CSCI 689 **Computer Assignment 9** **Spring 2016**
(30 points)

A Simple Calculator

Write a C++ program that implements a simple arithmetic calculator. Input to the calculator consists of lines composed of real numbers combined together using the arithmetic operators +, −, *, and /. The arithmetic expressions in input lines are given in *postfix* forms, which eliminates the need of using parentheses.

You can use a stack container (of type double) to evaluate a *postfix* expression. The main ( ) routine defines an empty stack and gets the input from the stdin token by token until EOF, where a token is defined a C++ string that doesn't contain any white spaces. For each token in the input stream, it calls the following routine to process the token.

- void process_token (const string& token, stack < double >& S): This routine inspects each character in the token in sequence and takes the proper action according to the inspected character in the token. If the character is a digit, a unary −/+ sign that can be determined by the *predicate* unarySign ( ), or a valid decimal point that can be determined by the *predicate* floatPoint ( ), it calls the routine getNumber ( ) to get all characters of the corresponding number from the token (whose description and the descriptions of the two helper functions in this process are given below) and puts the number in stack S.

- bool unarySign (const char& c, const string& token, const unsigned& i): This function (a *predicate*) checks if the character c in position i in the token is a valid unary −/+ sign, which is considered valid if it's not the last character in the token and the character follows it is either a digit or a decimal point.

- bool floatPoint (const char& c, const string& token, const unsigned& i): This function (a *predicate*) checks if the character c in position i in the token is a valid decimal point, which is considered valid if it's not the last character in the token and the character follows it is a digit.

- double getNumber (const char& c, const string& token, unsigned& i, const bool& floatPointFlag): This function starts with the character c in position i in the token and constructs a double–precision floating number from a set of consecutive characters in the token. It also counts the total number of decimal points in the corresponding number, and in case the number contains more than one decimal point, it indicates this error by printing out an error message on stderr. Otherwise, it calls the routine stringToDouble ( ) to convert the processed substring to a number and returns it to the routine process_token ( ). The last argument floatPointFlag indicates that if c is a decimal point.

If a character inspected in the routine process_token ( ) is a valid binary operator that can be determined by the *predicate* isValidOperator ( ), then the process_token ( ) pops two top numbers from the stack S, computes the corresponding operation on these numbers by

calling the helper function operation ( ), and puts the resulting number in the stack S. To pop a number from the stack S, the routine popStack ( ) can be used.

- bool isValidOperator (const char& c): This function (a *predicate*) checks if the character c is one of the four valid operators in { +, -, *, / }.

- double operation (const char& c, const double& x, const double& y): This function applies the operator c on the numbers x and y and returns the resulting number to the calling routine.

- double popStack (stack < double >& S): This routine first checks if the stack S is empty, and if it's, then it prints out an error message on stderr to indicate the error. Otherwise, it removes and returns the number at the top of S to the calling routine.

If a character inspected in process_token ( ) is an equal sign (=), it calls the routine printResult ( ) to print out the number at the top of the stack S, and if the character is 'c', it calls the routine emptyStack ( ) to empty the stack S. If the inspected character is not part of a number and if it's not one of the valid operators, process_token ( ) prints out an error message on stderr to indicate the error.

- void printResult (const stack < double >& S): This routine first checks if the stack S is empty, and if it's, then it prints out an error message on stderr to indicate the error. Otherwise, it prints out the top number in S on stdout by exactly two digits after the decimal point.

- void emptyStack (stack < double >& S): This routine pops the top element from the stack S and continue by popping the top element until S becomes empty.

This is an interactive calculator, so you can enter numbers (positive or negative) and operators from the keyboard, and in case of an error, your program needs to print out an error message on the screen but it doesn't stop its execution. Execution is terminated only when you enter <ctrl>–D from the keyboard.

Put the declarations of all routines that you use in your program in your header file prog9.h, and the implementation of all routines in your source file prog9.cc. At the top of your source file, insert the following statement: #include "prog9.h".

To compile your source file and link the generated object file with system library routines, first make a link to makefile in directory: ~cs689/progs/16s/p9 from your working directory, and then execute: make N=9.

For a final test of your program, execute: make execute N=9. This will test your program with data files prog9.d1 and prog9.d2, and generate the output files prog9.out1 and prog9.out2. Data files and the correct output files are all in the same directory with makefile.

When your program is ready, mail its source and header files to your TA by executing the following command: mail_prog prog9.cc prog9.h.