University of Newcastle
Discipline of Computing and Information Technology
Semester 2, 2017 - SENG1120/6120

Assignment 1

Due using the Blackboard Assignment submission facility:
11:59PM – Sunday, 03 September 2017

Update 17/Aug: Changes are indicated in yellow.

NOTE: *The important information about submission and code specifics at the end of this assignment specification***.**

INTRODUCTION

You are required to build the infrastructure to manipulate text as in a word processor. Your client further specifies that you are to create a class named `LinkedList` to store text inputted by the user. The `LinkedList` will store each word in a `Node` of the list.

ASSIGNMENT TASK

You are required to use a linked list, as discussed in lectures, to create your own implementation of the `LinkedList` class. It will use instances of `Node` to store instances of `value_type` (in this assignment, each `Node` will be used to store a single word using a `string` type).

Your `LinkedList` class will implement the following member functions:

- Constructors, which return an empty `LinkedList`, or use an argument-provided string to initialise the `LinkedList` on creation.
- `void add(string)` which takes a string as a parameter and appends it to the `LinkedList`, one word in each node.
- `int length()` which returns a count of the words in the `LinkedList`.
- `int count(value_type)` which takes a single word (represented as a `value_type`) as argument and returns a count of how many times the word appears in `LinkedList`.
- *Overloaded* concatenation operator (i.e. '+='). The result of use of this operator is the `LinkedList` arguments are concatenated and the answer is stored in the left argument.
- *Overloaded* output operator (i.e. '<<') that outputs the content of the `LinkedList` in a form suitable for printing.
- *Overloaded* input operator (i.e. '>>') that simplifies input of the content of a `LinkedList`.
- `remove(value_type)` which takes a single word (represented as a `value_type`) argument and removes any occurrence of that word in the `LinkedList`.

SENG6120 students should implement, in addition to the previous member functions, the following one:

- `void reverse()` which reverses the order of the nodes in `LinkedList` (e.g `Mary had a little lamb -> lamb little a had Mary`).

DEMO PROGRAM

To demonstrate your `LinkedList` class you will write a program called `LinkedListDemo` that uses your `LinkedList` class in the following way:

1. When executed, `LinkedListDemo` will ask the user to input a sentence, which will then be stored within an instance of your `LinkedList` class named `inputOne`.
    - You need to use the overloaded >> and << operators, and the function `add(string)`.
2. `LinkedListDemo` will then display the contents and the number of words in `inputOne` as well as reporting how many times the word "SENG1120" occurs.
    - You need to use the functions `length()` and `count(value_type)`.
3. `LinkedListDemo` will then request a second string, which will be stored within an instance of `LinkedList` class named `inputTwo`. Concatenate `inputOne` and `inputTwo`, storing the result in `inputOne`, and print both variables.
    - You need to use the overloaded >> and += operators and the function `add(string)`.
4. `LinkedList` will then display the length of `inputOne` as well as reporting how many times the word "is" occurs.
    - You need to use the functions `length()` and `count(value_type)`.
5. `LinkedListDemo` will input a word from the user and remove all its occurrences from `inputTwo`. Print out the contents of `inputOne` and `inputTwo`.
    - You need to use the overloaded << operator, and `remove(value_type)`.

SENG6120 students should implement, in addition to the previous demo steps, the following one:

6. `FlexStringDemo` will reverse the contents of `inputOne` and display them.
    - You need to use the function `reverse()` and the overloaded << operator.

Your submission should be made using the Assignments section of the course Blackboard site. **Incorrectly submitted assignments will not be marked. Assignments that do not use the specified class names will not be further marked.** You should provide all your `.h` and `.cpp` files, and a `Makefile`. Also, if necessary, provide a `readme.txt` file containing any instructions for the marker. Each program file should have a proper header section including your name, course and student number, and your code should be properly documented.

Remember that your code should compile and run correctly using Cygwin. There should be no segmentation faults or memory leaks during or after the execution of the program.

Compress all your files, including the cover sheet, into a single .zip file and submit it in by clicking in a link that I will create in the Assignments section on Blackboard especially for that.

Late submissions are subject to the rules specified in the Course Outline. Finally, a completed Assignment Cover Sheet should accompany your submission.

This assignment is worth 10 marks of your final result for the course.