

Homework 1: Linked Lists (due Friday, July 9th at 11:59:00 PM)

Here is a C++ class definition for an abstract data type `LinkedList` of `string` objects. Implement each member function in the class below. Some of the functions we may have already done in lecture, that's fine, try to do those first without looking at your notes. You may add whatever private data members or private member functions you want to this class. You may only implement a singly linked list and not a doubly linked list for this assignment.

```
typedef std::string ItemType;

struct Node {
    ItemType value;
    Node *next;
};

class LinkedList {
private:
    Node *head;
public:
    // default constructor
    LinkedList();

    // copy constructor
    LinkedList(const LinkedList& rhs);

    // Destroys all the dynamically allocated memory
    // in the list.
    ~LinkedList();

    // assignment operator
    const LinkedList& operator=(const LinkedList& rhs);

    // Inserts val at the front of the list
    void insertToFront(const ItemType &val);

    // Prints the LinkedList
    void printList() const;

    // Sets item to the value at position i in this
    // LinkedList and return true, returns false if
    // there is no element i
    bool get(int i, ItemType& item) const;
```

```

    // Reverses the LinkedList
    void reverseList();

    // Prints the LinkedList in reverse order
    void printReverse() const;

    // Appends the values of other onto the end of this
    // LinkedList.
    void append(const LinkedList &other);

    // Exchange the contents of this LinkedList with the other
    // one.
    void swap(LinkedList &other);

    // Returns the number of items in the Linked List.
    int size() const;
};

```

When we don't want a function to change a parameter representing a value of the type stored in the `LinkedList`, we pass that parameter by constant reference. Passing it by value would have been perfectly fine for this problem, but we chose the `const` reference alternative because that will be more suitable after we make some generalizations in a later problem.

The `get` function enables a client to iterate over all elements of a `LinkedList`. In other words, this code fragment

```

LinkedList ls;
ls.insertToFront("Hawkeye");
ls.insertToFront("Thor");
ls.insertToFront("Hulk");
ls.insertToFront("Black Widow");
ls.insertToFront("Iron Man");
ls.insertToFront("Captain America");

for (int k = 0; k < ls.size(); k++)
{
    string x;
    ls.get(k, x);
    cout << x << endl;
}

```

must write

Captain America

Iron Man
 Black Widow
 Hulk
 Thor
 Hawkeye

The `printList` and `printReverse` functions enables a client to print elements of a `LinkedList`. In other words, this code fragment

```
LinkedList ls;
ls.insertToFront("The Mandalorian");
ls.insertToFront("Baby Yoda");
ls.insertToFront("Cara Dune");
ls.insertToFront("Greef Karga");

ls.printList();
ls.printReverse();
```

must write

```
Greef Karga Cara Dune Baby Yoda The Mandalorian
The Mandalorian Baby Yoda Cara Dune Greef Karga
```

You should have one space between after each item printed with an additional newline after the last item.

Here is an example of the `append` function:

```
LinkedList e1;
e1.insertToFront("Athos");
e1.insertToFront("Porthos");
e1.insertToFront("Aramis");
LinkedList e2;
e2.insertToFront("Robin");
e2.insertToFront("Batman");
e1.append(e2); // adds contents of e2 to the end of e1
string s;
assert(e1.size() == 5 && e1.get(3, s) && s == "Batman");
assert(e2.size() == 2 && e2.get(1, s) && s == "Robin");
```

Here is an example of the `reverseList` function:

```
LinkedList e1;
e1.insertToFront("Jim");
e1.insertToFront("Oz");
```

```
e1.insertToFront("Paul");
e1.insertToFront("Kevin");
e1.reverseList(); // reverses the contents of e1
string s;
assert(e1.size() == 4 && e1.get(0, s) && s == "Jim");
```

Here's an example of the `swap` function:

```
LinkedList e1;
e1.insertToFront("A");
e1.insertToFront("B");
e1.insertToFront("C");
e1.insertToFront("D");
LinkedList e2;
e2.insertToFront("X");
e2.insertToFront("Y");
e2.insertToFront("Z");
e1.swap(e2); // exchange contents of e1 and e2
string s;
assert(e1.size() == 3 && e1.get(0, s) && s == "Z");
assert(e2.size() == 4 && e2.get(2, s) && s == "B");
```

When comparing items, just use the `==` or `!=` operators provided for the `string` type by the library. These do case-sensitive comparisons, and that's fine.

Turn It In

By Thursday, July 8, there will be a link on CCLE that will enable you to turn in this homework. Turn in one zip file that contains your solutions to the homework problem. The zip file must contain only the files `LinkedList.h`, `LinkedList.cpp`, and `main.cpp`. The header file `LinkedList.h` will contain all the code from the top of this specification (`includes`, `typedef`, `struct Node`, `class LinkedList`) and proper guards, while the C++ file `LinkedList.cpp` will contain the `LinkedList` member functions you will write. If you don't finish everything you should return dummy values for your missing definitions. The main file `main.cpp` can have the main routine do whatever you want because we will rename it to something harmless, never call it, and append our own main routine to your file. Our main routine will thoroughly test your functions. You'll probably want your main routine to do the same. Your code must be such that if we insert it into a suitable test framework with a main routine and appropriate `#include` directives, it compiles. (In other words, it must have no missing semicolons, unbalanced parentheses, undeclared variables, etc.)