Assessed Practical 3

Wednesday 12th December 2018

In this assessed practical, you have three parts to complete – (1) Stacks, (2) Lists, and (3) Trees. Even if you can't complete all of it, just do as much as you can.

In Microsoft Visual Studio, create a new empty Project called "Assessed_Practical3" on the C:\ drive or on your own drive. Add a new source file "assessed_main.cpp" to the project. There are three sets of files in the QOL folder Resources→Practical Booklets->CSC2040 Assessed Practical 3:

StackNode.h, LinkedStack.h (for building Linked Stack based applications)
ListNode.h, LinkedList.h (for building Linked List based applications)
TreeNode.h (for building Tree based application)

Save a copy of these files in your project folder (e.g., …\ Assessed_Practical3\ Assessed_Practical3), and #include them in your main program file "assessed_main.cpp".

Part 1: Extending the class LinkedStack (6 marks)

Add a new public function to the LinkedStack class:

```
int countStack (T item)
```

This function should return the number of occurrences of 'item' in the stack. It should not change the stack in any way.

Create a main function in the "assessed_main.cpp", and define an integer-type LinkedStack object *s*. Set *s* up by pushing 100 random integer numbers into *s*, each number with a value 0-9. Use this stack to test if your revised stack class works properly, by counting the occurrences of 0-9 and 10-19, respectively, in the stack. The returned counts for 10-19 should all be zero, and the returned counts for 0-9 should total to 100. [Hint: To generate the random numbers, first call srand((unsigned)time(NULL)) to initialise the generator, for which you need to #include <ctime>, and then call rand() % 10 for 100 times to obtain the random numbers of the required range.]

Part 2: Building linked lists of characters (8 marks)

1) Reading and building a list of characters

Rename the main function for Part 1 to main1 and create a new main function. For this section, a sentence consists of a list of characters terminated by a full stop. In the main function, write the following code to create a linked list of characters, *sentence*. The code allows the user to type a sentence e.g. *This is a test.* The input is ended with a full top, followed by hitting the Enter key. The sentence characters will be read in by cin.get() and added to the list. The code is incomplete, with a data type missing at the part marked by (1), and a statement missing at the part marked by (2). Complete the code so that it will do the expected job.

```
// a linked list of characters
LinkedList< (1) > sentence;

// reading and set up the list terminated by a full stop
char ch;
while (ch = cin.get()) {
      /* (2) */
      if (ch == '.') break;
}
```

2) Adding a printList function to the class LinkedList

To test your code above, you need to be able to print a list. To do this, extend your copy of the file LinkedList.h by adding an extra public member function:

```
void printList()  // Prints all items 1 per line, plus a blank line at end
```

Now, in the main function, use this to print the list of characters *sentence* you got from above. You should see them printed out one character per line, including spaces and the final full stop.

3) Adding a searchList function to the class LinkedList

Add another new public member function to the LinkedList class:

```
int searchList(T item)  // Return the position of the first occurrence of 'item'
                        // in the list. The position of the 1st item in the list
                        // is 0. If 'item' doesn't exit, you should return -1.
```

In the same main function, test the above search list function by searching in the list sentence for a letter that is in the list, and for a letter that is not in the list.

Part 3: Trees for representing and evaluating arithmetic expressions (6 marks)

Arithmetic expressions can be represented by a binary tree (of strings). For example, we will define two standard variables X and Y which can be suitably initialised; then the arithmetic expression X + (Y – 23) * 6 could be represented by the tree shown in Figure 1.

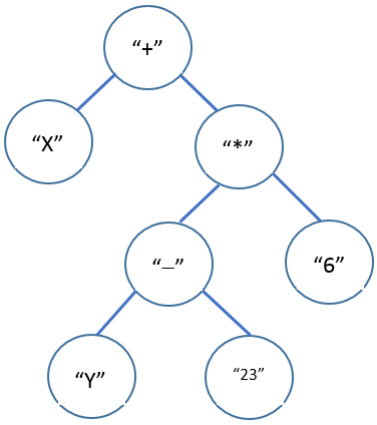

Figure 1

You are asked to build such a tree, and to evaluate it given initial values of X and Y. First, rename the main function for Part 2 to main2.

1) Build a binary tree

Create a new main function to build the binary tree for the arithmetic expression shown in Figure 1 using dynamic allocation. Each item will be a string: an operator ("+", "-" or "*"); or a variable ("X" or "Y"); or a number (e.g. "23").

2) Write a recursive function

```
int evaluate (TreeNode<string> *tree, int X, int Y)
```

which takes an expression tree such as that shown in Figure 1, plus the values of the two variables X and Y, and calculates and returns the actual result. For example, for the small tree shown in Figure 2,

```
cout << evaluate(smallTree, 5, -3) << endl;
```

should print out 2.

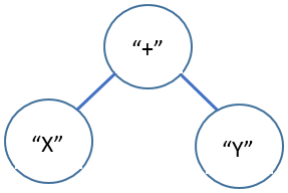

Figure 2

Your function should see which type of node the root is by comparing the (string) item with the various options for what it could be (e.g., one of the three operators, one of the two variables, or a number). If the root node item is an operator, your function should apply the specified operation to the left and right node items by using recursive calls, and return the result. If the root node item is a variable (e.g., "X"), your function should return the value of the variable (i.e., return X). If the root node item is a number (e.g., "23"), your function should return the number (i.e., return 23).

Hint: to convert a string to an integer, use the function stoi (for string-to-int): intValue = stoi (string_item);

In the main function call the evaluate function with the first tree set up above, with X and Y being 5 and 3 respectively, and print out the resulting value. Check that it is right!

Submission Notes

1. Save all your files:
   LinkedStack.h
   LinkedList.h
   assessed_main.cpp.
2. Close Visual Studio and locate the folder Assessed_Practical3 on the C:\ drive or on your own drive. ZIP this folder (by right clicking the folder and choosing 'Send To' and selecting Compressed (Zipped) folder).
3. Check that your .zip file contains your source files (double click on the .zip file. You do not need to extract the files again).
4. In case something goes wrong with your submission, first make a backup of your zip file on to a USB drive, and/or email it to yourself.
5. Upload the .zip file to Queens Online using the Assignment tool into CSC2040 Assessed Practical 3.