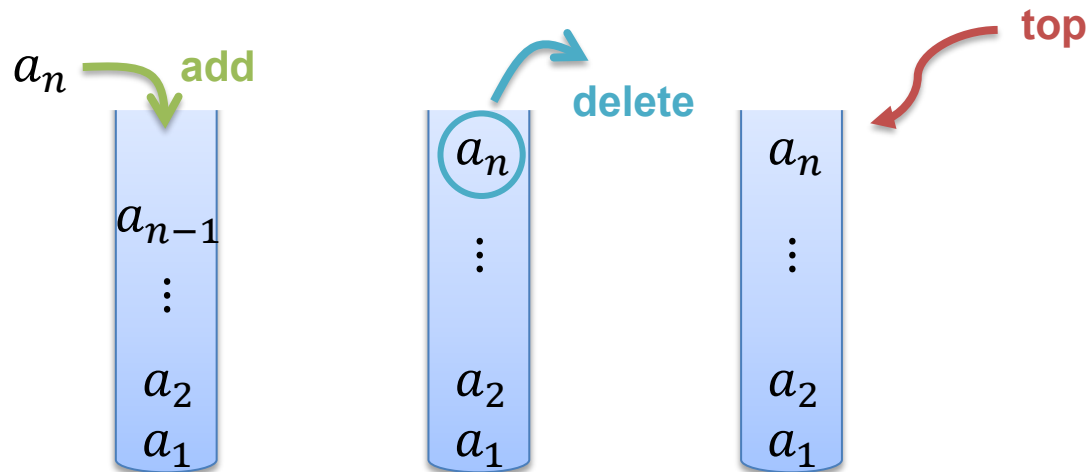


Queues

Kuan-Yu Chen (陳冠宇)

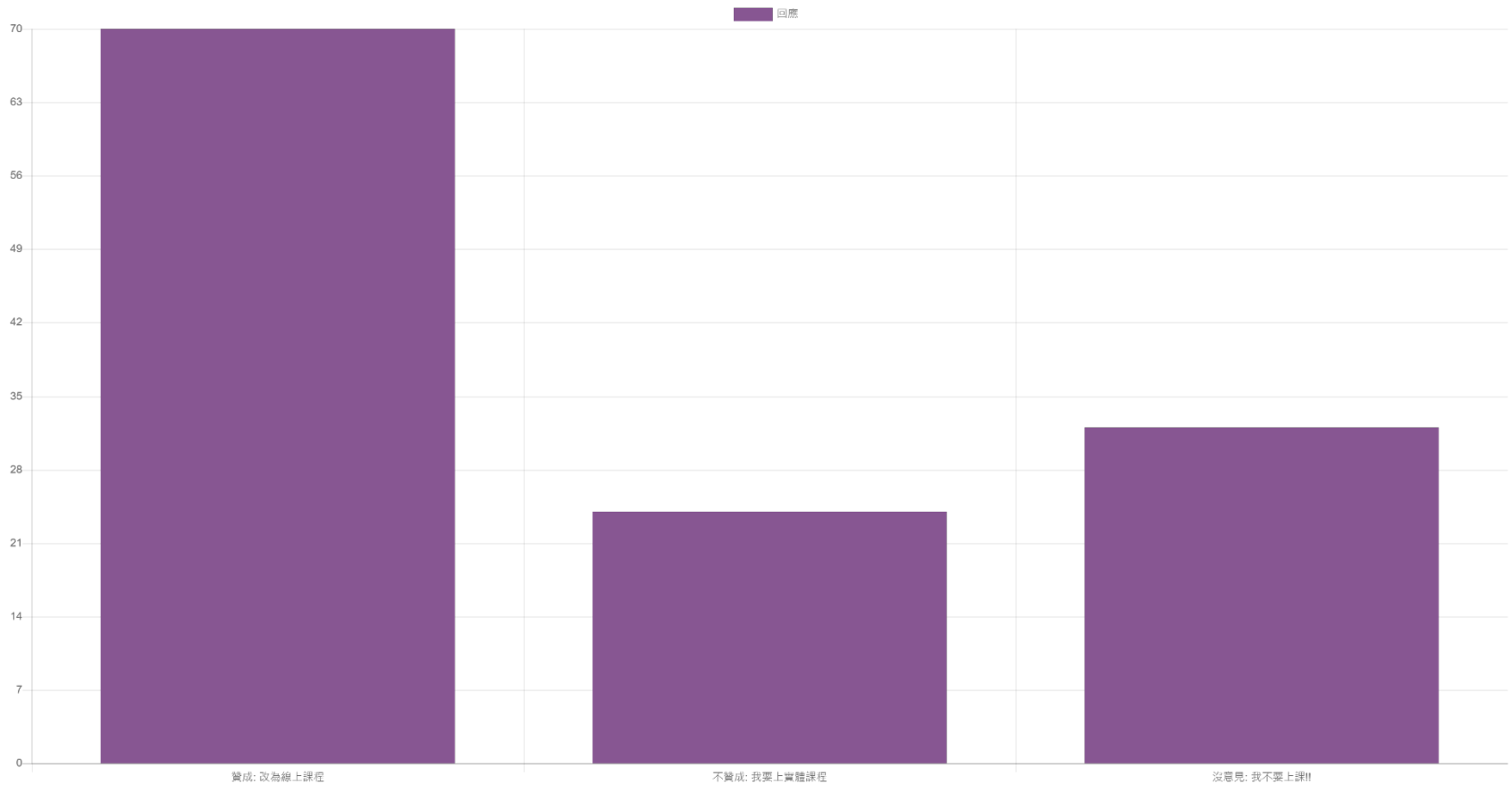
Review

- Stack
 - A **stack** is an **ordered** list in which insertions and deletions are made at one end called the **top**
 - Last in first out
 - First in last out
 - Stack Permutation



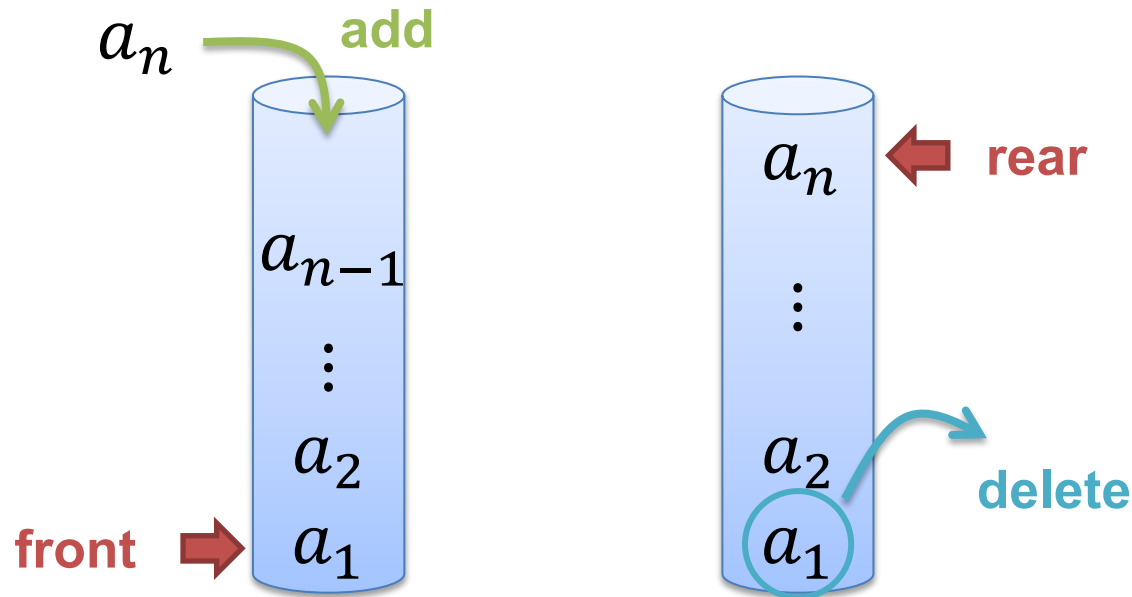
Wednesday Online!

- <https://ctld.webex.com/meet/kychen>



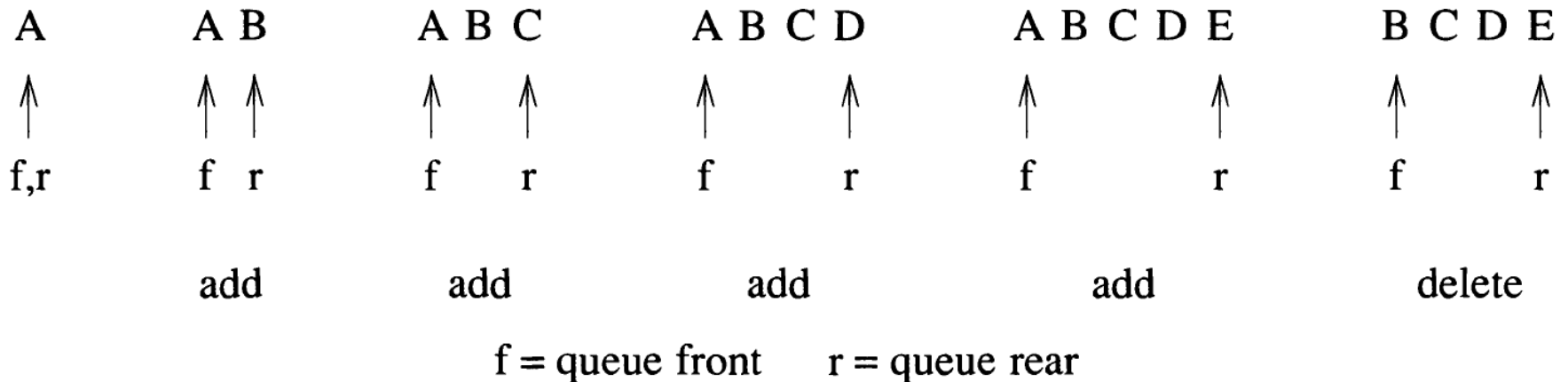
Queue.

- A **queue** is an **ordered** list in which insertions take place at one end (**rare**) and deletions are made at the opposite end (**front**)
 - Given a queue $Q = (a_1, a_2, \dots, a_n)$
 - a_1 is the front element
 - a_n is the rear element
 - a_i is behind element a_{i-1}



Queue..

- By the definition of queue, if we insert the elements A, B, C, D, E in the order, then A is the first element deleted from the queue
 - First-In-First-Out**



Applications – Queue

- Job scheduling
 - A fair method

front	rear	Q[0]	Q[1]	Q[2]	Q[3]	Comments
-1	-1					queue is empty
0	0	J1				Job 1 is added
0	1	J1	J2			Job 2 is added
0	2	J1	J2	J3		Job 3 is added
1	2		J2	J3		Job 1 is deleted
2	2			J3		Job 2 is deleted

Array Representation of Queues

- Queues can be easily represented using arrays
 - Given a queue

12	9	7	18	14	36				
0	1	2	3	4	5	6	7	8	9

- Insert an element

12	9	7	18	14	36	45			
0	1	2	3	4	5	6	7	8	9

rear
↓

- Delete an element

front
↓

	9	7	18	14	36	45			
0	1	2	3	4	5	6	7	8	9

Implementation for Queue by Array.

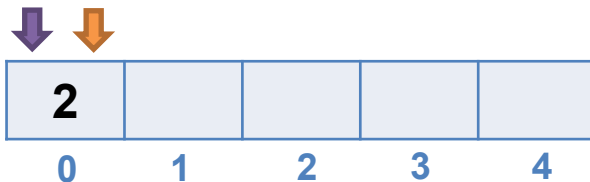
- Declare

```
#define MAX 10 // Changing this value will change length of array
int queue[MAX];
int front = -1, rear = -1;
void insert(void);
int delete_element(void);
void display(void);
```

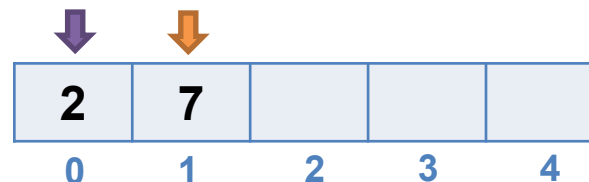

Implementation for Queue by Array..

```
void insert()
{
    int num;
    printf("\n Enter the number to be inserted in the queue : ");
    scanf("%d", &num);
    if(rear == MAX-1)
        printf("\n OVERFLOW");
    else if(front == -1 && rear == -1)
        front = rear = 0;
    else
        rear++;
    queue[rear] = num;
}
```

front rear

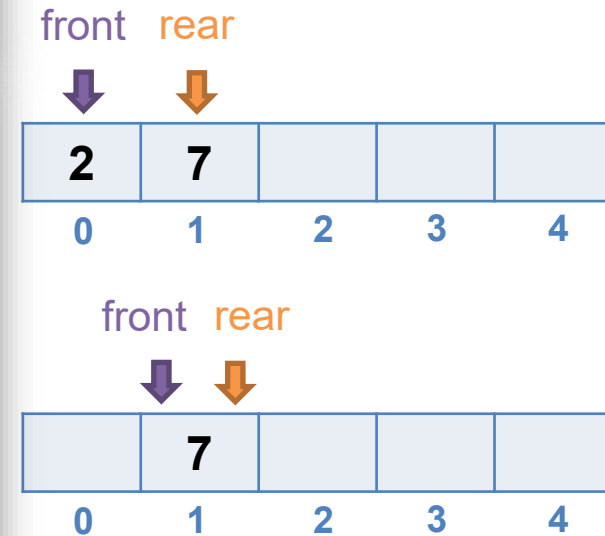


front rear



Implementation for Queue by Array...

```
int delete_element()
{
    int val;
    if(front == -1)
    {
        printf("\n UNDERFLOW");
        return -1;
    }
    else
    {
        val = queue[front];
        front++;
        if(front > rear)
            front = rear = -1;
        return val;
    }
}
```



Implementation for Queue by Array....

```
void display()
{
    int i;
    printf("\n");
    if(front == -1)
        printf("\n QUEUE IS EMPTY");
    else
    {
        for(i = front; i <= rear; i++)
            printf("\t %d", queue[i]);
    }
}
```

Types of Queues

- Actually a queue structure can be classified into four types
 - Circular Queue
 - Deque
 - Priority Queue
 - Multiple Queue

Circular Queue.

- Given a queue
 - If you want to insert another value, it will not be possible because the queue is completely full

54	9	7	18	14	36	45	21	99	72
0	1	2	3	4	5	6	7	8	9

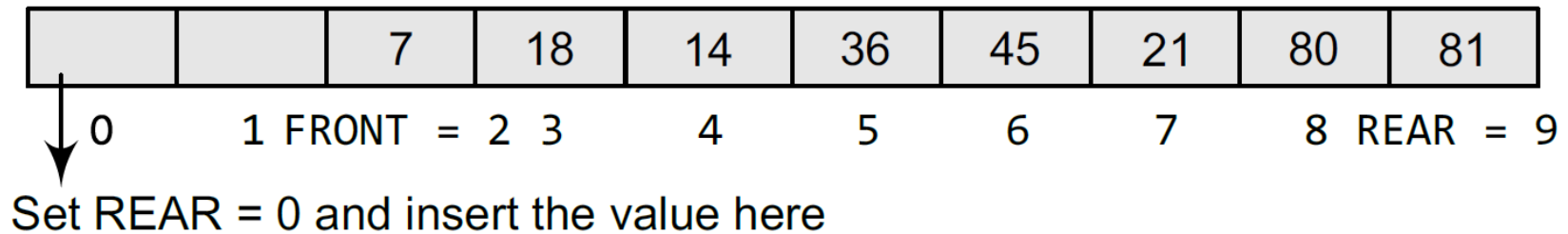
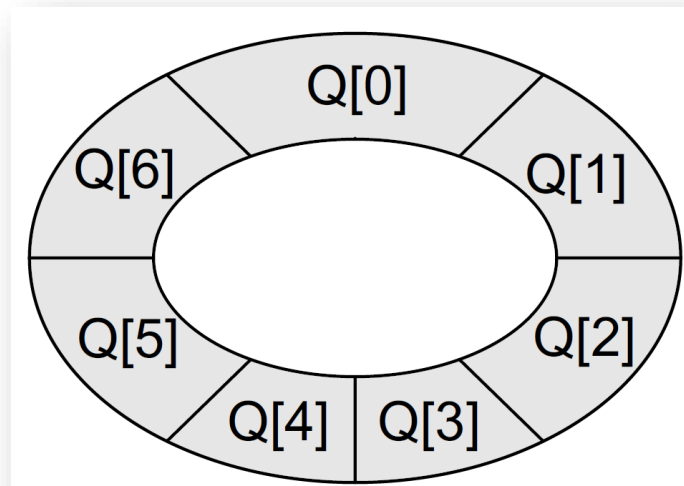
- Let's delete two elements from the queue

		7	18	14	36	45	21	99	72
0	1	2	3	4	5	6	7	8	9

- Even though there is space available, we still can not insert elements in the queue
 1. Shift the elements to the left so that the vacant space can be occupied and utilized efficiently
 2. Circular queue!

Circular Queue..

- A circular queue is implemented by using array



Deque.

- A deque (pronounced as “deck” or “dequeue”) is a list in which the elements can be inserted or deleted at either end
 - Double-ended queue
 - It is also known as a **head-tail linked list** because elements can be added to or removed from either the front (head) or the back (tail) end
 - No element can be added and deleted from the middle
- In a deque, two pointers are maintained, **LEFT** and **RIGHT**, which point to either end of the deque

			29	37	45	54	63		
0	1	2	LEFT = 3	4	5	6	RIGHT = 7	8	9

42	56						63	27	18
0	RIGHT = 1	2	3	4	5	6	LEFT = 7	8	9

Deque..

- There are two variants of a double-ended queue
 - Input restricted deque
 - In this queue, insertions can be done only at one of the ends, while deletions can be done from both ends
 - Output restricted deque
 - In this queue, deletions can be done only at one of the ends, while insertions can be done on both ends

Priority Queue.

- A priority queue is a data structure in which each element is assigned a priority
 - The priority of the element can be set based on various factors
- The priority of the element will be used to determine the order in which the elements will be processed
 - An element with higher priority is processed before an element with a lower priority
 - Two elements with the same priority are processed on a first-come-first-served (FCFS) basis
- Priority queues are widely used in operating system
 - The priority of the process may be set based on the CPU time it requires to get executed completely
 - Brake override system, BOS

Priority Queue..

- Array Representation of a Priority Queue
 - Each priority number has its own queue
 - The queue is usually implemented by **circular queue**
 - Every individual queue will have its own FRONT and REAR pointers

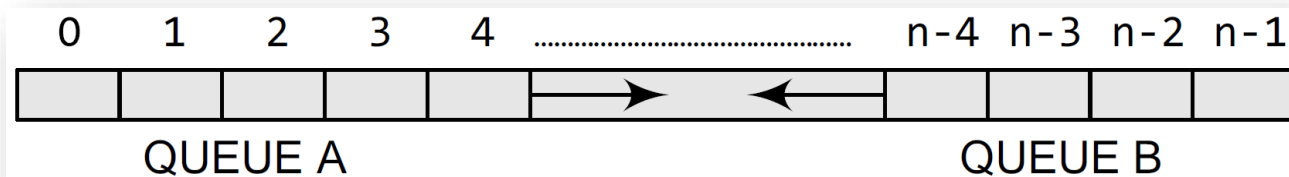
FRONT	REAR	1	2	3	4	5
3	3	1		A		
1	3	2	B	C	D	
4	5	3			E	F
4	1	4	I		G	H

- If we have to insert an element R with priority number 3, then the priority queue will become

FRONT	REAR	1	2	3	4	5
3	3	1		A		
1	3	2	B	C	D	
4	1	3	R		E	F
4	1	4	I		G	H

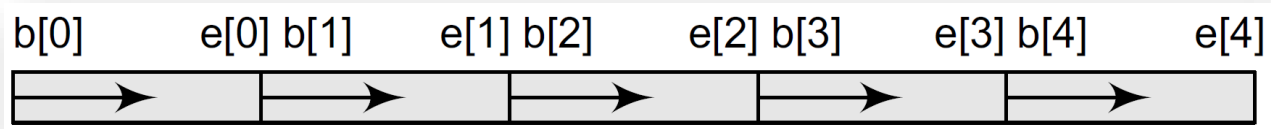
Multiple Queue.

- When we implement a queue using an array, the size of the array must be known in advance
 - In case we allocate a large amount of space for the queue, it will result in sheer wastage of the memory
- A better solution to deal with this problem is to have multiple queues or to have more than one queue in the same array of sufficient size
 - Queue *A* will grow from left to right, whereas queue *B* will grow from right to left at the same time

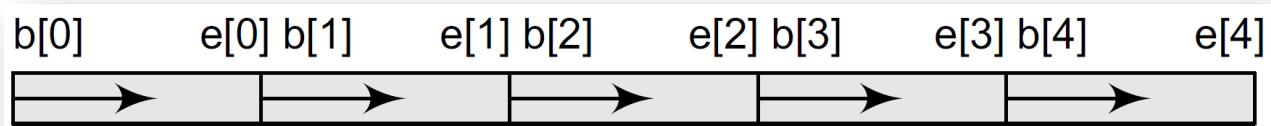


Multiple Queue..

- Extending the concept to multiple queues, another multiple queue has been introduced



- The concept of multiple queue can be extended to implement the “multiple stack”



Questions?



kychen@mail.ntust.edu.tw