CSE225L – Data Structures and Algorithms Lab
Lab 09
Queue (Array-based)

In today's lab we will design and implement the Queue ADT using array.

queuetype.h

```
#ifndef QUEUETYPE_H
#define QUEUETYPE_H

const int SIZE = 100;

// Exception class thrown by Enqueue when the queue is full
class FullQueue
{};

// Exception class thrown by Dequeue when the queue is empty
class EmptyQueue
{};

template<class T>
class QueueType
{
private:
    T* data;
    int front;
    int rear;
    int size;
public:
    QueueType();
    QueueType(int s);
    ~QueueType();
    void MakeEmpty();
    bool IsEmpty();
    bool IsFull();
    void Enqueue(T);
    void Dequeue(T&);
};
#endif // QUEUETYPE_H
```

queuetype.cpp

```
#include "queuetype.h"
#include <iostream>
using namespace std;

template<class T>
QueueType<T>::QueueType()
{
    data = new T[SIZE];
    front = rear = 0;
}

template<class T>
QueueType<T>::QueueType(int s)
{
    size = s + 1;
    data = new T[size];
    front = rear = 0;
}
```

```cpp
template<class T>
QueueType<T>::~QueueType()
{
    delete [] data;
}

template<class T>
void QueueType<T>::MakeEmpty()
{
    front = rear = 0;
}

template<class T>
bool QueueType<T>::IsEmpty()
{
    return (front == rear);
}

template<class T>
bool QueueType<T>::IsFull()
{
    return ((rear + 1) % size == front); // Full if next position of rear equals front
}

template<class T>
void QueueType<T>::Enqueue(T value)
{
    try
    {
        if (IsFull())
        {
            throw FullQueue();
        }
        else
        {
            data[rear] = value;       // Store the new value at the rear
            rear = (rear + 1) % size; // Move rear to next position (circularly)
        }
    }
    catch (FullQueue e)
    {
        cout << "Queue Overflow" << endl;
    }
}

template<class T>
void QueueType<T>::Dequeue(T &value)
{
    try
    {
        if (IsEmpty())
            throw EmptyQueue();
        else
        {
            value = data[front];        // Retrieve the value from the front
            front = (front + 1) % size; // Move front to next position (circularly)
        }
    }
    catch (EmptyQueue e)
    {
        cout << "Queue Underflow" << endl;
    }
}
```

Generate the **driver file (main.cpp)** where you perform the following tasks. Note that you cannot make any change to the header file or the source file.

Operation to Be Tested and Description of Action	**Input Values**	**Expected Output**
Create a queue of integers of size 5		
Print if the queue is empty or not		Queue is Empty
Enqueue four items	5, 7, 4, 2	
Print if the queue is empty or not		Queue is not Empty
Print if the queue is full or not		Queue is not full
Enqueue another item	6	
Print the values in the queue (in the order the values are given as input)		5, 7, 4, 2, 6
Print if the queue is full or not		Queue is Full
Enqueue another item	8	Queue Overflow
Dequeue two items		
Print the values in the queue		4, 2, 6
Dequeue three items		
Print if the queue is empty or not		Queue is Empty
Dequeue an item		Queue Underflow
Take an integer **n** from the user as input and use a queue to print binary values of each integer from 1 to **n**. Here is how it can be done. • Create an empty queue • Enqueue the first binary number "1" to the queue. • Now run a loop for generating and printing n binary numbers. • Dequeue and print the value. • Append "0" at the dequeued value and enqueue it. • Append "1" at the dequeued value and enqueue it.	10	1 10 11 100 101 110 111 1000 1001 1010