Programming Assignment Two

It is a simple but wide-reaching problem in administrative tasks that maintaining alphabetized lists of data is a time consuming process if left to human hands. Searching through a sorted list to insert a new data element in its proper place is a repetitive task whose time-to-completion increases with the number of elements in the list. We already know that computers are great at repetitive, time-consuming tasks - they don’t make errors, they are usually much faster than us, and they don’t complain. Therefore, your second programming task will be to implement a program that will allow the user to insert data elements into and remove data elements from a list such that the list maintains a sorted order. Consider a phone book company that needs to add and remove residential listings as time goes by, without disturbing the alphabetization of their master list, as your mental model.

More specifically, you will create a class system that uses a linked-list dynamic storage system. A linked-list is superior to a static array because it can grow or shrink as needed via dynamic memory allocation - an array needs to be sized once, and cannot grow or shrink. Each node of this linked-list will store one data element and one pointer to the next node; all node content will be sorted appropriately. You will also create a main program that will test your class by making and manipulating a list object, though this program will not be turned in for assessment.

Requirements

You will create a node struct for your specific data element type. This node struct should have space to store its string data element as well as a pointer to store the location of the next node.

You will also create a list class that will manage all nodes. This class will be responsible for inserting new nodes into the appropriate sorted position, for removing nodes, for printing the entire contents of the list, and for showing the number of data elements (nodes) the list holds. To make the coding of these operations easier, the list class should always have pointers to the front and the back of the list, though these shouldn’t be accessible to the outside world.

The class should have a constructor that initializes an empty list (one with zero nodes) and a destructor that frees the memory associated with each node - remember that a each new node must be added dynamically as the list grows, meaning a call to new will be made for each new node, which in turn requires that you “clean up” using calls to delete. Since we will not be making copies of the list, you do not need to write a copy constructor or an overloaded assignment operator. You may create any utility functions you wish in the private section, though none are required - they would only serve to make your coding a bit easier, if at all.

The only libraries you may use are iostream, new, and string. All functions should be written by you; they should not be copied from external sources.

You must also write a short (3-4 paragraphs) report on the issues you faced in creating the solution: problems that emerged, insights that came to you, and evaluations of how easy or how difficult the major parts of the project were. Also in this report you should declare any and all resources you used for the project:

Conversations: with whom and about what, including instructors and classmates  
Websites: URL(s) for page(s) you researched  
Textbooks: names and sections

This should be in a pure-text file called report.txt.

Interface:

list();

Your constructor should make an “empty” list - one with no nodes.
Your destructor should pass through the entire list, deleting the memory associated with each node. If the list is already empty, it should do nothing.

```cpp
void insertInOrder(const string& s);
```
Given an empty list, this function should create one node with data `s`. Given a non-empty list, a new node should be created and linked into the list such that the list maintains sorted order.

```cpp
void remove(const string& s);
```
Given an empty list, this function should print an error message for the user. Given a non-empty list that does not contain a node whose data element is the same as `s`, it should print an error message for the user. Given a non-empty list that does contain a node whose data element is the same as `s`, it should remove that node (freeing the associated memory) and the remaining list nodes should still be linked together in order.

```cpp
int getListSize();
```
Given an empty list, this function should return zero. Given a non-empty list, it should return the number of nodes in the list. This should always reflect an accurate count, given that nodes will be added and removed indiscriminantly.

```cpp
void print();
```
Given an empty list, this function should print an error message for the user. Given a non-empty list, it should print the string contents of each node in order, one string per line.

**Class Data:**
You are free to design your class data as you like, as long as you maintain pointers to the first and last node in the list, and a count of the nodes in the list. Any other data can be created at your discretion. Note, however, that your list object will not “contain” any of the nodes; the nodes will be created in memory via calls to new, and the pointers in your list object will simply point at two (first and last) or more of the nodes. Therefore, you should not have an actual node variable in your class data; you will use pointers to nodes instead.

**Example of the Class in Use**

```cpp
// in file main.C
list L;         // Visual Model for Illustration Purposes Only
L.insertInOrder(“James”);  // James
L.insertInOrder(“Joseph”);  // James → Joseph
L.insertInOrder(“Jim”);    // James → Jim → Joseph
L.insertInOrder(“John”);   // James → Jim → John → Joseph
L.remove(“Jim”);           // James → John → Joseph
L.remove(“James”);         // John → Joseph
```

**Submission**
You will submit three files - list.h, list.C, and report.txt. Details on how to submit will be provided later.

**Deadline**
Your three files must be submitted successfully by 11:59 pm on July 10, 2003.