```cpp
 1 /* * * * * * * * * * * * * * * * * * * * * * * * * * */
 2 // linkedlistcpp.h
 3 // A linked list implementation in C++.
 4 // Author: Lauren E. Scott
 5 // June 24, 2014
 6 //
 7 /* * * * * * * * * * * * * * * * * * * * * * * * * * */
 8
 9 #include <iostream>
10 #include <stdlib.h>
11
12 using namespace std;
13
14 template <class T>
15 class Node {
16
17 public:
18     Node() {};
19     ~Node() {}
20
21     Node* getNext() { return next; }
22     T getData() { return data; }
23     void setNext(Node* node) { next = node; }
24     void setValue(T value) { data = value; }
25
26 private:
27     Node* next;
28     T     data;
29
30 };
31
32 template<class T>
33 class LList {
34 public:
35     LList() { head = 0; }
36     Node<T>* getHead() { return head; }
37     void print();
38     void append(T data);
39     void push_front(T data);
40     void del(T data);
41     Node<T>* random_list();
42
43 private:
44     Node<T>* head;
45 };
```

```
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
```

```cpp
 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
```