```cpp
#include <iostream>
#include <stdlib.h> //srand, rand
#include <time.h>//clock_t, clock, CLOCKS_PER_SEC
using namespace std;

// implementing a doubly linked list-based queue


class Node{

    private:
        int data;
        Node* nextNodePtr;
        Node* prevNodePtr;

    public:
        Node(){}

        void setData(int d){
            data = d;
        }

        int getData(){
            return data;
        }

        void setNextNodePtr(Node* nodePtr){
                nextNodePtr = nodePtr;
        }

        Node* getNextNodePtr(){
            return nextNodePtr;
        }

        void setPrevNodePtr(Node* nodePtr){
                prevNodePtr = nodePtr;
        }

        Node* getPrevNodePtr(){
            return prevNodePtr;
        }

};

class Queue{

    private:
        Node* headPtr;
        Node* tailPtr;


    public:
        Queue(){
            headPtr = new Node();
            tailPtr = new Node();
            headPtr->setNextNodePtr(0);
            tailPtr->setPrevNodePtr(0);
        }

        Node* getHeadPtr(){
            return headPtr;
        }

        Node* getTailPtr(){
```

```
65              return tailPtr;
66          }
67
68          bool isEmpty(){
69
70              if (headPtr->getNextNodePtr() == 0)
71                  return true;
72
73              return false;
74          }
75
76
77          void enqueue(int data){
78
79              Node* newNodePtr = new Node();
80              newNodePtr->setData(data);
81              newNodePtr->setNextNodePtr(0);
82
83              Node* lastNodePtr = tailPtr->getPrevNodePtr();
84
85              if (lastNodePtr == 0){
86
87                  headPtr->setNextNodePtr(newNodePtr);
88                  newNodePtr->setPrevNodePtr(0);
89
90              }
91              else{
92
93                  lastNodePtr->setNextNodePtr(newNodePtr);
94                  newNodePtr->setPrevNodePtr(lastNodePtr);
95
96              }
97
98              tailPtr->setPrevNodePtr(newNodePtr);
99
100         }
101
102
103         int dequeue(){
104
105             Node* firstNodePtr = headPtr->getNextNodePtr();
106             Node* nextNodePtr = 0;
107
108             int poppedData = -100000; //empty queue
109
110             if (firstNodePtr != 0){
111                 nextNodePtr = firstNodePtr->getNextNodePtr();
112                 poppedData = firstNodePtr->getData();
113             }
114             else
115                 return poppedData;
116
117             if (nextNodePtr != 0){
118                 nextNodePtr->setPrevNodePtr(0);
119                 headPtr->setNextNodePtr(nextNodePtr);
120             }
121             else{
122                 headPtr->setNextNodePtr(0);
123                 tailPtr->setPrevNodePtr(0);
124             }
125
126             return poppedData;
127
128         }
```

```
129
130
131         int peek(){
132
133             Node* firstNodePtr = headPtr->getNextNodePtr();
134
135             if (firstNodePtr != 0)
136                 return firstNodePtr->getData();
137             else
138                 return -100000; //empty queue
139
140         }
141
142
143 };
144
145 int main(){
146
147     Queue queue;
148
149     int queueSize;
150
151     cout << "Enter the number of elements you want to enqueue: ";
152     cin >> queueSize;
153
154     srand(time(NULL));
155
156     int maxValue;
157
158     cout << "Enter the maximum value for an element: ";
159     cin >> maxValue;
160
161     cout << "Elements enqueued: ";
162     for (int i = 0; i < queueSize; i++){
163
164         int value = rand() % maxValue;
165         queue.enqueue(value);
166         cout << value << " ";
167     }
168
169     cout << endl;
170
171     cout << "Elements dequeued: ";
172     while (!queue.isEmpty()){
173
174         cout << queue.dequeue() << " ";
175     }
176
177     cout << endl;
178
179 return 0;
180 }
```

```
Enter the number of elements you want to enqueue: 10
Enter the maximum value for an element: 50
Elements enqueued: 42 40 1 11 46 19 46 41 40 5
Elements dequeued: 42 40 1 11 46 19 46 41 40 5
```