THE UNIVERSITY OF WESTERN ONTARIO

DEPARTMENT OF COMPUTER SCIENCE
LONDON CANADA

Software Tools and Systems Programming
(Computer Science 2211a)

ASSIGNMENT 3
Due date: Friday, October 20, 2023, 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 *scanf*() and *stdin*, 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 write a C program to display integers using seven-segment display.

Part one: 60%

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

(1) An arithmetic expression is an expression involving arithmetic operators, i.e. $+$, $-$, $*$, and /, numeric values, and parentheses (and) that results in a numeric value. We consider numeric value to be real or floating point numbers. In this assignment, we only consider simple arithmetic expressions which involve numeric values connected by arithmetic operators $+$ 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 simple arithmetic expression with more than one operator, the order of evaluation is from left to right. We will use **sim_exp** to represent simple arithmetic expression, **op** to represent operators $+$ 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.

sim_exp	→	**num**
sim_exp	→	**sim_exp op num**
op	→	$+ \mid -$
num	→	$int \mid float$

(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 or after 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 single character input 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 with more than one operator, the evaluation order is from left to right according to the recursive definition.

It is straightforward to evaluate a simple arithmetic expression with *C* loops. However, in this assignment we ask you to perform the evaluation using a recursive function. The implementation of the recursive function should follow the recursive definition for simple arithmetic expression in **(1)**.

To guide you toward this goal, we provide template functions. We ask you to use these templates and fill in the missing code.

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 before the operator should be skipped
// Output: return the next operator of the expression.

char get_op(void) {
}
```

```
// 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(void) {
}
```

```
// Input: 'sub_exp': the value of the current sub simple arithmetic
//                   expression to the left of the next operator,
//                   located at the beginning of current stdin.
//                   The rest of the simple arithmetic expression
//                   will be read in from stdin
// Effect: the simple arithmetic expression is evaluated using recursion:
//         get 'next_op' with get_op()
//         if 'next_op' is '\n'
//           return 'sub_exp'
//         else if ('next_op' is '+' or '-')
//           get 'next_num' with get_num()
//           use 'sub_exp next_op next_num' to do recursive call
//           i.e. call sub_sim_exp()
//         end
//           error
// Output: this function returns the value of the simple arithmetic expression

float sub_sim_exp(float sub_exp) {

}

// Input: none, the simple arithmetic expression will be read in from stdin
// Effect: the simple arithmetic expression is evaluated using afunction
//         call the recursive function
// Output: this function returns the value of the simple arithmetic expression

float sim_exp(void) {
  float num = get_num();
  return sub_sim_exp(num);
}
```

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 >

Hint for the implementation:

- First implement *get_num*() and *get_op*() functions.
- Then test these two functions by reading and then printing an input simple arithmetic expression
- After you can correctly read and print an input simple arithmetic expression, implement *sub_sim_exp*(*float sub_exp*)
- If you do not know how to implement recursive function *sub_sim_exp*(*float sub_exp*), you may consider to directly implement *sim_exp*() with loops, with some mark deduction.

(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.

- 6
- $2 + 4$
- $5 - -2$
- $2.6 + 2 - 1.6 - 10$
- $-2.0 + +20 + 1.6 - 10 - 100$
- $2.6 - -1.6 + 10 + 2.0 + 2 - 2$
- $2.6 - 1.6 * 10 - 2.0$

Part two: 40%

The goal of this exercise is to implement a C program, called "ss_integer.c", 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 as an *int*, 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] and segments[1] are 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 compute.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.