LAB 12

CSE225L

Queue (Linked List—Based)

In this lab, we will:

- Design and implement the Queue ADT using a linked list—based circular structure.
- Create the **QueueType** class with methods for Enqueue, Dequeue, and checking the queue's status (IsFull, IsEmpty).
- Test the queue by inserting and removing elements, and handling queue overflow and underflow scenarios.
- Determine the **minimum number of coins** required to make a target amount using the given coin denominations.

QUEUE (LINKED LIST–BASED)

queuetype.h

```
#ifndef QUEUETYPE_H
#define QUEUETYPE_H

class FullQueue
{};

class EmptyQueue
{};

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

queuetype.cpp

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

template <class T>
QueueType<T>::QueueType()
{
    front = NULL;
    rear = NULL;
}

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

```
template<class T>
bool QueueType<T>::IsFull()
{
    try
    {
        Node* temp = new Node;
        delete temp;
        return false;
    }
    catch (bad_alloc& exception)
    {
        return true;
    }
}

template <class T>
void QueueType<T>::Enqueue(T value)
{
    if (IsFull())
    {
        throw FullQueue();
    }
    else
    {
        Node* temp = new Node;
        temp->data = value;
        temp->next = NULL;

        if (rear == NULL)
            front = temp;
        else
            rear->next = temp;
        rear = temp;
    }
}

template <class T>
void QueueType<T>::Dequeue(T& value)
{
    if (IsEmpty())
        throw EmptyQueue();
    else
    {
        Node* temp = front;
        value = front->data;
        front = front->next;
        if (front == NULL)
            rear = NULL;
        delete temp;
    }
}

template <class T>
void QueueType<T>::MakeEmpty()
{
    Node* temp;
    while (front != NULL)
    {
        temp = front;
        front = front->next;
        delete temp;
    }
    rear = NULL;
}

template <class T>
QueueType<T>::~QueueType()
{
    MakeEmpty();
}
```

QUEUE (LINKED LIST–BASED)

TASKS:

Instructions:

- Create the driver file (main.cpp) and perform the following tasks.
- You cannot make any changes to the header (**.h**) or source (**.cpp**) files of the **QueueType** class.

OPERATION	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

TASK	DESCRIPTION
Problem	Given a set of **n** coin values and a target amount, determine the **minimum number of coins** required to make the target amount. The target amount is always possible to make using the given coin types.
Example 1	**Input:** `3 2 3 5 11` **Explanation:** You have `3` types of coin: `2, 3, 5`, and need to make `11`. The optimal way is `3 + 3 + 5` coins. **Expected Output:** Minimum number of coins needed: `3`
Example 2	**Input:** `3 5 20 30 40` **Explanation:** You have `3` types of coin: `5, 20, 30`, and need to make `40`. The optimal way is `20 + 20` coins. **Expected Output:** Minimum number of coins needed: `2`
Example 3	**Input:** `3 2 3 5 200` **Explanation:** You have `3` types of coin: `2, 3, 5`, and need to make `200`. The optimal way is `5 * 40 = 200` coins. **Expected Output:** Minimum number of coins needed: `40`