North South University
CSE-225L(Data Structures & Algorithm)
Summer - 2018
Lab-09 (Queue – Array Based)

Class "QueType":

quetype.h

```
#ifndef QUETYPE_H_INCLUDED
#define QUETYPE_H_INCLUDED
#include <iostream>
using namespace std;

class FullQueue{ // A Dummy Class to be thrown as Exception};

class EmptyQueue{// A Dummy Class to be thrown as Exception};

template<class DataType>
class QueType
{
public:
     QueType(int);
     ~QueType();
     void MakeEmpty();
     bool IsEmpty();
     bool IsFull();
     void Enqueue(DataType);
     void Dequeue();
     DataType Peak();
private:
     int front; // pointing to front item in the
                          //queue
     int rear; // pointing to rear item in the
                          //queue
     DataType* info; // items will point to
                               // the array where queue
                               // items will be stored
     int maxQue; // will define maximum size
                               //of the array
     };
#endif // QUETYPE_H_INCLUDED
```

quetype.cpp

```
#include "quetype.h"

template<class DataType>
QueType<DataType>::QueType(int max)
{
    maxQue=max;
    front= -1;
    rear= -1;
    info = new DataType[maxQue];
}
```

```
template<class DataType>
QueType<DataType>::~QueType()
{
    delete[] info;
}

template<class DataType>
void QueType<DataType>::MakeEmpty()
{
    front= -1;
    rear= -1;
}

template<class DataType>
bool QueType<DataType>::IsEmpty()
{
   return (front == -1);
}

template<class DataType>
bool QueType<DataType>::IsFull()
{
   return ((rear+1)%maxQue==front);
}

template<class DataType>
void QueType<DataType>::Enqueue(DataType i)
{
    if(IsFull())
    {
       throw FullQueue();
    }

    else
    {
         rear = (rear+1)%maxQue;
         info[rear] = i;
         if (front == -1)
         {
           front=0;
         }
     }

}
```

```
template<class DataType>
void QueType<DataType>::Dequeue()
{
     if (front == -1)
     {
          cout<< “Queue is Empty”<<endl;
     }
     if (front == rear)
     {
                MakeEmpty();
     }
     else
     {
                front = (front+1)%maxQue;
     }
}


template<class DataType>
DataType QueType<DataType>::Peak()
{
    return info[front];
}


template class QueType<int>; // so CodeBlocks can compile the
                  // template for int type data

template class QueType<double>;// so CodeBlocks can compile the
                             // template for double type data

template class QueType<char>;// so CodeBlocks can compile the
                             // template for char type data
```