```h
#ifndef H_linkedQueue
#define H_linkedQueue

#include <iostream>

using namespace std;


template<class Type>
class linkedQueueType
{
public:
    const linkedQueueType<Type>& operator=
        (const linkedQueueType<Type>&);
    // overload the assignment operator
    bool isEmptyQueue();
    bool isFullQueue();
    void destroyQueue();
    void initializeQueue();
    void addQueue(const Type& newElement);
    void deQueue(Type& deqElement);
    linkedQueueType(); //default constructor
    linkedQueueType(const linkedQueueType<Type>& otherQueue);
        //copy constructor

    void printQueue();

    ~linkedQueueType(); //destructor

private:
    nodeType<Type>* front; //pointer to the front of the queue
    nodeType<Type>* rear;  //pointer to the rear of the queue
};


template<class Type>
linkedQueueType<Type>::linkedQueueType() //default constructor
{
    front = NULL; // set front to null
    rear = NULL;  // set rear to null
}


template<class Type>
bool linkedQueueType<Type>::isEmptyQueue()
{
    return(front == NULL);
}

template<class Type>
bool linkedQueueType<Type>::isFullQueue()
{
```

```
53      return false;
54  }
55
56  template<class Type>
57  void linkedQueueType<Type>::destroyQueue()
58  {
59      nodeType<Type>* temp;
60
61      while (front != NULL)  //while there are elements left in the queue
62      {
63          temp = front;         // set temp to point to the current node
64          front = front->link; // advance front to the next node
65          delete temp;          // deallocate memory occupied by temp
66      }
67
68      rear = NULL;  // set rear to null
69  }
70
71  template<class Type>
72  void linkedQueueType<Type>::initializeQueue()
73  {
74      destroyQueue();
75  }
76
77  template<class Type>
78  void linkedQueueType<Type>::addQueue(const Type& newElement)
79  {
80      nodeType<Type>* newNode;
81
82      newNode = new nodeType<Type>;   //create the node
83      newNode->info = newElement;     //store the info
84      newNode->link = NULL;           //initialize the link field to null
85
86      if (front == NULL)              //if initially queue is empty
87      {
88          front = newNode;
89          rear = newNode;
90      }
91      else                //add newNode at the end
92      {
93          rear->link = newNode;
94          rear = rear->link;
95      }
96  }//end addQueue
97
98  template<class Type>
99  void linkedQueueType<Type>::deQueue(Type& deqElement)
100 {
101     nodeType<Type>* temp;
102
103     deqElement = front->info;  //copy the info of the first element
104
```

```
105     cout << "deQueued item is " << deqElement << endl;
106
107     temp = front;              //make temp point to the first node
108     front = front->link;       //advance front to the next node
109     delete temp;               //delete the first node
110
111     if (front == NULL)          //if after deletion the queue is empty
112         rear = NULL;           //set rear to NULL
113 }//end deQueue
114
115
116
117 template<class Type>
118 linkedQueueType<Type>::~linkedQueueType() //destructor
119 {
120     nodeType<Type>* temp;
121
122     while (front != NULL)     //while there are elements left in the queue
123     {
124         temp = front;         //set temp to point to the current node
125         front = front->link; //advance first to the next node
126         delete temp;          //deallocate memory occupied by temp
127     }
128
129     rear = NULL;  // set rear to null
130 }
131
132 template<class Type>
133 const linkedQueueType<Type>& linkedQueueType<Type>::operator=
134 (const linkedQueueType<Type>& otherQueue)
135 {
136     //Write the definition of to overload the assignment operator
137
138 }
139
140 //copy constructor
141 template<class Type>
142 linkedQueueType<Type>::linkedQueueType(const linkedQueueType<Type>& otherQueue)
143 {
144     //Write the definition of the copy constructor
145 }//end copy constructor
146
147
148 template<class Type>
149 inline void linkedQueueType<Type>::printQueue()
150 {
151     cout << "Printing queue:" << endl;
152     nodeType<Type>* tempPtr = front;
153     for (nodeType<Type>* tempPtr = front; tempPtr != NULL; tempPtr = tempPtr-
        >link)
154     {
155         cout << tempPtr->info << endl;
```

```
156     }
157     cout << endl;
158 }
159
160 #endif
```