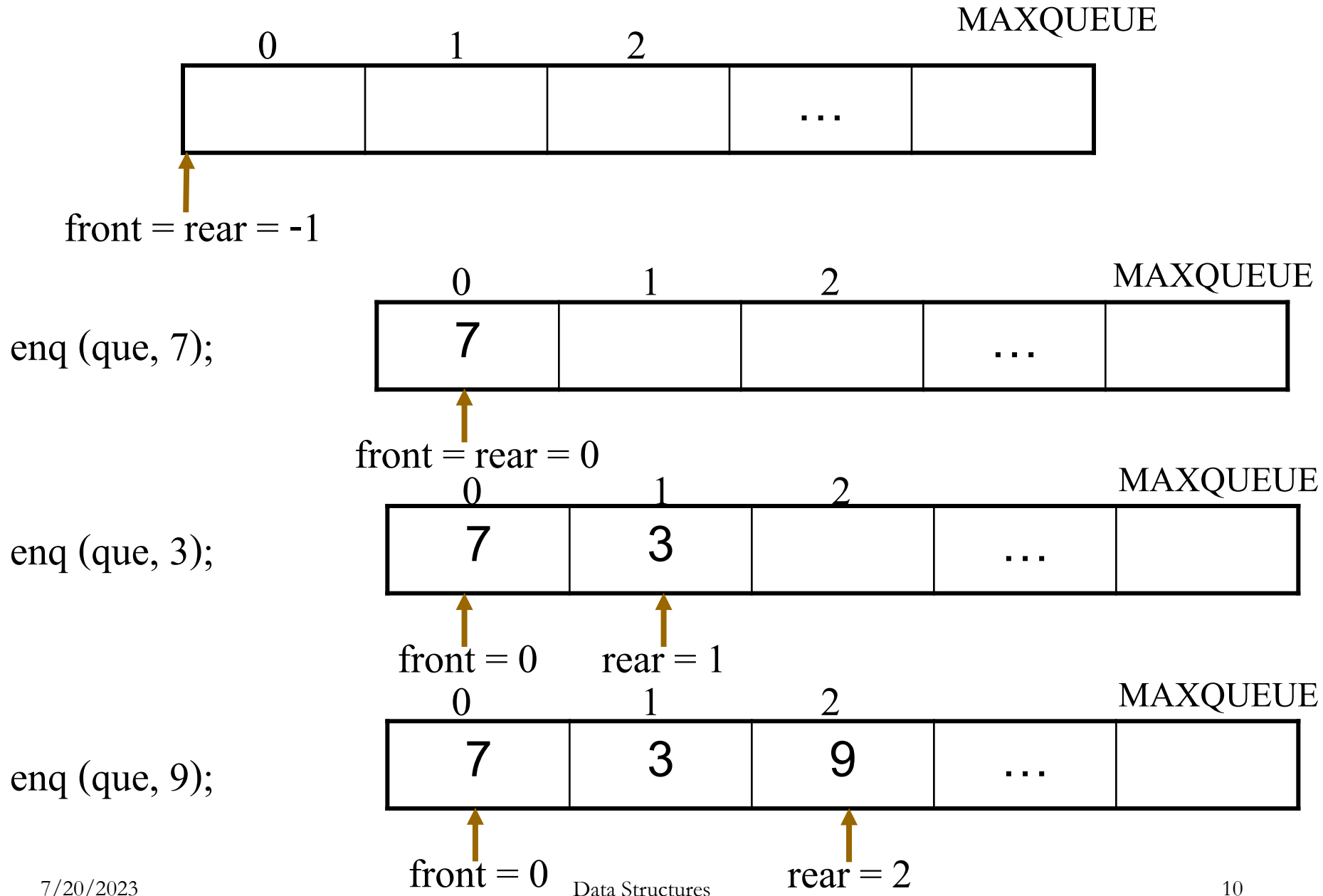
โดยตรวจสอบจากค่า **rear = MAXQUEUE** หรือไม่

- ถ้าคิวเต็มแล้ว ให้แสดงข้อความเตือนว่าคิวเต็ม

ถ้ายังไม่เต็มให้ทำการเพิ่มข้อมูลเข้าไปในคิว

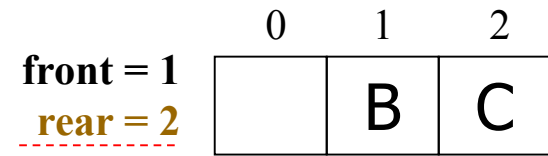
และเพิ่มค่า **rear** ที่ละ 1 ก่อนจะนำสมาชิกใหม่เข้ามา และทำลักษณะเช่นนี้จนกว่าคิวจะเต็ม

แสดงการ enqueue

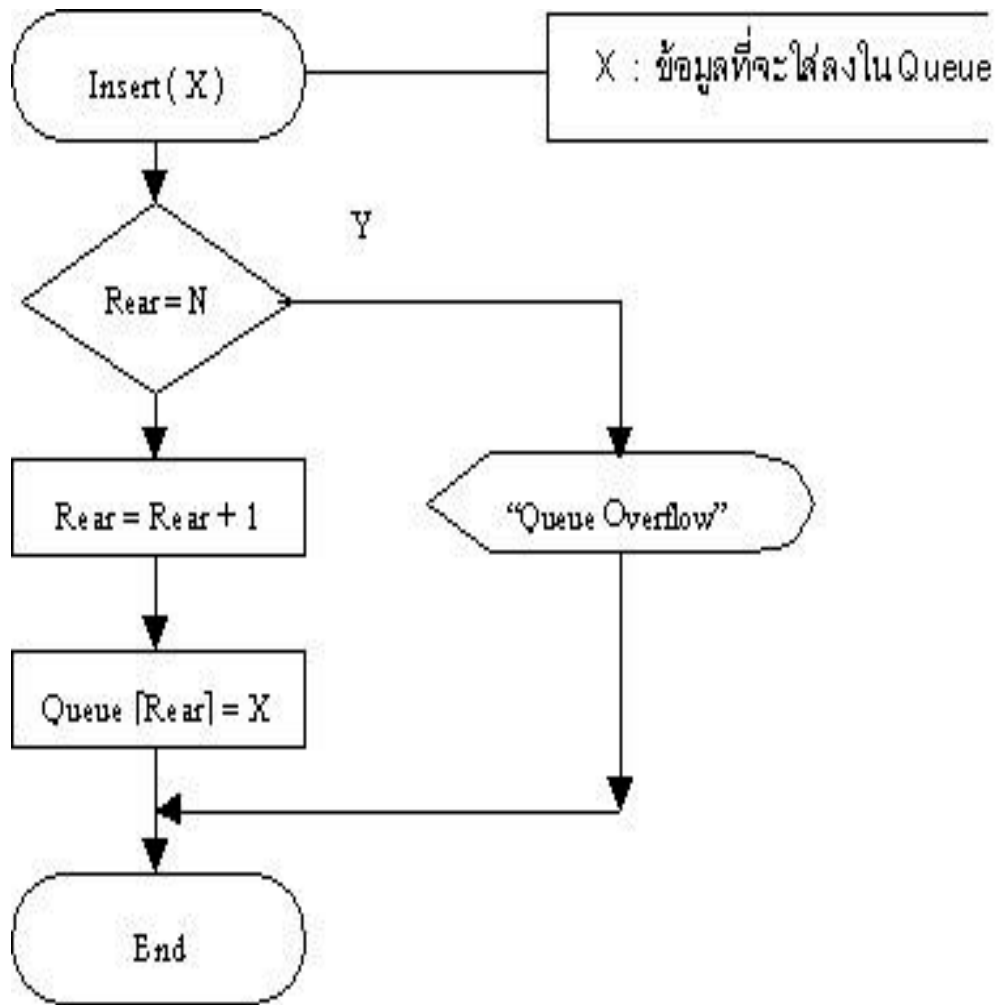


Enqueue

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



e) enqueue C



Algorithm

1. ตรวจสอบว่า Queue เต็ม ? (Rear = N)

- ถ้า Queue เต็ม ให้แสดงข้อความว่า "Queue Overflow" แล้วเลิกงาน

- ถ้า Queue ไม่เต็ม ให้ทำงานข้อที่ 2 และ 3

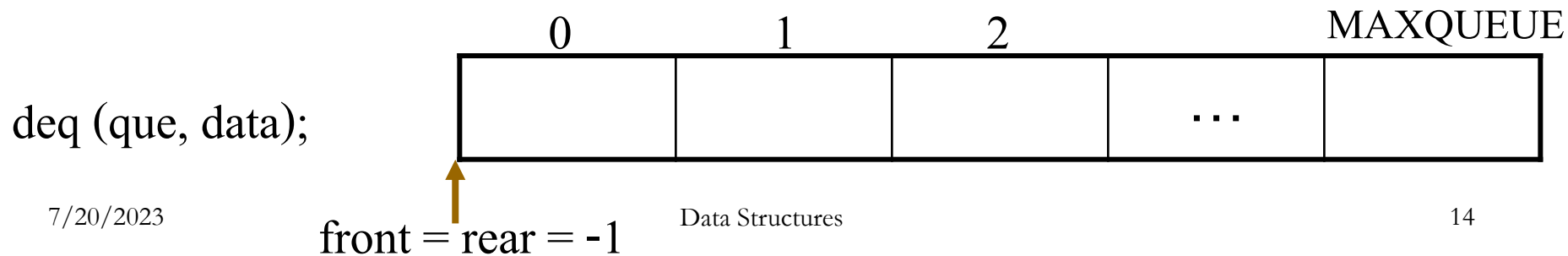
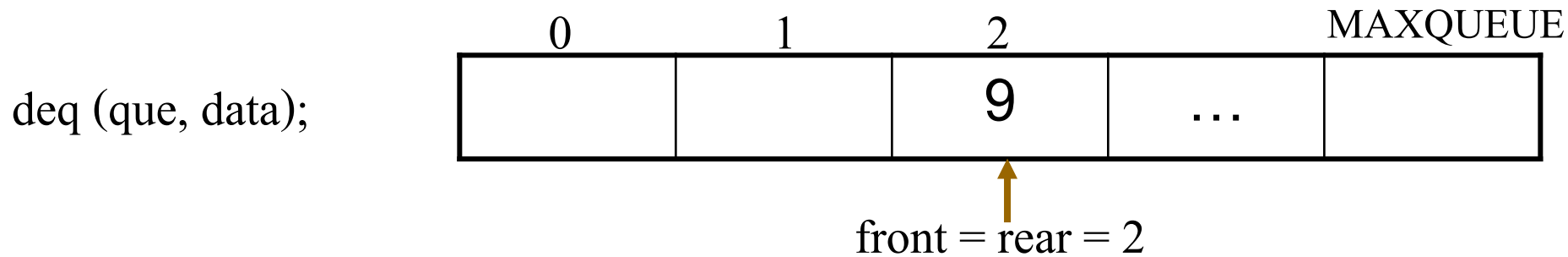
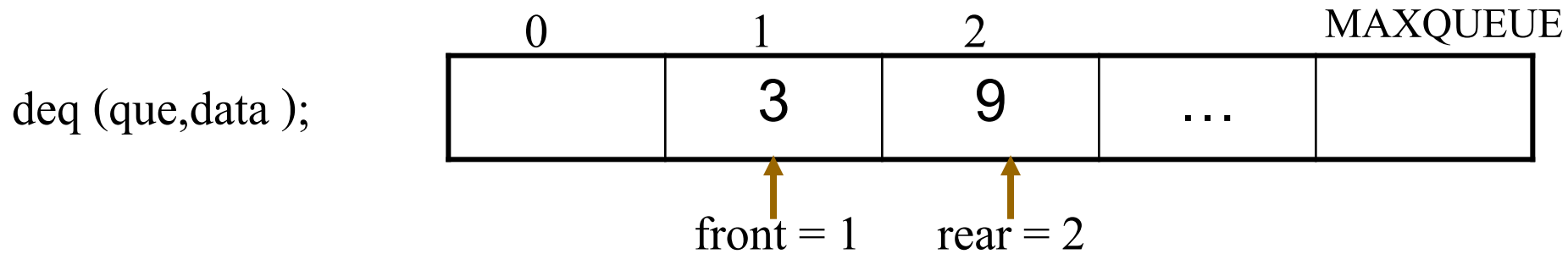
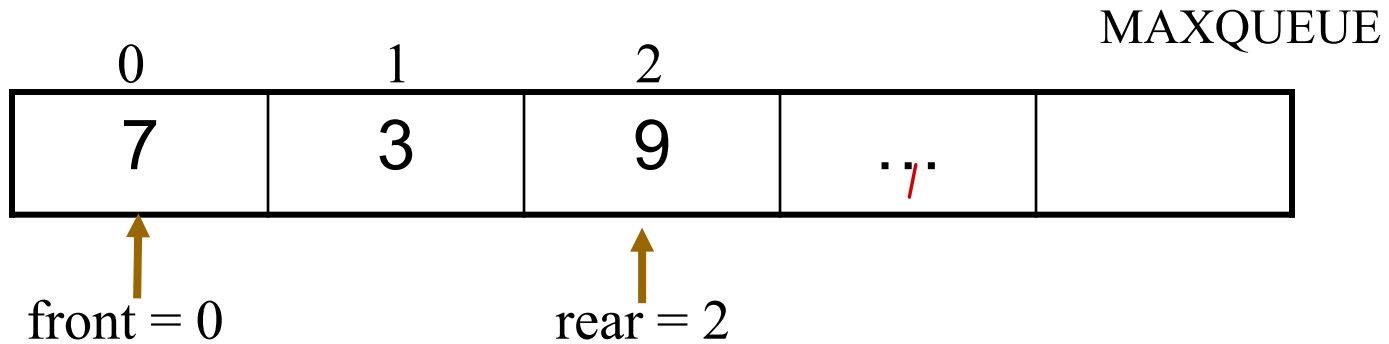
2. ให้เพิ่มค่าของ Rear อีก 1

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



Dequeue

MAXQUEUE

0	1	2		
7	3	9	...	

↑
front = 0

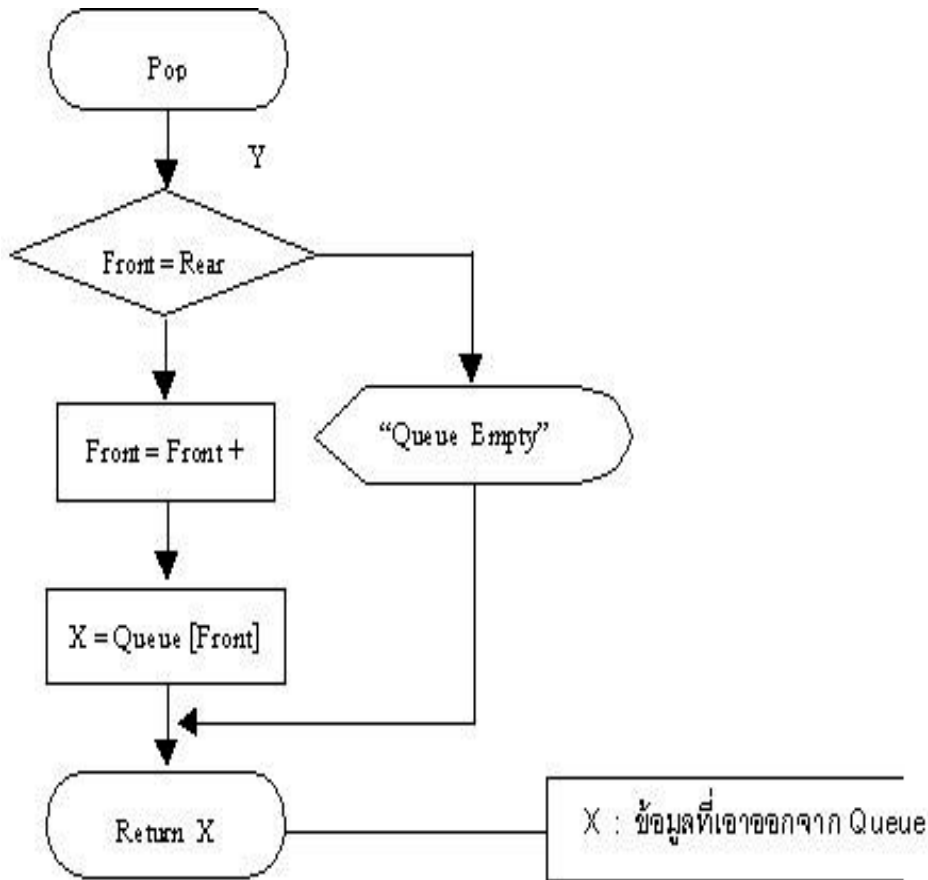
↑
rear = 2

```
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 ;  
}
```

get empty

ไปดึงตัวแถวตอน เดียว

กรณีลบคิวที่เหลือเพียงตัวเดียว



Algorithm

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

- ถ้า Queue ว่าง ให้แสดงข้อความว่า "Queue Empty" แล้วเลิกงาน

- ถ้า Queue ยังมีข้อมูล ให้ทำงานข้อที่ 2 และ 3

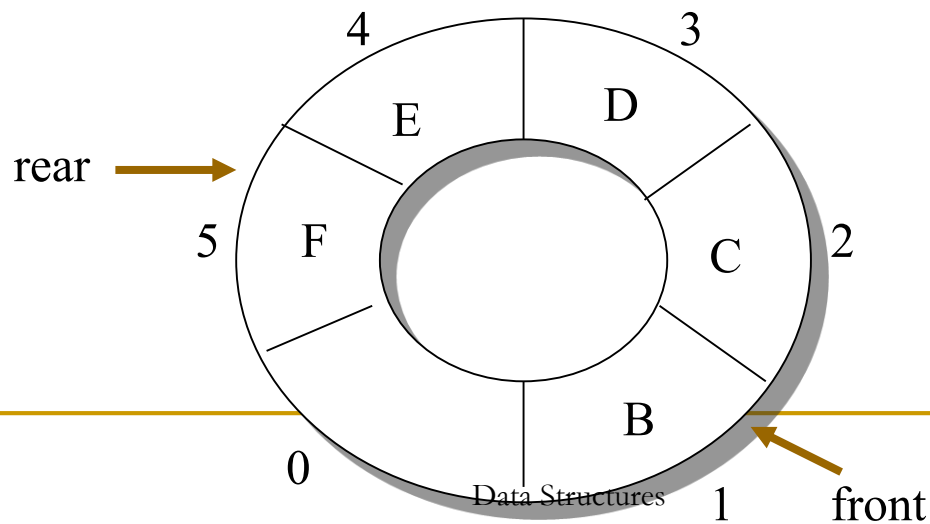
2. เพิ่มค่าของตัวแปร Front

3.ให้นำข้อมูลในตำแหน่งที่ Front ออกจาก Queue

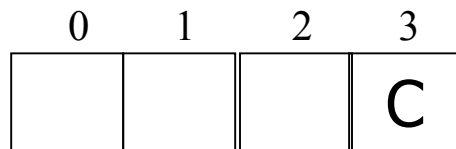
คิววงกลม : Circular Queue

- เป็นการออกแบบมาเพื่อใช้ในการแก้ปัญหา
ในกรณีที่ยังมีพื้นที่เหลือในการเก็บข้อมูล
- คิววงกลมสามารถกลับไปใช้พื้นที่ด้านหน้าได้ในกรณีที่ใช้พื้นที่
มาถึงปลายสุดของคิวแล้ว

คิวงกลม : Circular Queue



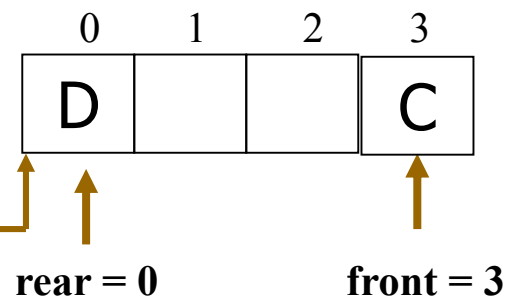
คิวแบบวงกลม



“queue overflow”

enqueue (D)

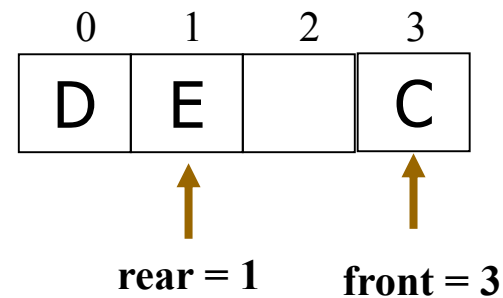
กรณีคิวเต็ม กำหนดให้ rear ไปชี้ที่
จุดเริ่มต้นใหม่ (0) เพื่อสามารถเพิ่ม D
เข้าไปได้



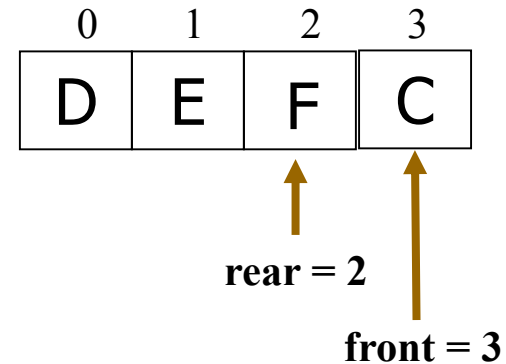
เพิ่มข้อมูลในคิวแบบวงกลม

$(\text{Rear} + 1) \% 4$

enqueue (E)



enqueue (F)



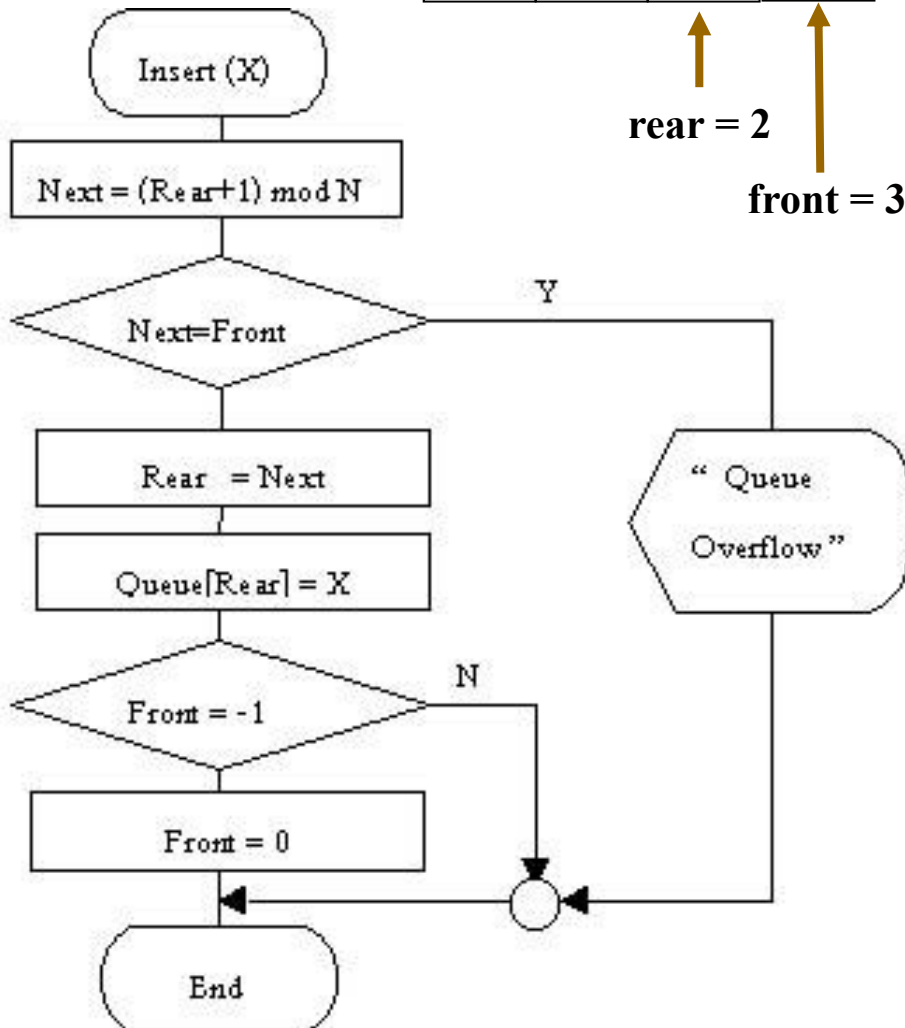
0	1	2	3
D	E	F	C

↑
rear = 2

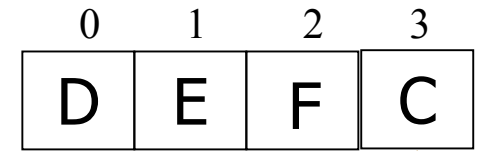
↑
front = 3

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 $Front = 0$
($Front$ รวที่ตำแหน่งแรกของข้อมูลในคิว)



Enqueue of Circular Queue



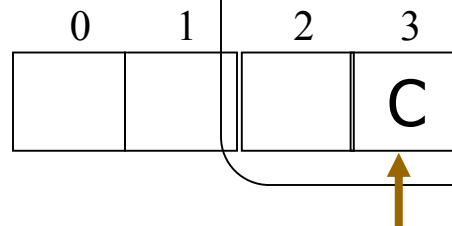
↑
rear = 2

↑
front = 3

```
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"

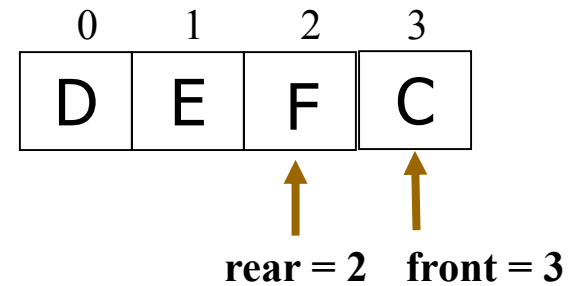
เมื่อต้องการเพิ่มข้อมูลเข้าไปอีก ให้กำหนด **rear** ไปชี้ที่
จุดเริ่มต้นใหม่



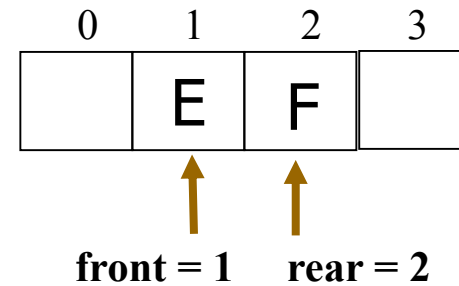
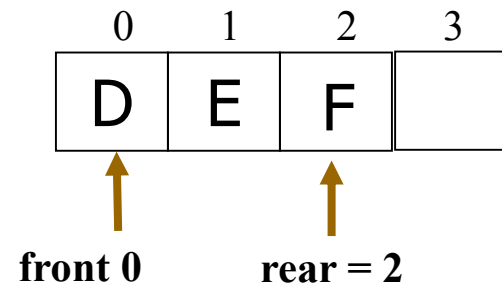
↑
front = 3
rear = 3

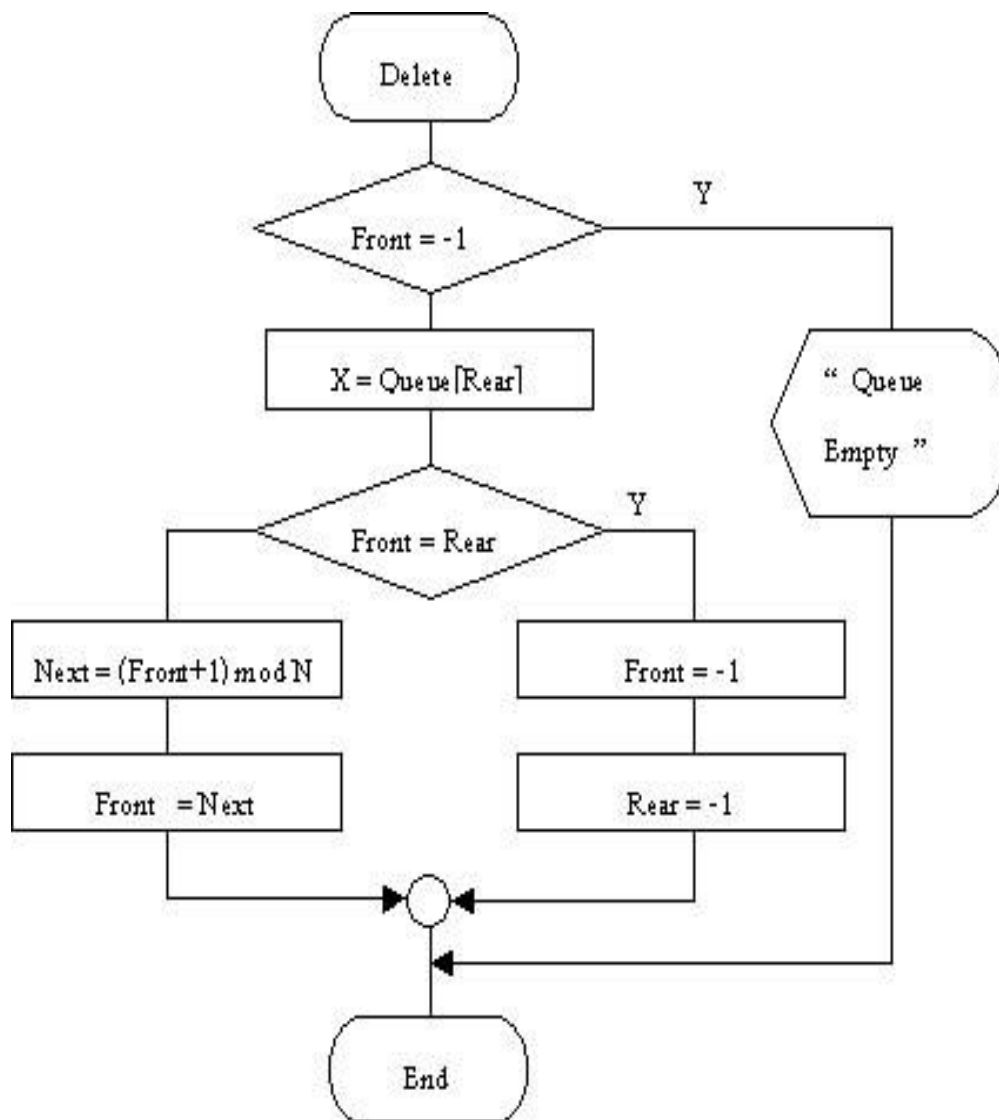
ลบข้อมูลในคิวแบบวงกลม

deq (que,data);



deq (que,data);



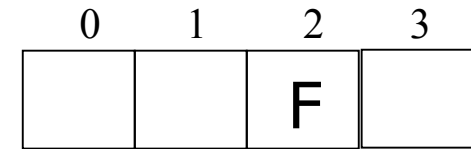


Algorithm Delete

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 ;  
}
```



↑
front = 2
rear = 2

ลบคิวที่เหลือเพียงตัวเดียว

ลบคิวที่ช่องความจำสุดท้าย กำหนดให้ **front** วนไปชี้ที่จุดเริ่มต้นใหม่

ตัวอย่างโปรแกรม Queue

```
1 #include <stdio.h>
2 #define MAX_QUEUE_SIZE 4
3 #define TRUE 1
4 #define FALSE 0
5 typedef struct int_queue_type {
6     int item[MAX_QUEUE_SIZE];
7     int front, back, count;
8 } Queue;
9
10 void initQueue(Queue *queue){
11     queue->count = 0;
12     queue->front = queue->back = 0;
13 }
14
15 int enQueue(Queue *queue, int x){
16     if (queue->count >= MAX_QUEUE_SIZE)
17         return FALSE;
18     else {
19         queue->count++;
20         queue->item[queue->back] = x;
21         queue->back = (queue->back+1) % MAX_QUEUE_SIZE;
22         return TRUE;
23     }
24 }
25
26 int deQueue(Queue *queue, int *x){
27     if (queue->count <= 0)
28         return FALSE;
29     else {
30         queue->count--;
31         *x = queue->item[queue->front];
32         queue->front = (queue->front + 1) % MAX_QUEUE_SIZE;
33         return TRUE;
34     }
35 }
36
37 void printQueue(Queue queue){
38     int i;
39     int pos = queue.front;
40
41     printf("Queue:");
42     for (i=0; i< queue.count; i++){
43         printf("%d", queue.item[pos]);
44         pos = (pos + 1) % MAX_QUEUE_SIZE;
45     }
46     printf("\n");
47 }
```

```
49 void main(){
50     Queue q;
51     int choice;
52     int cont = TRUE;
53     int x;
54
55     initQueue(&q);
56     while (cont == TRUE){
57         printf("Please select [1:enQueue 2:deQueue 3:printQueue 0:exit]:");
58         scanf("%d", &choice);
59         switch(choice){
60             case 1:
61                 printf("Please enter a number to enqueue:");
62                 scanf("%d", &x);
63                 if (!enQueue(&q,x))
64                     printf(" Error putting into the queue\n");
65                 break;
66             case 2:
67                 if (deQueue(&q,&x))
68                     printf("The number from the queue is: %d\n", x);
69                 else
70                     printf(" Error getting an item from the queue\n");
71                 break;
72             case 3:
73                 printQueue(q);
74                 break;
75             case 0:
76                 cont = FALSE;
77                 break;
78         }
79     }
80 }
```

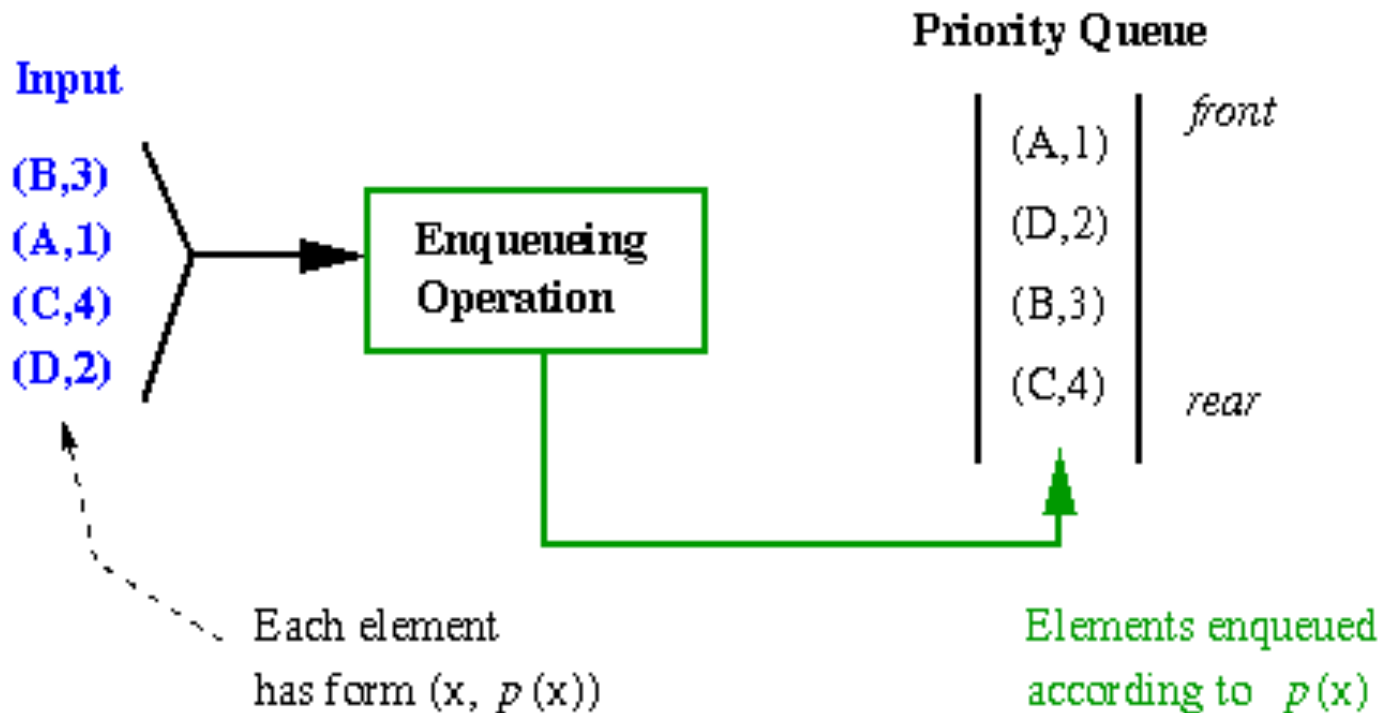
ตัวอย่างโปรแกรม Queue ภาษา C++

```
1  #include <iostream>           // std::cin, std::cout
2  #include <queue>               // std::queue
3  using namespace std;
4
5  int main ()
6  {
7      queue<int> myqueue;
8      int myint;
9
10     cout << "Please enter some integers (enter 0 to end):\n";
11
12     do {
13         cin >> myint;
14         myqueue.push (myint);
15     } while (myint);
16
17     cout << "myqueue contains: ";
18     while (!myqueue.empty()){
19         cout << ' ' << myqueue.front ();
20         myqueue.pop ();
21     }
22     cout << '\n';
23
24     return 0;
25 }
```

คิวลำดับความสำคัญ หรือ แถวคอยเชิงบูรุมภาพ (Priority Queue)

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

คิวลำดับความสำคัญ หรือ แถวคอยเชิงบูรุมภาพ (Priority Queue)



ตัวอย่างโปรแกรม Priority Queue ภาษา C++

```
1  #include <iostream>           // std::cout
2  #include <queue>               // std::priority_queue
3  using namespace std;
4
5  int main ()
6  {
7      priority_queue<int> mypq;
8
9      mypq.push(30);
10     mypq.push(100);
11     mypq.push(25);
12     mypq.push(40);
13
14     cout << "Popping out elements...";
15     while (!mypq.empty())
16     {
17         cout << ' ' << mypq.top();
18         mypq.pop();
19     }
20     cout << '\n';
21
22     return 0;
23 }
```

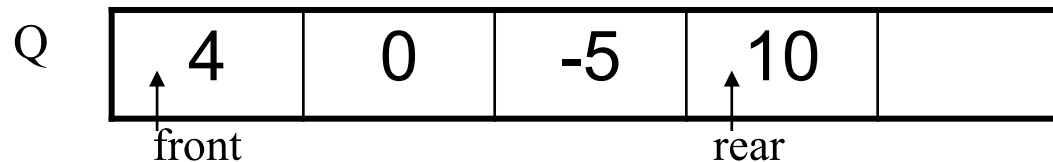
Exercise

เขียนโปรแกรมมีเงื่อนไขว่า ใครที่อายุ(Age)น้อยที่สุด จะถูกเรียกชื่อก่อนเสมอ
โดยหากมีหลายคนที่มีอายุเท่ากัน คนที่ถูกเพิ่มเข้าไปในคิวภายหลังจากถูกเรียกก่อน
เสมอ

<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. $\text{deq}(Q, x)$ | 2. $\text{enq}(Q, 2)$ |
| 3. $\text{deq}(Q, x)$ | 4. $\text{deq}(Q, x)$ |
| 5. $\text{enq}(Q, -3)$ | 6. $\text{deq}(Q, x)$ |

แบบฝึกหัด (ต่อ)

4. คิวขนาด 7 ค่า มีรายการต่อไปนี้

front = 2 , rear = 5

Queue[] = {null, null, “ชนุน”, “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