School of Computer Science, McGill University

COMP-206 Introduction to Software Systems, Winter 2020

Mini Assignment 4: C Programming - Character Arrays and Control Structures

Due Date Nov 6th, 23:55

This is an individual assignment. You need to solve these questions on your own. If you have questions, post them on Piazza, but do not post major parts of the assignment code. Though small parts of code are acceptable, we do not want you sharing your solutions (or large parts of them) on Piazza. If your question cannot be answered without sharing significant amounts of code, please make a private question on Piazza or utilize TA/Instructors office hours. Please ensure to check "Mini 4 general clarifications" pinned post in Piazza before you post a new question. It might have been already clarified there if it is a popular question.

Late penalty is -5% per day. Even if you are late only by a few minutes it will be rounded up to a day. Maximum of 2 late days are allowed.

You MUST use `mimi.cs.mcgill.ca` to create the solution to this assignment. You must not use your Mac command-line, Windows command-line, nor a Linux distro installed locally on your laptop. You can access `mimi.cs.mcgill.ca` from your personal computer using **ssh** or **putty** as seen in class and in Lab A.

DO NOT edit files on your laptop; use `vim` on `mimi`. If TAs are not able to compile your code on `mimi` as-is, you will get 0 points. This assignment has restrictions on the functions you are allowed to use.

Labs E and F provide some background help for this assignment.

Questions in this exercise require you to turn in one C program. Instructors/TAs upon their discretion may ask you to demonstrate/explain your solution. **No points are awarded for commands that do not execute at all or programs that do not compile on `mimi`.** (Commands/programs that execute/compile, but provide incorrect behavior/output will be given partial marks.). All questions are graded proportionally. This means that if 40% of the question is correct, you will receive 40% of the grade.

Please read through the entire assignment before you start working on it. You can lose up to 3 points for not following the instructions.

Unless otherwise stated, all the names of the scripts and programs that you write, commands, options, input arguments, etc. are case-sensitive.

Total Points: 20

Ex. 1 — A Scientific Calculator (20 Points)

As discussed in class, built-in data types of programming languages (including C) have limitations in terms of size and range of data (example integers) that they can work with. This is often an impediment for many calculations (such as scientific computations) which can involve numbers beyond the range that is supported by built-in data types.

In this assignment you will develop a scientific calculator that can perform calculations over arbitrarily large numbers.

Understanding the ASCII table, and the fact that characters are internally represented by their ASCII values which makes basic arithmetic operators over them possible, can help significantly in implementing a basic solution for this assignment.

Your calculator should accept three arguments as input, `x`, `op`, `y`, where `x` and `y` are integers and `op` is an arithmetic operator.

1. Use `vim` to create a C program source file `scalc.c` . All of your source code logic should be contained in this file. Further, all of your logic should be in a single function, `main`.
2. **(2 Points)** The source code file should include a comment section at the beginning that describes its purpose (couple of lines), the author (your name), your department, a small "history" section indicating what changes you did on what date.

```
/*
 Program to implement a scientific calculator
 ************************************************************
 * Author          Dept.              Date          Notes
 ************************************************************
 * Joseph D        Comp. Science. Oct 10 2020    Initial version.
 * Joseph D        Comp. Science. Oct 11 2020    Added error handling.
*/
```

   The code should be properly indented for readability as well as contain any additional comments required to understand the program logic.
3. **(1 Point)** The source code should be compilable by (exactly) using the command `gcc -o scalc scalc.c`
4. **(1 Point)** Compilation should not produce any warnings!
5. **(1 Points)** Your program should accept the three arguments mentioned above as its input. If the user does not pass sufficient arguments, you should display an error message and terminate with error code `1`.

```
$ ./scalc 99 99
Error: invalid number of arguments!
scalc <operand1> <operator> <operand2>
$ echo $?
1
```

6. **(2 Points)** Operands are supposed to be positive integers (no decimals), including zero. For the purpose of this assignment, you need to support only the + operator. Your program should check if the arguments passed to it are valid and throw an error message.

```
$ ./scalc 99 99
Error: invalid number of arguments!
scalc <operand1> <operator> <operand2>
$ echo $?
1
$ ./scalc 99 a 99
Error: operator can only be + !
$ echo $?
1
$ ./scalc 99.12 + 99
Error!! operand can only be positive integers
$ echo $?
1
$
```

7. **(4 Points)** Your program should produce correct results for valid inputs. It should only print the final output, followed by a newline character. There should not be any spaces and extra characters printed other than the result. The result itself should not have any leading zeroes. The program should then terminate with error code 0. You do not have to handle the case of the input operands themselves having leading zeroes (e.g. 091.)

```
$ ./scalc 10 + 90
100
$ echo $?
0
$
```

8. **(9 Points)** Your program should produce correct results for very large numbers. To simplify, we will assume that the final result will have a maximum of 1000 digits and the individual operands themselves will not exceed 999 digits.

```
$ ./scalc 99999999999999999999999999999 + 99999999999999999999999999999
199999999999999999999999999999998
$ echo $?
0
$
```

9. **(0 Points) The only functions that you are allowed to use in this assignment are `printf` and `putchar`. Violating this would result in a deduction of 6 points !!**.

HINT

For a basic solution, I recommend that you approach the problem the same way you would have performed addition in elementary school. Starting with the last digits of both operands, perform the addition, move over to the next pair of digits and so forth.

WHAT TO HAND IN

Turn in the C program source code `scalc.c`, named properly. You do not have to zip the file, The file must be uploaded to mycourses. DO NOT turn in the executable `scalc`. TAs will compile your C program on their own as indicated in the problem descriptions above.

MISC. INFORMATION

There is a tester script `mini4tester.sh` that is provided with the assignment that you can use to test how your programs are behaving. TAs will be using the exact same tester script to grade your assignment.

```
$ ./mini4tester.sh
```

However, it is recommended that when you start writing your program, test it yourself first with the above examples. Once you are fairly confident that your program is working, you can test it using the tester script.

You can compare the output produced by running the mini tester on your C programs to that produced by the mini tester on the solution programs which is given in `mini4tester.out.txt`.

** **Important** ** If your program "hangs" / is stuck while executing it through the tester script and requires TAs to interrupt it, you will loose points associated with any remaining test cases in the tester script.

FOOD FOR THOUGHT!

The following discussion is meant to encourage you to search independently for creative and optimal ways to perform rudimentary tasks with less effort and/or make your program robust and sophisticated. It does not impact the points that you can achieve in the above questions.

- The assignment only requires you to implement the addition (`+`) operator. Can you figure out how to extent your program to support subtraction (`-`) as well? Keep in mind that your result could be negative.
- Can you make your program work properly even if there are leading zeroes for otherwise valid integers as input operands? (Your program should not still print leading zeroes in its output).