Assignment 9

Due Nov 23 by 7pm **Points** 120

Project 9.a

Using an appropriate definition of ListNode, design a simple linked list class called StringList with the following member functions:

- void add (std::string);
- int positionOf (std::string);
- bool setNodeVal(int, std::string);
- std::vector<std::string> getAsVector();
- a default constructor
- a copy constructor
- a destructor

The add() function adds a new node containing the value of the parameter to the end of the list. The positionOf() function returns the (zero-based) position in the list for the first occurrence of the parameter in the list, or -1 if that value is not in the list. In the setNodeVal() function, the first parameter represents a (zero-based) position in the list. The setNodeVal() function sets the value of the node at that position to the value of the string parameter. If the position parameter is >= the number of nodes in the list, the operation cannot be carried out and setNodeVal() should return false, otherwise it should be successful and return true. The getAsVector() function returns a vector with the same size, values and order as the StringList. The default constructor should initialize a new empty StringList object. The copy constructor should create a completely separate duplicate of a StringList object (a **deep copy**). The destructor should delete any memory that was dynamically allocated by the StringList object.

Files must be called: **StringList.hpp** and **StringList.cpp**

Project 9.b

We normally write arithmetical expressions using *infix* notation, meaning that the operator appears between its two operands, as in "4 + 5". In *postfix* notation, the operator appears after its operands, as in "4.3 5.0 +". Here is a slightly more complex postfix expression: "25 12 7 - 2 * /". The equivalent infix expression is: "25 / ((12 - 7) * 2)". The result of that expression should be 2.5 (**beware integer division**). Postfix expressions don't require parentheses.

Write a function named *postfixEval* that uses a stack<double> (from the Standard Template Library) to evaluate postfix expressions. It should take a C-style string parameter that represents a postfix expression. The only symbols in the string will be +, -, *, /, digits and spaces. '+' and '-' will only appear in the expression string as binary operators - not as unary operators that indicate the sign of a number. The return type should be double. You may find the isdigit() function useful in parsing the expression. You may also use strtok() and atof().

Hint: Read a postfix expression from left to right. When you read a number, push it on the stack. When you read an operand, pop the top two numbers off the stack, apply the operator to them, and push the result on top of the stack. At the end, the result of the expression should be the only number on the stack.

File must be called: **postfixEval.cpp**