**CSE225L – Data Structures and Algorithms Lab**

**Lab 07**

**Queue (array based)**

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

|  |  |
| --- | --- |
| **quetype.h**  #ifndef QUETYPE\_H\_INCLUDED  #define QUETYPE\_H\_INCLUDED  class FullQueue  {};  class EmptyQueue  {};  template<class ItemType>  class QueType  {  public:  QueType();  QueType(int max);  ~QueType();  void MakeEmpty();  bool IsEmpty();  bool IsFull();  void Enqueue(ItemType);  void Dequeue(ItemType&);  private:  int front;  int rear;  ItemType\* items;  int maxQue;  };  #endif // QUETYPE\_H\_INCLUDED  **quetype.cpp**  #include "quetype.h"  template<class ItemType>  QueType<ItemType>::QueType(int max)  {  maxQue = max + 1;  front = maxQue - 1;  rear = maxQue - 1;  items = new ItemType[maxQue];  }  template<class ItemType>  QueType<ItemType>::QueType()  {  maxQue = 501;  front = maxQue - 1;  rear = maxQue - 1;  items = new ItemType[maxQue];  } | template<class ItemType>  QueType<ItemType>::~QueType()  {  delete [] items;  }  template<class ItemType>  void QueType<ItemType>::MakeEmpty()  {  front = maxQue - 1;  rear = maxQue - 1;  }  template<class ItemType>  bool QueType<ItemType>::IsEmpty()  {  return (rear == front);  }  template<class ItemType>  bool QueType<ItemType>::IsFull()  {  return ((rear+1)%maxQue == front);  }  template<class ItemType>  void QueType<ItemType>::Enqueue(ItemType newItem)  {  if (IsFull())  throw FullQueue();  else  {  rear = (rear +1) % maxQue;  items[rear] = newItem;  }  }  template<class ItemType>  void QueType<ItemType>::Dequeue(ItemType& item)  {  if (IsEmpty())  throw EmptyQueue();  else  {  front = (front + 1) % maxQue;  item = items[front];  }  } |