CSI 333 – Programming at the Hardware-Software Interface  
SQUPT, Spring 2019

**Project I**

The total grade for the assignment is 100 points.

You must follow the programming and documentation guidelines (see file *Programming Assignments Requirements and Recommendations.docx*).

**Due date: 11:59pm Sunday, March 24, 2019**

# Description

You are required to write a C program that performs two tasks.

## Task #1. Strict Left-to-Right Evaluation of an Arithmetic Expression

In a *strict left-to-right* evaluation, there is no notion of precedence. For example, the value of the expression 6 + 4 ∗ 3 when evaluated in a strict left-to-right fashion is 30. Under usual precedence rules, where multiplication has higher precedence than addition, the value of the above expression would be 18. Similarly, the value of the expression 6 + 4 ∗ 3/7 − 9 when evaluated in a strict left-to-right fashion is −5.

This part of your C program must carry out a strict left-to-right evaluation of an arithmetic expression consisting of digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 and the operators +, −, ∗, and /, where, the operator / denotes integer division (that is, the remainder is discarded).

Each time your program is executed, it should handle just one expression. You may assume that the input expression satisfies all of the following conditions:

* The expression consists *only* of integer numbers, the operators +, −, ∗, / and spaces. (In particular, the expression won’t contain parentheses.)
* Each integer number in the expression consists of only *one* decimal digit.
* There may be zero or more *spaces* between successive non-blank characters of an expression.
* In the expression, integer numbers and operators *alternate*.
* The expression begins with an integer numbers, *without* any preceding sign.
* Each input expression is terminated by the *newline* (’\n’) character.

Thus, no error checks are needed. Bear in mind that an expression consisting of a single digit integer number, without any operators, is a valid expression.

|  |  |
| --- | --- |
| **Expression** | **Value** |
| 6 | 6 |
| 9 + 5 \* 0 - 7 | -7 |
| 0+ 7 + 4 \* 5 - 9 | 46 |
| 9+9-7+4-2/2+4-6 | 4 |
| 7\*8-9\*4-5\*6-6/3+4 | 368 |

Some expressions and their values when evaluated in a strict left-to-right fashion are given in the table on the right. However, you should remember that when we compile and run your source files, we will use other data. Just because your programs work for the sample inputs, you shouldn't assume that they will work for all inputs. Therefore, you should test your programs thoroughly with other input values.

The outline of this part of your C program is as follows:

1. Prompt the user for an expression.
2. Read the expression character by character and carry out a strict left-to-right evaluation of the expression.
3. Print the value of the result obtained on the previous step.
4. Pass control to Task #2.

## Task #2. Number Conversion

This part of you C program must show the result of the first part using arbitrary number system with the radix specified by the user. Possible radix will be one of 2, 3, 4, *. . .*, 15, 16. Negative value of the integer in all number systems must be denoted as the minus sign (-).

Examples:

1. Suppose the input is the decimal integer -138 and the radix is 16. In this case, the output produced by your program should be -8A, which is the hexadecimal representation of the decimal integer -138.
2. Suppose the input is the decimal integer 284 and the radix is 13. In this case, the output produced by your program should be 18B, which is the radix 13 representation of the decimal integer 284. (In base 13, the digits used are 0, 1, 2, *. . .*, 9, A, B and C, where A, B and C represent 10, 11 and 12 respectively.)

| **Input Number** | **Radix** | **Output** |
| --- | --- | --- |
| 138 | 16 | 8A |
| -2279 | 12 | -139B |
| 37373 | 10 | 37373 |
| 741 | 2 | 1011100101 |
| 0 | 11 | 0 |
| 284 | 13 | 18B |

The table on the right gives several examples of inputs and the corresponding outputs to test your programs. However, you should remember that when we compile and run your source files, we will use other data. Just because your programs work for the sample inputs given below, you shouldn't assume that they will work for all inputs. Therefore, you should test your programs thoroughly with other input values.

The outline of this part of your program is as follows:

* 1. Take the result of the first part as the first input.
  2. Prompt the user to specify the radix and read it.
  3. Convert the first input into its representation in the radix specified by the user.
  4. Print the representation.
  5. Stop the program.

## Programming Suggestions

* Program and test two tasks separately and combine them in a one program as two functions.
* For Task #2:
  + recall that for any radix *r* ≥2, the digits to be used are 0, 1, *. . .*, *r −* 1. Use the letters A, B, C, D, E and F to represent 10, 11, 12, 13, 14 and 15 respectively, as done in the hexadecimal system. Thus, representations in radix 11 can use the digits 0, 1, *. . .*, 9, A; representations in radix 12 can use 0, 1, *. . .*, 9, A, B, and so on;
  + use the division method (discussed in Lecture 1) to generate the digits of the required representation;
  + use a char array to store each digit generated by the division method as an appropriate character. This array should be printed out at the end.

## Notes

* Your program reads inputs from stdin (keyboard) and writes outputs to stdout (terminal window).
* No error checks are needed.
* After each call to the function printf, include the following C statement: fflush(stdout);.

Examples:

printf("Value = %d\n", result); fflush(stdout);  
printf("Enter radix: "); fflush(stdout);

# Example of program execution

The following examples assume that the executable version of the program is in the file p1.exe (if Windows).

> p1.exe

Enter expression: 9\*2 - 5/3 -9

Value = -5

Enter radix: 2

Answer = -101

>

# Submission

You must perform submissions as directed by your co-instructor.

*Ignoring any of the following rules will result in penalty or even ZERO grade for the project!*

Submission should include:

* A file named as directed by your co-instructor with source code for the project. More details will be given in your lab classes.
* Screenshot with program output.

At the top of your source code file the following information must appear in the form of comments:

* 1. Course code and title (i.e. “CSI 333. Programming at the Hardware-Software Interface”),
  2. Semester (e.g., Spring 2019),
  3. The name of your lab classes supervisor,
  4. Your class (e.g., ZR170102),
  5. Your student ID,
  6. Your pinyin name.

Make sure that your programs compile and produce correct results on the lab machines. Programs that cause compiler or linker errors on these machines will NOT receive any credit.

# Project Grading

Program will be graded by co- instructors. The total grade for this assignment is 100 points, with 40 points for correctness of Task #1 and 40 points for correctness of Task #2 and 20 points for structure/documentation.