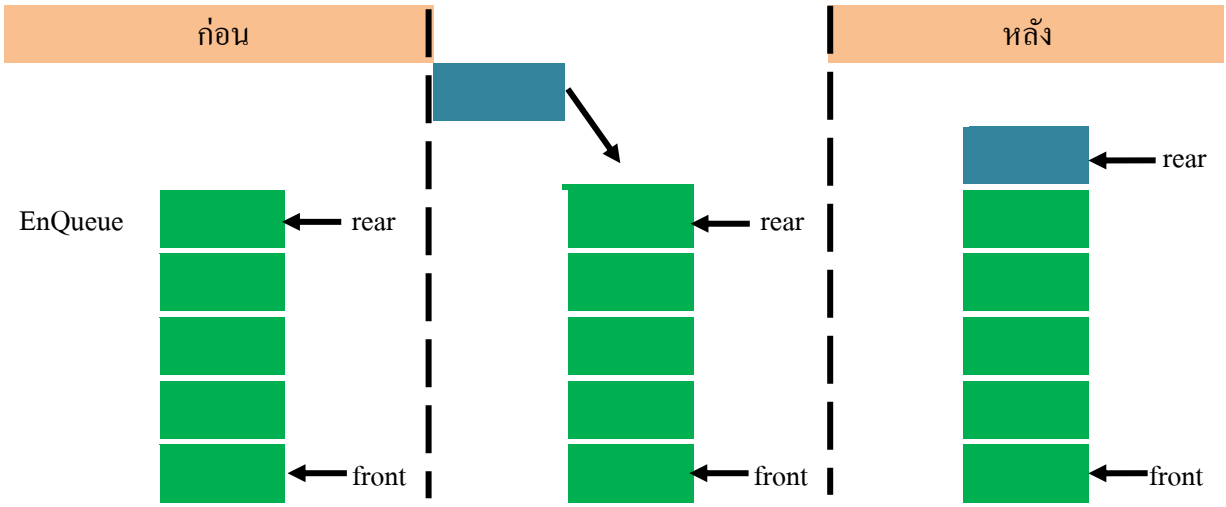
โครงสร้างข้อมูลแบบคิว (Queue)

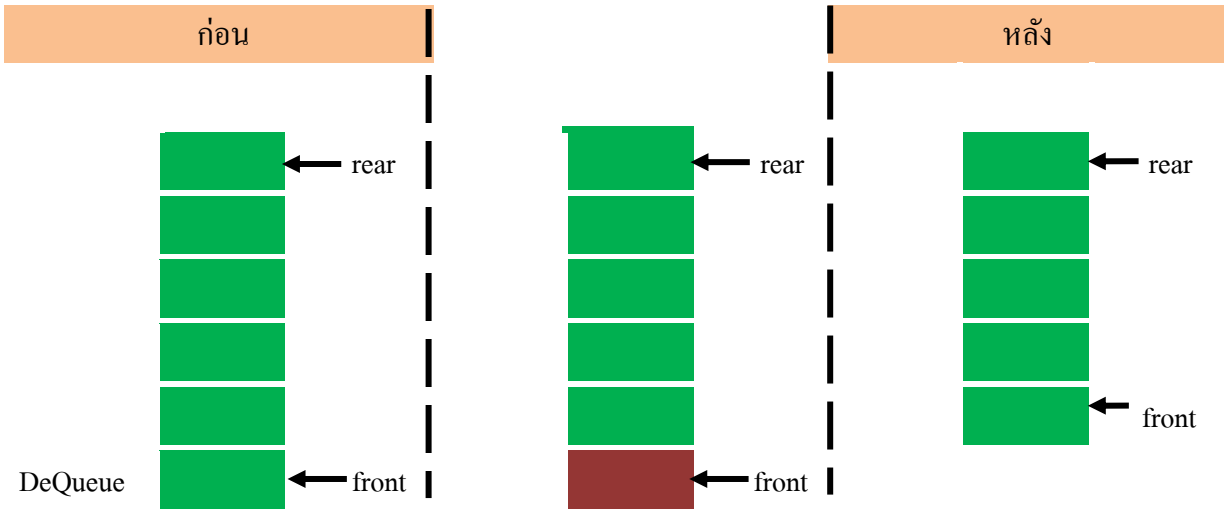
โครงสร้างข้อมูลแบบคิว (Queue) หรือแถวคอย จัดเป็นโครงสร้างข้อมูลเชิงนามธรรมแบบเชิงเส้น (Linear Abstract Data Type) แบบ FIFO (First – In – First - Out) นั้นคือข้อมูลที่เข้ามาก่อนจะถูกนำออกมาก่อน เพราะมีลักษณะคล้ายกับแถวเรียงต่อกัน และ โดยการทำงานกับโครงสร้างข้อมูลคิวนั้นจะมีการเข้าถึงข้อมูล 2 ทิศทางโดยการเพิ่มข้อมูลจะเข้าถึงที่ส่วนท้ายคิว และการนำข้อมูลออกจะเข้าถึงข้อมูลที่ส่วนหัวของคิว

การดำเนินการกับโครงสร้างข้อมูลแบบคิวมีเพียง 2 แบบ คือ

- การนำเข้ามูลเข้า (ENQUEUE) คือการเอาข้อมูลใส่เข้าไปในโครงสร้างแบบคิว โดยการทำงานจะทำที่ส่วนท้าย (rear หรือ tail) ของคิว



- การนำข้อมูลออก (DEQUEUE) คือการนำข้อมูลตัวที่อยู่ในตำแหน่งแรกสุดของข้อมูลออกจากโครงสร้างข้อมูลแบบคิว โดยการทำงานจะทำที่ส่วนหัว (front หรือ head) ของคิว



ตัวอย่างการทำงานของโครงสร้างข้อมูลแบบคิว

เริ่มต้นการทำงาน = Empty Queue

front

rear

[0] [1] [2] [3] [4] [5] [6] [7] [8] [9] [10]

- ENQUEUE(5)

front

rear

[0] [1] [2] [3] [4] [5] [6] [7] [8] [9] [10]

- ENQUEUE(12)

front rear

[0] [1] [2] [3] [4] [5] [6] [7] [8] [9] [10]

- ENQUEUE (9)
- ENQUEUE (20)
- ENQUEUE (32)

front rear

[0] [1] [2] [3] [4] [5] [6] [7] [8] [9] [10]

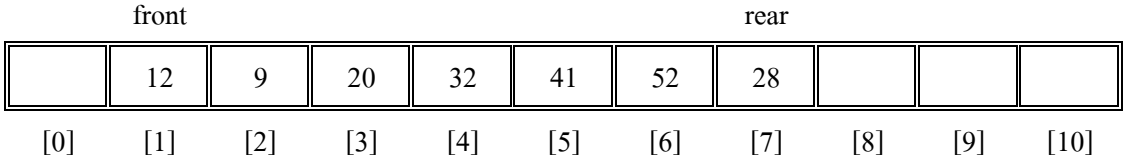
- DEQUEUE ( ) = 5

front rear

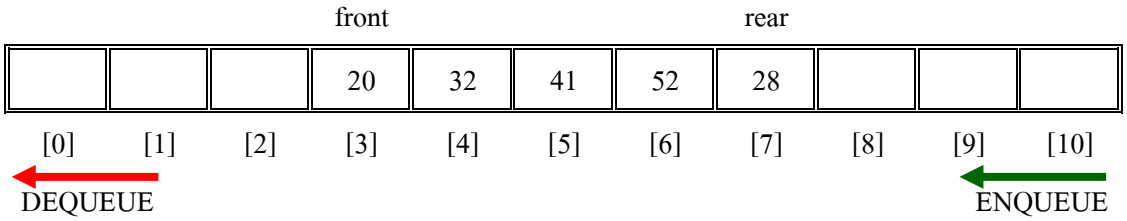
[0] [1] [2] [3] [4] [5] [6] [7] [8] [9] [10]

ตัวอย่างการทำงานของคิวแบบ Crawling Queue

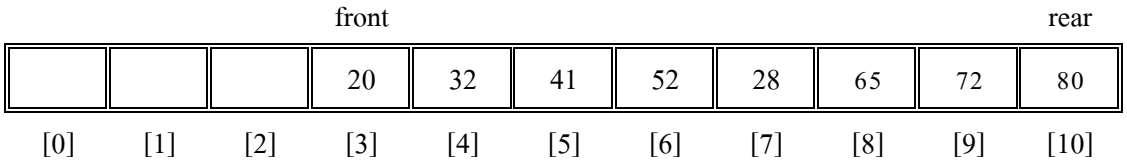
- ENQUEUE (41)
- ENQUEUE (52)
- ENQUEUE (28)



- DEQUEUE ( ) = 12
- DEQUEUE ( ) = 9



- ENQUEUE (65)
- ENQUEUE (72)
- ENQUEUE (80)

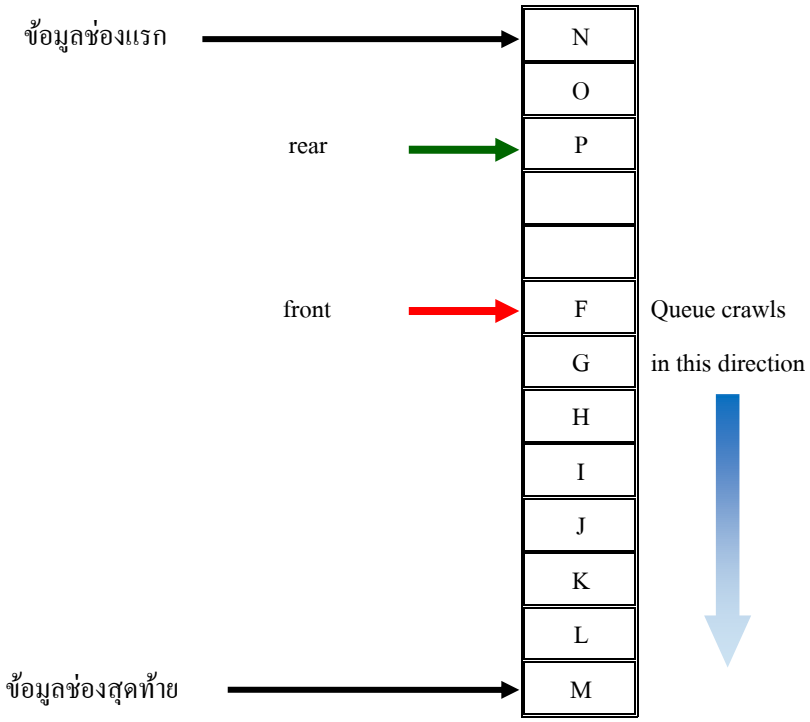


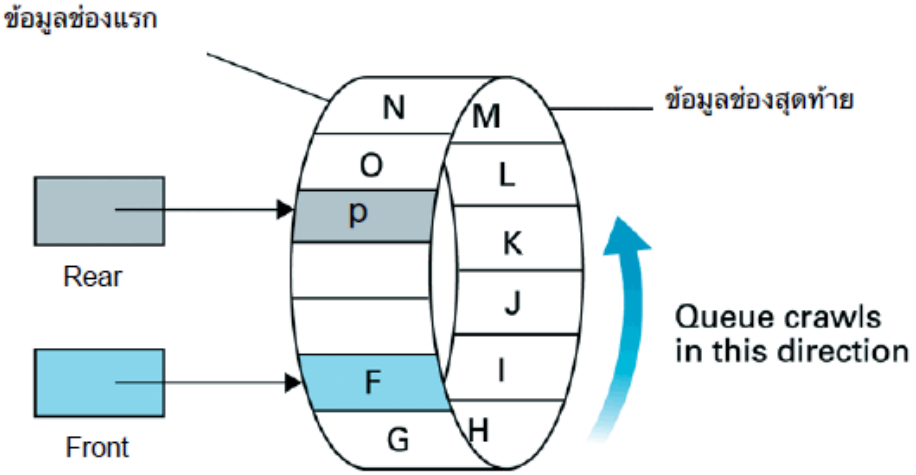
จากตัวอย่างการทำงานของโครงสร้างข้อมูลแบบคิวข้างต้นเป็นการทำงานของคิวแบบ Crawling Queue ซึ่งการเพิ่มข้อมูลสำหรับคิวในรูปแบบนี้เมื่อเพิ่มจนถึงตำแหน่งสุดท้ายของโครงสร้างแล้วจะไม่สามารถเพิ่มข้อมูลได้อีก เนื่องไม่สามารถกลับมาใช้พื้นที่ในส่วนหน้าได้อีก ดังนั้นจึงมีการปรับปรุงคิวเป็นโครงสร้างคิวแบบวงกลม (Circular Queue) ซึ่งเป็นโครงสร้างแบบที่สามารถกลับไปใช้พื้นที่ในส่วนหน้าของคิวได้อีกครั้งเมื่อมาถึงตำแหน่งสุดท้ายของคิวแล้ว

โครงสร้างข้อมูลแบบคิววงกลม (Circular Queue)

- คิววงกลม เป็นคิวที่มีดัชนี front ชี้ที่ส่วนหัว และดัชนี rear ชี้ที่ส่วนท้าย
- ถ้า front หรือ rear มีค่าเท่ากับ max แล้ว สามารถที่จะวนกลับมาชี้ที่ตำแหน่งแรกสุดได้
- Front มีค่ามากกว่า rear ได้ (คิวปกติค่า front มีค่าน้อยกว่าหรือเท่ากับค่า rear)

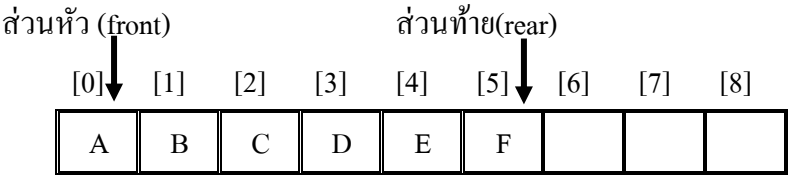
ตัวอย่างโครงสร้างข้อมูลแบบคิววงกลม





การสร้างโครงสร้างข้อมูลแบบคิว สามารถทำการสร้างได้โดยใช้โครงสร้างพื้นฐาน 2 รูปแบบดังนี้คือ

- สร้างโดยใช้พื้นฐานเป็นโครงสร้างแบบแถวลำดับ (Array – Based Implementation) เป็นการนำเอาโครงสร้างพื้นฐานแบบแถวลำดับมาใช้ในการพัฒนาโครงสร้างแบบคิว ซึ่งจะใช้การอ้างอิงตำแหน่งของตัวชี้ตำแหน่ง front และ rear ในลักษณะของดัชนี เพื่ออ้างอิงตำแหน่งของแถวลำดับ



การประกาศโครงสร้างแบบแถวลำดับสำหรับ Queue จากภาพข้างต้น ในภาษา C ทำได้ดังนี้

- char Queue[9] ;
- int front = -1;
- int rear = -1 ;

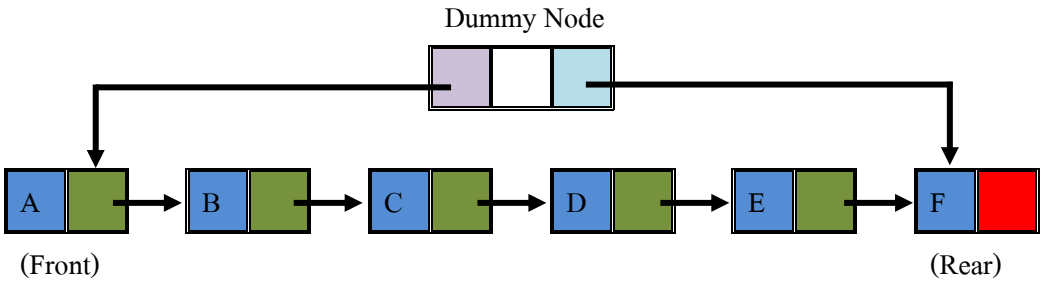
ฟังก์ชัน ENQUEUE สำหรับ Queue เมื่อใช้พื้นฐานของโครงสร้างข้อมูลแบบแถวลำดับ

```
void ENQUEUE (element_type x , element_type QUEUE [] )
{
        if (rear = = n-1)
           {
               printf(" Full Queue ") ;
               exit(1);
            }
          else
           {
               rear = rear + 1;
               QUEUE[rear] = x;
           }
}
```

ฟังก์ชัน DEQUEUE สำหรับ Queue เมื่อใช้พื้นฐานของโครงสร้างข้อมูลแบบแถวลำดับ

```
element_type  DEQUEUE  ( element_type  QUEUE [] )
{
        element_type  x;
         if ((front = = -1) || (front == n))
           {
                printf(“ Empty  Queue ”) ;
                exit(1);
            }
          else
           {
                x = QUEUE[front]
                front  = front + 1;
           }
          return(x)
}
```

- สร้างโดยใช้พื้นฐานเป็นโครงสร้างแบบลิสต์เชื่อมโยง (Linked – List Based Implementation)



การประกาศโครงสร้าง Linked List สำหรับ Queue จากภาพข้างต้น ในภาษา C ทำได้ดังนี้

```
typedef  struct  node
{
     element_type  element ;
     struct node  *next ;
 } NODE
หมายเหตุ : element_type คือ ชนิดข้อมูลเช่น int,
char, float , …
```

```
typedef  struct  dummy_node
{
     element_type  element ;
     struct node  *front ;
     struct node  *rear ;
}DUMMY_NODE
typedef  DUMMY_NODE  *QUEUE
```

ฟังก์ชัน ENQUEUE สำหรับ Queue เมื่อใช้พื้นฐานของโครงสร้างข้อมูล Linked List

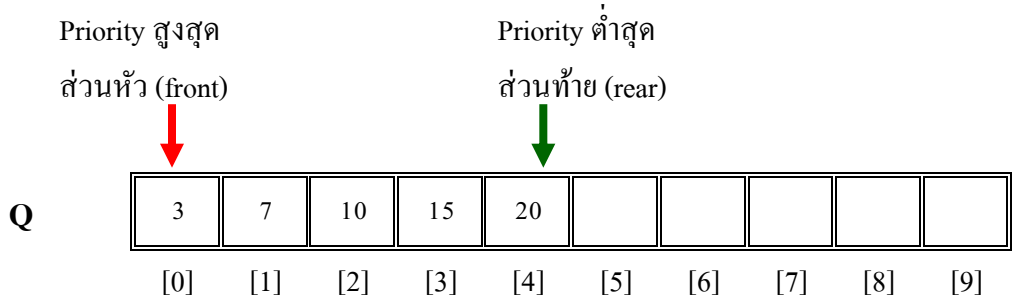
```
void  ENQUEUE  (element_type  x , QUEUE  Q )
{       NODE   *tmp;
        tmp = (NODE *) malloc (sizeof(NODE)) ;
        if (tmp = = NULL)
        {     printf("Out  of  Space ") ;
              exit(1) ;
        }
        if (is_empty (Q))
        {
              tmp -> element = x;
              tmp-> next = NULL;
             Q ->front = tmp;
             Q ->rear = tmp;
        }
       else
        {
              tmp -> element = x;
              tmp-> next = NULL;
             Q ->rear->next = tmp;
             Q ->rear = tmp;
        }
}
```

ฟังก์ชัน DEQUEUE สำหรับ Queue เมื่อใช้พื้นฐานของโครงสร้างข้อมูล Linked List

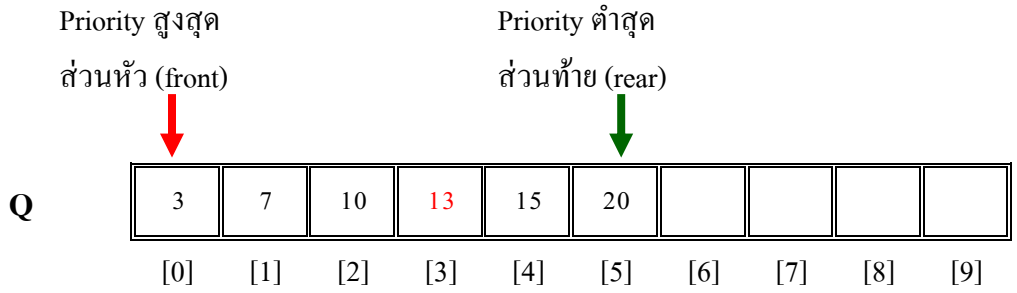
```
element_type  DEQUEUE  (QUEUE  Q )
{
        element_type   x;
        NODE   *tmp;
        if (is_empty (Q))
        {
              printf ("Empty Queue \n ");
              exit(1);
        }
       else
        {
             tmp  =  Q->front;
             x = tmp ->element ;
             Q->front =  Q ->front->next ;
             free(tmp) ;
            return(x)
        }
}
```

Priority Queue

- เป็นคิวรูปแบบที่จัดเก็บข้อมูลตามความสำคัญ (Priority) ของข้อมูล
- การกำหนดค่าความสำคัญ (Priority) ขึ้นอยู่กับประเภทของงาน
- ตัวอย่างเช่น กำหนดให้ข้อมูลที่มีค่าน้อย Priority สูงสุด และข้อมูลที่มีค่ามาก Priority ต่ำ



เพิ่ม 13



การประยุกต์ใช้โครงสร้างข้อมูลแบบคิว

ในระบบกาทำงานของคอมพิวเตอร์มี การนำโครงสร้างข้อมูลแบบคิวไปประยุกต์ใช้งานในหลายๆ ส่วน ด้วยกันเช่น

- Operating System นำมาใช้เพื่อการจัดสรรทรัพยากรของคอมพิวเตอร์ทังหมด เช่น อุปกรณ์ input อุปกรณ์ output รวมทั้ง CPU memory
- กรณีการส่งข้อมูลไปยัง Browser ที่เครื่อง Client ของ Web Server ซึ่งจะมารส่งข้อมูลไปให้เครื่อง client แต่ละเครื่องตามลำดับของคำขอ

ตัวอย่างโปรแกรม Priori Queue

```
 1
 2 void Penqueue(element_type x, QUEUE Q )
 3 {
 4     if (is_full(Q )) {
 5         printf("Full queue\n");
 6         exit(1);
 7     }
 8     else {
 9         Q->q_size++;
10         Q->q_rear = succ(Q->q_rear,Q );
11         Q->q_array[Q->q_rear] = x;
12     }
13
14     int i,tmp;
15     i = Q->q_rear;
16     while (i>Q->q_front&&Q->q_array[i-1]>Q->q_ar
17     {
18           tmp = Q->q_array[i-1];
19           Q->q_array[i-1] = Q->q_array[i];
20           Q->q_array[i] = tmp;
21           i--;
22     }
23
24 }
```