```
 1  /* Filename: mylist.cpp
 2   * --------------------
 3   * Author:  M. van Bommel & Darwin Jacob Groskleg
 4   * Date:    Thursday, March 22, 2018
 5   * Class:   CSCI 162
 6   * Lab:     #21
 7   *
 8   * Purpose: to add a method to the template for a list class that reverses the
 9   * elements in a list according to a the given specification.
10   * Shows the user the output of 7 specific operations as a demonstration.
11   *
12   * Console Output Sample:
13   * ------------------------------------------------------------------------
14   * 1. Creates an empty list and displays it.
15   * Head -> NULL
16   *
17   * 2. Reverse the empty list and displays it.
18   * Head -> NULL
19   *
20   * 3. Adds a single node to the list and displays it.
21   * Head -> 0 -> NULL
22   *
23   * 4. Reverses the single node list and displays it.
24   * Head -> 0 -> NULL
25   *
26   * 5. Adds several more nodes to the list and displays it.
27   * Head -> 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> NULL
28   *
29   * 6. Reverses the larger list and displays it.
30   * Head -> 7 -> 6 -> 5 -> 4 -> 3 -> 2 -> 1 -> 0 -> NULL
31   *
32   * 7. Reverses the reversed list again and displays it.
33   * Head -> 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> NULL
34   * ------------------------------------------------------------------------
35   */
36  #include "list.h"
37  #include <iostream>
38
39  int main() {
40      std::cout << "1. Creates an empty list and displays it.\n";
41      List<int> myList;
42      myList.display();
43
44      std::cout << "\n2. Reverse the empty list and displays it.\n";
45      myList.reverse();
46      myList.display();
47
48      std::cout << "\n3. Adds a single node to the list and displays it.\n";
49      myList.insertAtEnd(0);
50      myList.display();
51
52      std::cout << "\n4. Reverses the single node list and displays it.\n";
53      myList.reverse();
54      myList.display();
55
56      std::cout << "\n5. Adds several more nodes to the list and displays it.\n"
57      for (int i=1; i<8; i++) { myList.insertAtEnd(i); }
58      myList.display();
59
60      std::cout << "\n6. Reverses the larger list and displays it.\n";
61      myList.reverse();
62      myList.display();
63
64      std::cout << "\n7. Reverses the reversed list again and displays it.\n";
65      myList.reverse();
66      myList.display();
67
68      return 0;
69  }
```

```
 1  /* Filename: list.h
 2   * --------------------
 3   * Author:  M. van Bommel & Darwin Jacob Groskleg
 4   * Date:    Thursday, March 22, 2018
 5   * Class:   CSCI 162
 6   * Lab:     #21
 7   *
 8   * Purpose: define (declare and implement) the classes for ListNode and List.
 9   * Because this is a template, the implementation is also defined in this
10   * file.
11   *
12   * Changes: List#reverse() method was added.
13   */
14  #ifndef LIST_H_INCLUDED
15  #define LIST_H_INCLUDED
16
17  #include <iostream>
18  #include <cstdlib>
19
20  /* ListNode class, required to define the List class.
21   */
22  template <class T>
23  class ListNode {
24      public:
25          T value;
26          ListNode *next;
27
28          ListNode(T val, ListNode *n = NULL) { value = val; next = n; }
29  };
30
31
32  template <class T>
33  class List {
34      private:
35          ListNode<T> *head, *tail;
36          int size;
37      public:
38          List()                  { head = tail = NULL; size = 0; }
39          bool insertAtFront(T);
40          bool insertAtEnd(T);
41          bool deleteFromFront();
42           T   getFront() const;
43          int  getSize()          { return size; }
44          bool empty()            { return size == 0; }
45          void display() const;
46          void reverse();
47  };
48
49  template <class T>
50  bool List<T>::insertAtFront(T val) {
51      ListNode<T> *temp = new ListNode<T>(val, head);
52      if (temp == NULL) return false;
53      head = temp;
54      if (tail == NULL) tail = head;
55      size++;
56      return true;
57  }
58
59  template <class T>
60  bool List<T>::insertAtEnd(T val) {
61      ListNode<T> *temp = new ListNode<T>(val, NULL);
62      if (temp == NULL) return false;
63      if (head == NULL) head = temp;
64      else tail->next = temp;
65      tail = temp;
66      size++;
67      return true;
68  }
69
```

```
 70  template <class T>
 71  bool List<T>::deleteFromFront() {
 72      ListNode<T> *temp = head;
 73      if (temp == NULL) return false;
 74      if (tail == head) head = tail = NULL;
 75      else head = head->next;
 76      delete temp;
 77      size--;
 78      return true;
 79  }
 80
 81  template <class T>
 82  T List<T>::getFront() const {
 83      if (head == NULL)
 84      {
 85          std::cout << "Cannot take front of empty list!\n";
 86          exit(EXIT_FAILURE);
 87      }
 88      return head->value;
 89  }
 90
 91  template <class T>
 92  void List<T>::display() const {
 93      ListNode<T> *temp = head;
 94      std::cout << "Head -> ";
 95      while (temp != NULL)
 96      {
 97          std::cout << temp->value << " -> ";
 98          temp = temp->next;
 99      }
100      std::cout << "NULL\n";
101  }
102
103  /* Method Name: List#reverse()
104   * Usage: mylist.reverse();
105   * ------------------------
106   * Implementation notes: uses 3 extra pointers to reverse the list,
107   *  - previous
108   *  - current
109   *  - temp (to not be confused with next)
110   *
111   *  Does not check if there's enough memory to create ListNode pointers.
112   *  Assumes tail->next points to null.
113   */
114  template <class T>
115  void List<T>::reverse() {
116      tail = head;
117      ListNode<T> *previous;
118      ListNode<T> *current  = head;
119
120      if (size > 1) {
121          ListNode<T> *temp = head->next;
122          while (temp != NULL) {
123              previous = current;
124              current  = temp;
125              temp     = temp->next;
126
127              current->next = previous;
128          }
129      }
130
131      head = current;
132      if (size > 0) tail->next = NULL;
133  }
134
135  #endif // LIST_H_INCLUDED
```