CISC 220 – 010 Homework #4 Due Monday, October 2, 2000

For this week's homework, we will be working mostly with the linear list ADT and a little bit with the `String` class. We'll add a couple of small features to the `String` class (an example of how an application can suggest new features), and use the `List` class to manage two lists at once.

Part A) First, a couple of small additions to the `String` class are needed – one to allow retrieval of a single character from the `String`, the other to allow the `getline` function to work on a `String` object. For example, we need operations like the following to work (use this to test your new features):

```
int main() {

   String inputLine(100);

   cin.getline(inputLine, 100);   // input a line

   cout << inputLine << endl;

   char FirstChar = inputLine[0]; // grab first character

   cout << "The first character is " << FirstChar << endl;


}
```

To make this work, you will need to implement the overloaded [] operator, and change the `String to char*` cast operator to return a non-constant pointer. After making these changes to `String.h` and `String.cc`, use the driver program above to debug your code. Then make a script file (`HW-4-A.scr`) in which you cat the header file and implementation file for the `String` class, the driver, and then compile and run the driver.

Part B) Make sure that your `List` class works, where `String` is the base class – use the following little driver to test it. Once it works, make a script file (`HW-4-B.scr`) containing a cat of the list header file, this driver, a compile and a run.

```
int main() {
   List<String> myList(10);
   String item("a test string");
   myList.insert(0, item);
   item = "the last line";
```

```
    myList.insert(1, item);
    myList.insert(0,”numero uno”);

    for(int k = 1; k <= myList.size(); k++ ) {
        myList.retrieve( k, item );
        cout << “item “ << k << “ is “ << item << endl;
    }

}
```

Part C) We are now ready to develop a small application which uses the List and String classes to manage a “to do” list. ☺ The following sample main program should give you a good idea of what we mean. All that is left for you to do is write the missing (global!) functions. Here is the little starter program:

```
#include "String.h"
#include "LinkedList.h"  // replace with your header file name
#include <fstream.h>

void ReadFile( const char*, List<String>& );
void SaveList( const char*, const List<String>& );
void ShowList( const List<String>& );
void AddItem( List<String>& );
void MarkItemDone( List<String>& , List<String>& );

int main(){

   cout << "This is your ToDo list manager at your service. Please"
        << endl;
   cout << "wait while I get your lists for you. . . .\n" << endl;

   List<String> toDoList, doneList;

   ReadFile( "ToDo.data", toDoList );
   ReadFile( "Done.data", doneList );

   String input(20);
   char command;
   do{
      cout << "Yes? (A = Add, D = Do, S = Show, Q = Quit) -> ";
      cin.getline(input,20);
      command = input[0];
      switch (command) {
         case 'A': case 'a':
            AddItem( toDoList );
            break;
         case 'D': case 'd':
            MarkItemDone( toDoList, doneList );
            break;
         case 'S': case 's':
            cout << "Your things to do:" << endl;
```

```
                ShowList( toDoList );
                cout << "\nWhat you have completed:" << endl;
                ShowList( doneList );
                break;
             case 'Q': case 'q':
                SaveList( "ToDo.data", toDoList );
                SaveList( "Done.data", doneList );
                break;
             default:
                cout << "I can't do that." << endl;
          }
       } while(command != 'Q' && command != 'q' );

       cout << " Good day! :-)" << endl;
    }
```

Following are a few comments about the required functions. Generally, make sure the functions do something reasonable when an error or pseudo-error occurs. For example, the function `ReadFile` has to open a data file and read from it; if the file does not exist, or is empty, the function should not complain, but instead just leave the list empty. Other functions should properly handle empty lists. Remember that these functions are not members of the List class, so they **must** use the List interface to do their work. Once the program runs correctly, make a script file (HW-4-C.scr) in which you cat you source files, compile and run, illustrating that program works.

Extra Credit: If you find parts A through C a little too easy, then implement a linked version of the linear list, and use it in the "to do" application. If you can think of other useful functions for the "to do" application, go ahead and add them – with comments in the source code about why you think they should be there. Make a script file similar to the one for Part C.

What to hand in: Hand in the three script files generated above (and the extra credit if you did it), stapled together as one set. (Don't forget to print your name on the front page!)