THE UNIVERSITY OF WESTERN ONTARIO

DEPARTMENT OF COMPUTER SCIENCE
LONDON CANADA

Software Tools and Systems Programming
(Computer Science 2211a)

ASSIGNMENT 3
Due date: Tuesday, October 30, 2018, 11:55 PM

Assignment overview

We would like students to understand and use C types such as char, int, and float, and to experience with the input and output, the flow control structures, the recursive functions, and the arrays.

This assignment consists of two parts.

In part one, you are required to write a C programs to evaluate simple arithmetic expression.

In part two, you are to wrote a C program to display integers using seven-segment display.

Part one: 70%

The goal of the exercise is to implement a simple calculator, called "evaluate", to evaluate simple arithmetic expression.

(1) An arithmetic expression is an expression that results in a numeric value. We consider numeric value to be real or floating point numbers. A simple arithmetic expression involves numeric values connected by arithmetic operators. In this exercise, numeric value will be real or floating point numbers and operators will be $+$, $-$, $*$, and $/$.

For example, 4 + 35 is a simple arithmetic expression and $2.3 * 4 - 5 -$ -7.8 + 9 / 3 is another simple arithmetic expression.

In evaluating arithmetic expression, $*$ and $/$ have higher precedence than $+$ and $-$. With operators of the same precedence, the order of evaluation is from left to right. We will use **s_exp** to represent simple arithmetic expression, **m_exp** to represent simple arithmetic expression in which all operators are $*$ or $/$, **l_op** to represent operators $+$ and $-$, **h_op** to represent operators $*$ and $/$, and **num** to represent numeric value.

In the following, simple arithmetic expression is represented recursively, where | represent OR relationship. This recursive definition would be useful in designing your

solution for this exercise.

s_exp	→	**m_exp**
s_exp	→	**s_exp l_op m_exp**
m_exp	→	**num**
m_exp	→	**m_exp h_op num**
l_op	→	$+ \mid -$
h_op	→	$* \mid /$

(2) In your program, first the user is asked to input a simple arithmetic expression. In the inputted simple arithmetic expression, there could be space characters before a number or an operator, The input numbers could be either integers or floating numbers and we assume that the user will always enter valid numbers. Your program should handle non-valid input character for operators.

(3) After user input, the program will calculate and print the numeric value of the inputted simple arithmetic expression. The program does not read the whole expression before its calculation. The calculation proceeds while reading numbers and operators of the inputted simple arithmetic expression.

In evaluating simple arithmetic expression, $+$ and $-$ have the same precedence and the evaluation order is from left to right, and $*$ and $/$ have the same precedence and the evaluation order is from left to right.

We will use two recursive functions to perform the evaluation. The implementation of these recursive functions should follow the recursive definition for simple expression in **(1)**. To guide you toward this goal, we provide a template function. We ask you to use this template and fill in the missing code.

```
// Input: 'sub_exp': the value of the sub s_expression to the left of 'op'
//        location in stdin.
//        'op' : an operator, '+' or '-'. 'op' could also be
//               '\n' indicating the end of the s_expression
//        the rest of the expression will be read in from stdin
// Effect: the whole s_expression is evaluated using recursion:
//         get next_num with m_exp() and then get next_op with get_op()
//         use 'sub_exp op next_num' and 'next_op' to do recursive call
// Output: this function returns the value of the s_expression

float s_exp(float sub_exp, char op) {

}
```

```
// Input: 'sub_exp': the value of the current sub m_expression
//                   to the left of 'op' location in stdin.
//        'op' : an operator, '*' or '/'.  'op' could also be
//               '+', '-', or '\n' indicating the end of the m_expression.
//               "+', '-', or '\n' should be pushed back to stdin.
//        the rest of the m_expression will be read in from stdin
// Effect: the m_expression is evaluated using recursion:
//         get next_num with get_num() and then get next_op with get_op()
//         use 'sub_exp op next_num' and 'next_op' to do recursive call
// Output: this function returns the value of the current m_expression

float m_exp(float sub_exp, char op) {

}
```

The following two functions should also be used to simplify the programming task.

```
// Input: none, read from stdin
// Effect: get the next operator of the expression
//         this operator can be +, -, *, /, or '\n'
//         '\n' indicates the end of the expression input
//         leading spaces should skipped
// Output: return the next operator of the expression.

char get_op() {

}
```

```
// Input: none, read from stdin
// Effect: get the next numeric value of the expression
// Output: return the next numeric value of the expression.

float get_num() {

}
```

To push back one character in *ch* to stdin, use $ungetc(ch, stdin)$. $ungetc()$ is a library function that can push back a character to a specified stream.

When an error is detected, you can exit your program by a function call $exit(EXIT\_FAILURE)$. To use this function, the required header is:
$\#include\ <stdlib.h>$

(4) Then the user is asked if she/he wants to continue. Two characters can be entered, each corresponding to a possible action.

- Y for continuing inputting a simple arithmetic expression
- N for quit

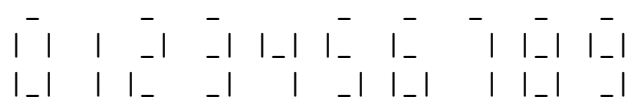
(5) Evaluate and print values, or report input format errors, for the following simple arithmetic expressions with your program.

- 5
- $3 * 4$
- $4 + 3$
- $2.6 / 2 - 1.5 * 10$
- $-2.0 * 2 + 1.5 * 10 - 100$
- $3.5 - -1.5 * 10 / 3.0 + 2 * 3$
- $3.5 - 1.5 \% 10 - 2.0$

Part two: 30%

The goal of this exercise is to implement a C program to display integer in seven-segment display format.

(1) Calculators, watches, and other electronic devices often rely on seven-segment displays for numerical output. To form a digit, such devices "turn on" some of the seven segments while leaving others "off".



(2) In your program, the user is asked to input an integer. After reading the input, the program will output the inputted integer using seven-segment displays.

You should use a three dimensional array of characters to store the 10 digits.

Here is what the array may look like:

```
const char segements[10][3][3] =
   { { {' ', '_', ' '}, {'|', ' ', '|'}, {'|', '_', '|'} }, ... };
```

Only the initialization for segments[0] is given and you should fill in the rest.

Do not forget the sign of the integer.

(3) Then the user is asked if she/he wants to continue. Two characters can be entered, each corresponding to a possible action.

- Y for continuing inputting an integer
- N for quit

(4) Testing your program by inputing several representative integers.

Testing your program

You should test your program by running it on Gaul. Capture the screen of your testing by using script command. There should be two script files, one for each part.

Your program should follow good programming styles, i.e. write clear code, choose good variable names, use appropriate functions, make proper comments, and etc.