CSE225L – Data Structures and Algorithms Lab
Lab 09
Queue (Linked List)

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

quetype.h

```
#ifndef QUETYPE_H_INCLUDED
#define QUETYPE_H_INCLUDED class
FullQueue
{};
class EmptyQueue
{};
template <class ItemType> class
QueType
{
    struct NodeType
    {
        ItemType info;
        NodeType* next;
    };     public:
QueType();
        ~QueType();         void
MakeEmpty();         void
Enqueue(ItemType);         void
Dequeue(ItemType&);         bool
IsEmpty();         bool
IsFull();     private:
        NodeType *front, *rear;
};

#endif // QUETYPE_H_INCLUDED
```

quetype.cpp

```
#include "quetype.h"
#include <iostream> using
namespace std;
 template <class ItemType>
QueType<ItemType>::QueType()
{
    front = NULL;
rear = NULL;
}
template <class ItemType>
bool QueType<ItemType>::IsEmpty()
{
    return (front == NULL);
}
template<class ItemType>
bool QueType<ItemType>::IsFull()
{
    NodeType* location;
try     {
        location = new NodeType;
delete location;         return
false;
    }
    catch(bad_alloc& exception)
    {
        return true;
    }
}
template <class ItemType>
void QueType<ItemType>::Enqueue(ItemType newItem)
{
    if (IsFull())
throw FullQueue();
else
    {
        NodeType* newNode;
newNode = new NodeType;
newNode->info = newItem;
newNode->next = NULL;
if (rear == NULL)
front = newNode;         else
            rear->next = newNode;
rear = newNode;
    }
}
template <class ItemType>
void QueType<ItemType>::Dequeue(ItemType& item)
{
    if (IsEmpty())
throw EmptyQueue();
else
    {
        NodeType* tempPtr;
tempPtr = front;
item = front->info;
front = front->next;
if (front == NULL)
rear = NULL;         delete
tempPtr;
    }
}
template <class ItemType>
void QueType<ItemType>::MakeEmpty()
{
    NodeType* tempPtr;
while (front != NULL)
    {
        tempPtr = front;
front = front->next;
delete tempPtr;
    }
    rear = NULL;
} template <class
ItemType>
QueType<ItemType>::~QueType()
{
    MakeEmpty();
}
```

Generate the **Driver file (main.cpp)** and check your program with the following outputs:

Operation to Be Tested and Description of Action	**Input Values**	**Expected Output**
Create a queue object		
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 the values in the queue (in the order the values are given as input)		**5 7 4 2**
Print if the queue is full or not		queue is not full
Dequeue an item		
Given a set of coin values and an amount of money Determine the minimum number of coins to make the given amount of money. The input starts with an integer n specifying the number of coin types. Next n integers are the coin values. The final integer is the amount of money you have to make. You can assume that the amount will always be possible to make using the given coin types.	**3 2 3 5 11** **3 5 20 30 40**	**3** **2**