```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
// linkedlistcpp.h
// A linked list implementation in C++.
// Author: Lauren E. Scott
// June 24, 2014
//
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include <iostream>
#include <stdlib.h>

using namespace std;

template <class T>
class Node {

public:
    Node() {};
    ~Node() {}

    Node* getNext() { return next; }
    T getData() { return data; }
    void setNext(Node* node) { next = node; }
    void setValue(T value) { data = value; }

private:
    Node* next;
    T     data;

};

template<class T>
class LList {
public:
    LList() { head = 0; }
    Node<T>* getHead() { return head; }
    void print();
    void append(T data);
    void push_front(T data);
    void del(T data);
    Node<T>* random_list();

private:
    Node<T>* head;
};
```

```
46
47  /* * * * * * * * * * * * * * * * * * * * * * */
48  // Function: append
49  // Appends a value to the back of the list.
50  /* * * * * * * * * * * * * * * * * * * * * * */
51
52  template <class T>
53  void LList<T>::append(T data) {
54      Node<T>* newNode = new Node<T>();   // First pointer for the constructio
55      newNode->setValue(data);
56      newNode->setNext(0);
57
58      Node<T>* tmp = head; // Second pointer for iterating through list to fin
59
60      if(tmp != 0) {
61          while(tmp->getNext() != 0) {
62              tmp = tmp->getNext();
63          }
64          tmp->setNext(newNode); // Appending new node to back of list.
65      } else {
66          head = newNode; // If there is no head, make it the first node. (Spe
67      }
68  }
69
70  /* * * * * * * * * * * * * * * * * * * * * * */
71  // Function: push_front
72  // Pushes a value to the front of the list.
73  // Wrote this one all myself, with no help from online sources.
74  /* * * * * * * * * * * * * * * * * * * * * * */
75
76  template <class T>
77  void LList<T>::push_front(T data) {
78      Node<T>* newNode = new Node<T>();
79      newNode->setValue(data);
80      newNode->setNext(head);
81
82      head = newNode;
83
84  }
85
86  /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
87  // Function: del
88  // Deletes a node corresponding to a data element from the list.
89  /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
90
```

```
91  template <class T>
92  void LList<T>::del(T data) {
93      Node<T>* tmp = head;
94
95      if(tmp == 0)
96          return;
97
98      if(tmp->getNext() == 0) {   // Deletes the head element if it is the onl
99          delete tmp;
100         head = 0;
101     } else {
102         Node<T>* prev;  // Pointer for keeping track of previous node.
103         do {
104             if(tmp->getData() == data) break;
105             prev = tmp;
106             tmp = tmp->getNext();
107         } while (tmp != 0);
108
109         prev->setNext(tmp->getNext());  // Previous node's next pointer skip
110
111         delete tmp;
112     }
113 }
114
115 /* * * * * * * * * * * * * * * * * * * * * * * */
116 // Function: print
117 // Prints out the entire list.
118 /* * * * * * * * * * * * * * * * * * * * * * * */
119
120 template <class T>
121 void LList<T>::print() {
122     Node<T>* tmp = head;
123
124     if(tmp == 0) {
125         cout << "List is empty." << endl;
126         return;
127     }
128
129     if(tmp->getNext() == 0) {
130         cout << tmp->getData();
131         cout << " --> ";
132         cout << "NULL" << endl;
133     } else {
134         do {
135             cout << tmp->getData();
```

```
136             cout << " --> ";
137             tmp = tmp->getNext();
138         } while (tmp != 0);
139 
140 //      cout << "NULL" << endl;
141     }
142 }
143 
144 
145 
```