# Object Oriented Programming

# Assignment 1

## Submit separate cpp files in zip format. File name should be in the format of your roll number 23L-1234.cpp

## Question no. 1 Marks: 20

You are given a 2D list of integers, where each element may be positive, negative, or zero. Your task is to write a program that prints a new 2D list containing only the positive elements of the original list, and also calculates the sum of all positive elements in the new list. The number of rows and columns in the original list is determined by the user. Assume that rows and columns of the input array are defined by the user

| | |
|---|---|
| 2 | 3 |
| 0 | 0 |
| -3 | 0 |
| 11 | -12 |
| -55 | 0 |

Output Array:

| | |
|---|---|
| 2 | 3 |
| 11 | |

Sum: 16

# Question no. 2 Marks: 60

Write a program in C++ to implement dynamic memory allocation for a 2D maze game. The game should allow a player to move from one cell to another in the maze and reach the end. The player should be able to move in any of the four directions (up, down, left, right) as long as there is no wall in that direction. The maze should be randomly generated at the start of each game, using dynamic memory allocation. The game should keep track of the player's position and the number of moves taken to reach the end of the maze. The game should also check if the player has reached the end of the maze and display a path to the player indicating whether they won or lost the game. You need to start from top left and reach till bottom right.

**Example:**

1 1 0 0 0

0 1 1 1 0

1 1 0 1 0

0 0 1 1 0

0 0 1 1 1

**Path 1** => **right, down, right, right, down, down, left, down, right, right**

**Path 2** => **right, down, right, right, down, down, down, right**

**Example:**

1 1 0 0 0

0 1 1 1 0

1 1 0 1 0

0 0 1 0 0

0 0 1 1 1

**Path** => **no path exists**

Your task is to print shortest path as in first example is path 2.

# Question no. 3 Marks: 20

You are given a file named **input.txt** with values Yes and No. Read the file and insert the strings in a 2D array of **strings**, where the elements are either "Yes" or "No". Do not use extra space. You must calculate number of rows and columns after reading the file. Now, your task is to compress the input array by counting the number of "Yes" values and creating a new array, which will contain the count of "Yes" values in the first column and the string "No" in the second column. Then, you need to use the compressed array to transform it back to the original input array. You can use another 2D array as an intermediate representation. Additionally, you need to calculate the number of "Yes" values in the original array and print the result. Make an input function to make Input 2D array from user and fill it up

| | | |
|---|---|---|
| Yes | Yes | Yes No No No |
| Yes | No | No No Yes No |
| Yes | No | No No No No |
| Yes | Yes | Yes No No Yes |
| Yes | No | No No Yes No |

Output Array:

3 3

2 4

1 5

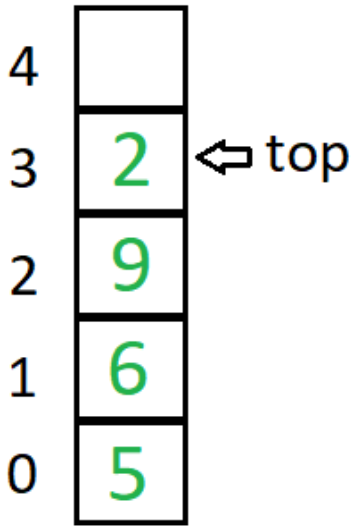
4 2

2 4

In this 2D array the first row will contain the location of “Yes”. For example, the first row will look like this 1, 2, 3. Which represents that “Yes” in first row are at location 1, 2 and 3 and rests of the elements are “No”. **Use string not cstring.**

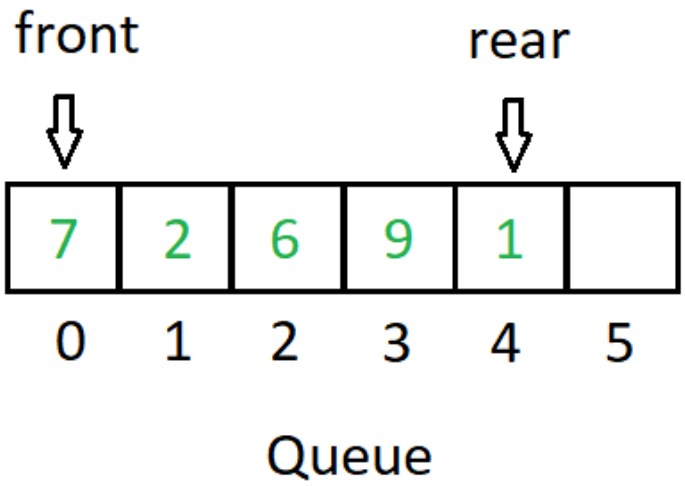
## Question no. 4 Marks: 50

**Stack:** A stack is a linear data structure in which elements can be inserted and deleted only from one side of the list, called the **top**. A stack follows the **LIFO** (Last In First Out) principle, i.e., the element inserted at the last is the first element to come out. The insertion of an element into the stack is called **push** operation, and the deletion of an element from the stack is called **pop** operation. In stack, we always keep track of the last element present in the list with a pointer called **top**.
The diagrammatic representation of the stack is given below:




Stack

Queue is a linear data structure in which elements can be inserted only from one side of the list called **rear**, and the elements can be deleted only from the other side called the **front**. The queue data structure follows the **FIFO** (First In First Out) principle, i.e. the element inserted at first in the list, is the first element to be removed from the list. The insertion of an element in a queue is called an **enqueue** operation and the deletion of an element is called a **dequeue** operation. In queue, we always maintain two pointers, one pointing to the element which was inserted at the first and still present in the list with the **front** pointer and the second pointer pointing to the element inserted at the last with the **rear** pointer.
The diagrammatic representation of the queue is given below:




now your job is to make two classes named as stack and queues. Each class will have only 1 dynamic array which will be updated on basic operations.

Stack class will have following functions: **(20 Marks)**

- **push(int data)** to insert an element into the stack
- **pop()** to remove an element from the stack
- **top()** Returns the top element of the stack.
- **isEmpty()** returns true if stack is empty else false.
- **size()** returns the size of stack.

Queue class will have the following functions: **(30 Marks)**

- **enqueue():** Inserts an element at the end of the queue i.e. at the rear end.
- **dequeue():** This operation removes and returns an element that is at the front end of the queue.
- **front():** This operation returns the element at the front end without removing it.
- **rear():** This operation returns the element at the rear end without removing it.
- **isEmpty():** This operation indicates whether the queue is empty or not.
- **isFull():** This operation indicates whether the queue is full or not.
- **size():** This operation returns the size of the queue i.e. the total number of elements it contains.

# Note:

- You need to traverse arrays in all questions using '*' symbol. Do not use [] to traverse arrays.
- Do not use any built in classes in any question.