# 肆、佇列

## 一、佇列 (Queue) 簡介?

1. 何謂佇列(Queue)?

①是一種先進先出(First In First Out, FIFO)的有序串列

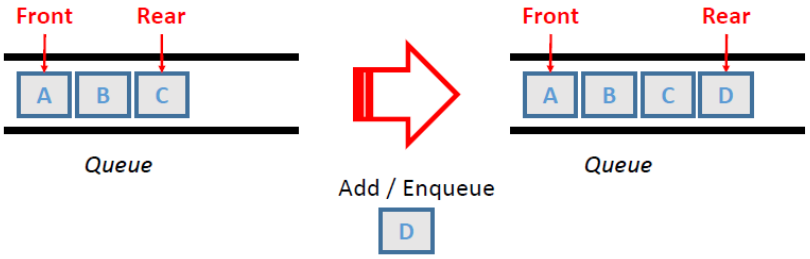
②資料由串列的一端加入，該端稱為佇列的尾端(rear)

③資料的取出，從佇列的另一端，該端稱為佇列的前端(front)

2. 佇列常用運算或操作

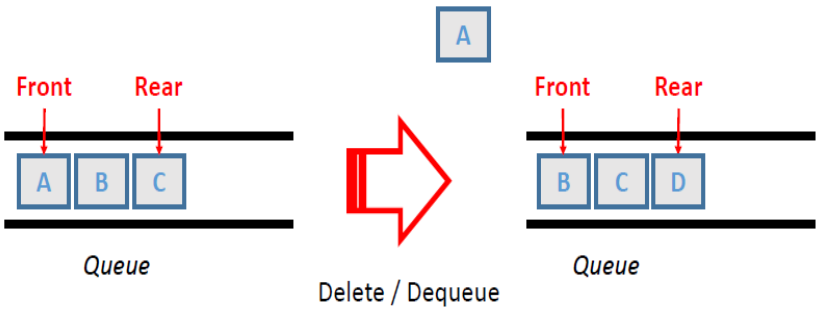
① **Add 或 Enqueue：**

從佇列尾端加入一筆新的資料，該筆資料成為在佇列的尾端



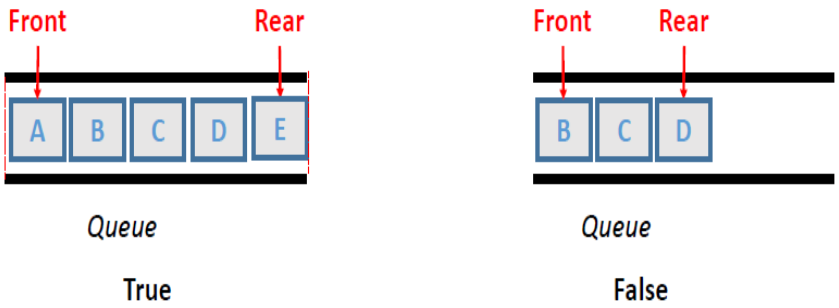

② **Delete 或 Dequeue：**

從佇列前端移出(或稱刪除)一筆資料，原本排在前端後的資料，變成在佇列的前端



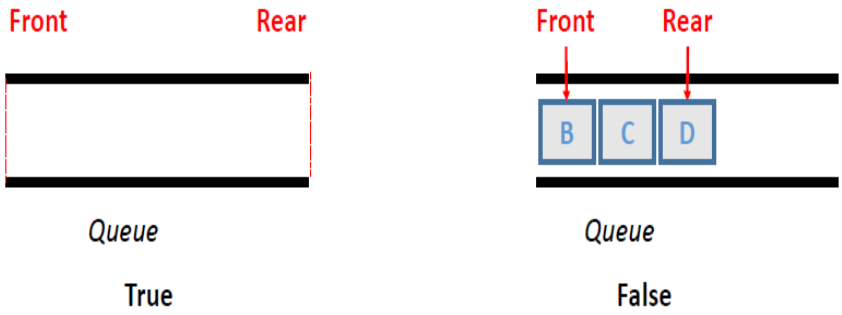

③ **IsFull：**

判斷佇列是否已滿，若是，回傳真(True)；反之，回傳假(False)



④ **IsEmpty**：

判斷佇列是否為空，若是，回傳真(True)；反之，回傳假(False)




## 二、以陣列實作佇列

1. 使用陣列來實作佇列

①宣告一個一維陣列，通常取名為 queue，陣列內可存放的資料數目 N，就是該佇列的可放入資料數目

②宣告一個整數變數，名稱為 front，其值表示佇列的前端所在，若佇列為空，其值為-1

③宣告一個整數變數，名稱為 rear，其值表示佇列的尾端所在，若佇列為空，其值為-1；若佇列已滿，其值為 N-1

- 實現 **IsFull()** 運算的演算法

```
Procedure IsFull(Queue)
Begin
if (Rear==N-1)
return True; //Queue is full.
else
return False; //Queue is not full.
End
```

- 實現 **IsEmpty()** 運算的演算法

```
Procedure IsEmpty(Queue)
Begin
if( rear == -1 and front == -1 )
return True; //Queue is empty.
else
return False; //Queue is not empty.
End
```

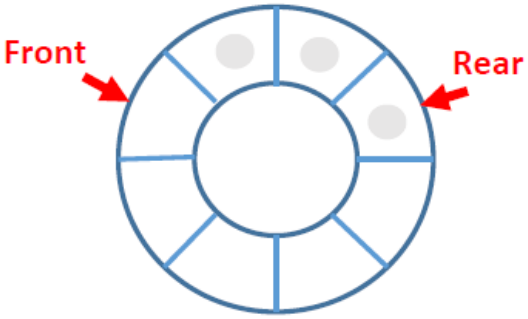
- **實現 Add()運算的演算法**

```
Procedure Add(Queue, item)
Begin
if ( isFull(Queue) ) return "Queue is full, Add item is
failure.";
else
Rear = Rear + 1;
Queue[Rear] = item;
return Queue;
End
```

- **實現 Delete()運算的演算法**

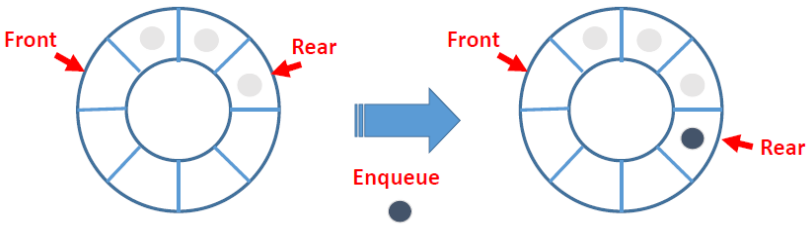
```
Procedure Delete(Queue, item)
Begin
if ( IsEmpty(Queue) ) return "Queue is empty, delete is
failure."
else
item = Queue[Front] ;
if ( Front <> Rear )//佇列尾端不等於佇列前端，也就是移出前，佇
                    列非只剩一個元素。
Do_Loopk=Front, k<rear, k++//佇列內元素，往前移動一格
Queue[k] = Queue[k+1] ;
else
front = -1 ;
rear = rear - 1 ;// 調整尾端
End
```

## 三、環形佇列

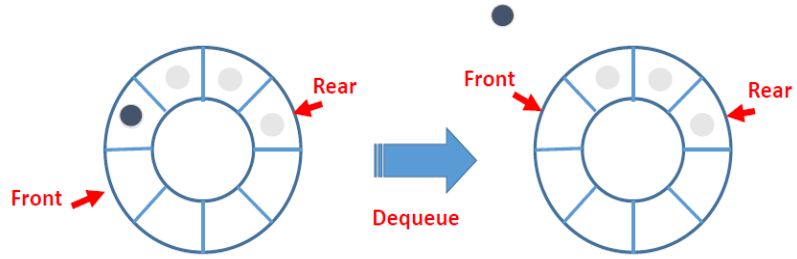



1. 環形佇列(Circular Queue)：可以解決之前佇列每次資料刪除後，需要搬移造成計算量增加的問題

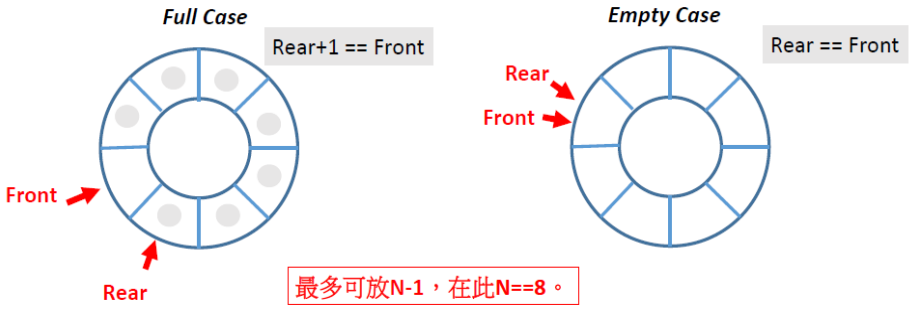
①環形佇列新增資料



②環形佇列刪除資料




③環形佇列滿與空的情況




2. 以一維陣列實現大小為 N-1 的環形佇列

①宣告一個大小為 N 的 1 維陣列，通常名稱為 circule_queue

②宣告一個名稱為 Front 的整數變數，初值為 0

③宣告一個名稱為 Rear 的整數變數，初值為 0

※課堂程式講解

```
#include <stdio.h>
#include <stdlib.h>

#define M 5
typedef struct{
    double data[M];
    int front;
    int rear;
}CQUEUE;

_Bool isCQEmpty( CQUEUE q );
_Bool isCQFull( CQUEUE q );
int Enqueue ( CQUEUE *q, double item );
int Dequeue ( CQUEUE *q, double *item );
```

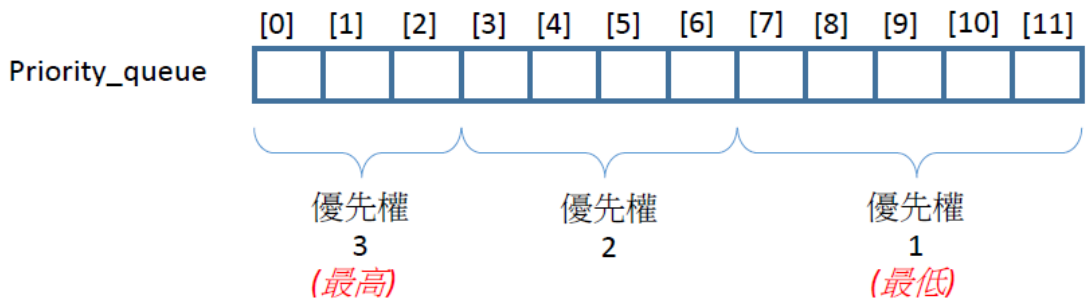
```c
int main(void) {
    int i=0;
    double item=0.0;
    CQUEUE q1;
    /*Initialize cqueue*/
    q1.rear = 0;
    q1.front = 0;
    const int repeatTimes = 2;
    printf("\n");
    for(int t=0; t<repeatTimes; t++){
        printf("Test Times:%d\n",t+1);

        /*Test Empty function*/
        if( isCQEmpty(q1))printf("The queue is Empty!\n");
        else printf("The queue is not Empty!\n");

        /*Make queue full*/
        for(i=0; i<M-1; i++){
            item = (double)(rand()%10);
            printf("Before enqueue, front:%d, rear:%d\n",q1.front,q1.rear);
            if(Enqueue( &q1, item )==1)printf("Queue Full! Enquque Failure!!\n");
            else printf("Add %f into queue.\n",item);
            printf("After enqueue, front:%d, rear:%d\n",q1.front,q1.rear);
        }
        if( isCQFull(q1))printf("\nThe queue is Full!\n");
        else printf("\nThe queue is not Full!\n");

        for(i=0; i<M-1; i++){
            printf("Before dequeue, front:%d, rear:%d\n",q1.front,q1.rear);
            if(Dequeue( &q1, &item)==1)printf("Queue Empty! Dequeue Failure!\n");
            else printf("Take %f from queue.\n",item);
            printf("After dequeue, front:%d, rear:%d\n",q1.front,q1.rear);

        }
        if( isCQEmpty(q1))printf("The queue is Empty!\n");
        else printf("The queue is not Empty!\n");
    }
    return EXIT_SUCCESS;
}
```

```
_Bool isCQEmpty(CQUEUE q){
    if(q.front == q.rear)return 1;
    else return 0;
}

_Bool isCQFull(CQUEUE q){
    int r;
    r = (q.rear+1)%M;
    if( r == q.front )return 1;
    else return 0;
}

int Enqueue(CQUEUE*q, double item){
    if(isCQFull(*q)) return 1;
    else{
        q->rear = (q->rear + 1)%M;
        q->data[q->rear] = item;
        return 0;
    }
}

int Dequeue(CQUEUE *q,double *item){
    if(isCQEmpty(*q))return 1;
    else{
        q->front = (q->front + 1)%M;
        *item = q->data[q->front];
        return 0;
    }
}
```

## 四、優先佇列

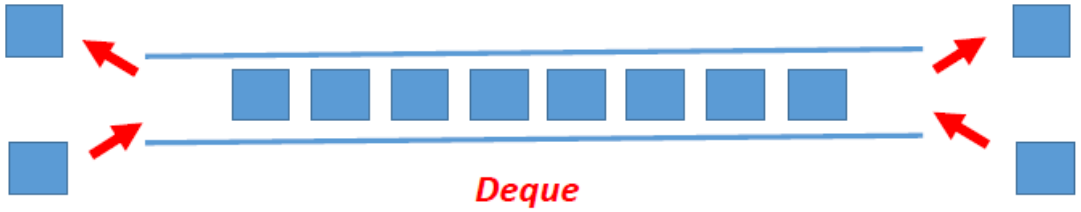
1.優先佇列(Priority Queue)：一種有序串列，其資料離開佇列，非依據先進先出原則，而是依據佇列元素的優先權，優先權高的先出，若優先權相同，才依據先進先出原則

2.以陣列實現優先權佇列：使用一維陣列，其內部依據優全權分隔成數個區塊，每個區塊相當於一個一般的佇列，優先權高的一般佇列空時，才換次低優先權佇列送出資料



# 五、雙向佇列

1. 雙向佇列(Double-ended Queue, Deque)：雙向佇列是一種特殊的資料結構，他的兩端都可以作為加入與取出資料的動作，不像 Stack 的 LIFO 或 Queue 的 FIFO。也就是它是一種前後兩端都可輸入或取出資料的有序串列




2. Deque 使用例

- 放入資料

・假設循序放入 1, 2, 3，則依據放入的方式，在 Deque 內的結果可能有：
・1, 2, 3
・2, 1, 3
・3, 1, 2
・3, 2, 1

- 取出資料

・若 Deque 內有資料：5, 1, 2, 3, 4
・依據取出的方式不同，可能產生的取出結果有：
・4, 5, 1, 3, 2
・5, 1, 2, 3, 4
・5, 4, 1, 2, 3
・5, 1, 4, 2, 3
・5, 1, 4, 3, 2
・……

※但不可能有：1, 5, 4, 2, 3