COP3275: Programming using C

Programming Assignment #1

Instructor: Roozbeh Ketabi

Fall 2019

1. You have to upload your codes in both Canvas and the Judge system at `https://cop3275.cise.ufl.edu`
2. For guide on how to access the Judge, visit Judge Access Page under Pages section in Canvas.
3. The judge will run test cases against your code and assign it a grade. You can have multiple submission for the same problem but you have to choose which one is the final (we will consider your final submission for grading). Canvas is only used for record keeping.
4. When uploading on Canvas, zip your programs in a zipfile with your ufid as name (nothing more, just 8 digit ufid, for instance 12345678.zip). Add a .txt extension to all your files. i.e. name your files (inside the zipped archive) p1.c.txt, p2.c.txt, and so on.
5. For all the problems input is read from standard input (e.g. read using scanf) and must be written to standard output (e.g. printf).
6. **Don't print extra stuff. Since the assignments are automatically graded, if the output doesn't match the expected output, you will not get the points.** For instance: If the problem is to read a number and return its square, the expected output is a number (i.e. `printf("%d",i*i)`. If you print using something like `printf("square is %d",i*i);` you will not receive any points.
7. The assignment is due on 11:59 pm Wednesday Sept. 25, 2019. There is a 20% penalty for late submission of one day. No submission is accepted beyond that time.

Problem 1: Register Machine [60 Points]

Expected LoC: 50 added to code stub

In this problem we try to mimic a simplified computer architecture. This register-operand computer support two type of instructions: those with all register operands and those with two registers and one immediate value as the third operand. For instance, an ADD instruction looks like "ADD R1 R2 R3" and means to add the value in register R2 with register R3 and store the sum in R1. An example for immediate type instruction is "ADDI R1 R2 100" which means add 100 to the value in register R2 and store the results in R1.

Your program will be given a series of instruction (each in a separate line of stdin). You code must execute those instructions and store their results in the registers. Assume the computer has 8 registers that are of type unsigned integer (denoted as R0 through R7) and they all start with the value of zero. Each line of input will consist of 4 numbers (except for print and exit commands) separated by whitespace: instruction opcode, first operand, second operand and the third operand. The first two operands are always registers so the number will be in range [0,7] (telling us which register to choose). The third operand can be either the register number or the immediate value (determined by the instruction type). Examples:

OP Code	Instruction Type	Example Input Line	Description
0	EXIT	0	Stop the program execution
1	PRINT	1	Print the value of all the registers (space separated) in one line.
2	ADD	2 5 3 1	ADD R5 R3 R1 (R5 = R3 + R1).
3	ADDI	3 5 3 1	ADDI R5 R3 1 (R5 = R3 + 1)
4	MULT	4 5 3 1	MULT R5 R3 R1 (R5 = R3 * R1).
5	MULTI	5 5 3 1	MULTI R5 R3 R1 (R5 = R3 * 1)
6	DIV	6 5 3 1	DIV R5 R3 R1 (R5 = R3/ R1)
7	DIVI	7 5 3 1	DIV R5 R3 1 (R5 = R3/1)
8	MOD	8 5 3 1	MOD R5 R3 R1 (R5 = R3 % R1)
9	MODI	9 5 3 1	MOD R5 R3 1 (R5 = R3 % 1)

- in order to model the registers and access them in a convenient way you either need to store them in an array (covered later in the course), have a preprocessor macro (covered later in the course), or declare a function (which is also covered later) to help you get and set values from the appropriate register. Copy and pasting the same code many times is a very bad design practice. Since we are going to practice writing clean code from the beginning, try utilizing the code stub (pa1_p1_stub.c). Note that if you are familiar with arrays, that may be the easiest/cleanest way to address this issue and you are welcome to try your own way.

- The outputs of the program are only those generated by the print instruction. Note that print instruction does not have any operands.
- DIV, DIVI, MOD and MODI would result in undefined behavior if the divisor or the modulus (operand 3) is zero. To overcome this, ignore these instructions if their third operand evaluates to zero (a register that has value 0 in it or immediate value of 0).
- Note that you may need to declare your variables (the ones simulating the registers) as unsigned long long integers since it is possible have a series of multiplication instructions that result in a number that does not fit in a regular integer.

Examples:

Input	Expected output
3 2 2 10 3 3 3 20 2 1 2 3 1 9 1 1 7 1 0	0 30 10 20 0 0 0 0 0 2 10 20 0 0 0 0 1

Problem 2: Fibonacci Series [15 points]

Expected LoC: < 20

A number in the Fibonacci sequence is the one that is the sum of the two preceding numbers in the sequence. For instance, if the first number is 5 and the second is 3 the third would be 8, fourth would be 11 and so on. For this problem you have to write a program where it takes in the first two numbers of the sequence and another number N in a newline where you have to compute and print the Nth element of the sequence.
Examples:

Input	Expected output
5 10 9	275
1 1 12	144

Problem 3: Sacred Number! [25 points]

Expected LoC: < 40

A number is called sacred if either of the two conditions is true:

- The sum of its digits is even, and the number is divisible by 7.
- The sum of its digits is odd, and the number is divisible by 13.

You are tasked with writing a program that counts the number of sacred numbers among the inputs. Input will consist of the total number of numbers to read N in the first line, followed by N numbers that are whitespace (either space, tab or newline) separated. The numbers will fit in an unsigned integer. Print the count of sacred numbers among them. Examples:

Input	Expected output	Explanation
5 3003512 5810 63201 494 9893	3	5810, 494 and 9893 are sacred. Note that 5 is the number of inputs and must not be considered for sanctity!

- hint: `a % 10` extracts the right-most digit of a positive decimal (in base 10). What does `a / 10` do?
- hint: if the remainder of `a` over `b` is zero, `a` is divisible by `b`.