CS14 Summer 2016
Programming Assignment 1

Due Sunday June 26 at 11:30 PM

Synopsis - You will implement a singly-linked list of characters called *MyList*. *MyList* has a `head` pointer, but you may NOT add a `tail` pointer. A node in your list has two data members, a char type variable called *value* and a pointer of type char called `next` which points to the next node positioned in the list. You will be provided the interface for both the *Node* and *MyList* classes. They are in an archive called `assn1_classes.tgz` on the iLearn website. You may NOT make any changes to existing function prototypes, variables etc. in the files provided. Helper functions are allowed if you create them to assist you in implementing the required functionality.

1 MyList

You must implement MyList. Your source file must be called *MyList.cc*. The descriptions of the required functionality is below. A reasonable plan of attack is to implement and test the more straight-forward functions prior to working on the more challenging parts. Your goal should be to complete a portion of the program each day.

1.1 Constructors and Destructor

- ~~`MyList();`~~

  Default constructor.

- `MyList(const MyList& str);`

  Constructs a list from a passed in `MyList` object, i.e. `MyList l1(l2);`

- `MyList(const string& str);`

  Constructs a list from a passed in *string* object, i.e. `string name = "Jake"; MyList l1(name);`

- `MyList(const char* str);`

  Constructs a list from a passed in *string literal*, i.e. `MyList l1("Hello World");`

- ~~`~MyList();`~~

  Destructor.

1.2 Mutators

- ~~`void push_front(char value);`~~

  Inserts `value` at the front of the list.

- ~~`void push_back(char value);`~~

  Inserts `value` at the back of the list.

- ~~`void pop_front();`~~

  Removes the front item from the list.

- ~~`void pop_back();`~~

  Removes the last item from the list.

- `void swap(int i, int j);`

  Swaps the value of the node at position i in the list with value of the node at position j. Be sure you handle out-of-range calls.

- ~~`void insert_at_pos(int i, char value);`~~

  Inserts a node with `value` at position i in the list.

- ~~`void reverse();`~~

  Reverses the items in the list.

1.3 Accessors

- ~~`int size()const;`~~

  Returns the number of nodes in the list.

- ~~`void print()const;`~~

  Prints the contents of the list.

- ~~`int find(char value)const;`~~

  Finds the position of the first occurrence of a character `value` in the list and returns that position. If the character is not in the list, the function returns -1.

- `int find(MyList& query_str)const;`

  Finds the position of the first occurrence of the MyList `query_str` in the list and returns that position. If `query_str` is not in the list, the function returns -1.

1.4 Operator Overloaders

- `MyList& operator=(const MyList& str);`

  For assignment of one list to another, i.e. `l1 = l2;`

- `char& operator[](const int i);`

  Returns the character at position *i*.

- `MyList& operator+(const MyList& str);`

  Concatenates two lists together, i.e. `l1 + l2;`

You can read more about operator overloading here:
http://courses.cms.caltech.edu/cs11/material/cpp/donnie/cpp-ops.html

2 Testing and Error Handling

You must provide us with the `main.cc` you developed to test your program. A portion of your grade with be based on how thorough your tests are. You are also expected to perform error checking in your program. How you handle out of bounds cases, your use of `assert` or other protective measures will be inspected. **We will use our own main function when testing your program's functionality. Ensure you do not have any segfaults, as what cannot be run, cannot be tested.** Your code should be self-documenting and/or well commented and as efficient as possible. Good style and Object Oriented Principles should be obeyed.

3 Submission

You must turn-in a tar archive through the iLearn electronic submission system named assn1.tgz which has in it your main.cc, MyList.h, MyList.cc, and node.h. Use the flags czvf when creating your archive. **Do not forget to put a class header on every file you create or modify. Files lacking a header will not be graded.** Re-download and test your submission. Corrupted files or programs that do not compile with our main will receive a zero.

A reminder about collaboration on programming assignments - Please remember, at-home programming assignments are not lab assignments and you may not team-code with your lab partner or any other individual. Limited collaboration may be acceptable, but programs must represent YOUR OWN original work. Sharing code or team-coding are strictly not allowed. Copying code from ANY source (any book, current or past students, past solutions, web, etc.) is strictly prohibited. Collaboration may consist of discussing the general approach to solving the problem, but should not involve communicating in code or even pseudo-code. Students may help others find bugs.

It is a process to become a good programmer. A process by which confidence is earned and experience gained through honestly solving problems that once seemed insurmountable. Don't short-cut the process. Trust your abilities.