Simple Calc

Cyber Solutions Development - Tactical

September 28, 2021

Abstract

Your task is build a simple calculator application that will take an equation as an argument, and produce the result to standard out.

1 Requirements

In this assignment, you will build a simple calculator application that will compute simple equations and produce the result. The requirements are below:

1.1 Basic Requirements

1. Written in C
2. Take 3 arguments that forms an equation: (operand1) (operator) (operand2)

   NOTE: operands and operators are separated by a single space.

   NOTE: these are 3 separate arguments - do not wrap with quotes
3. Handle bad inputs such as bad format as described above, divide by zero, or integer overflow
4. Single binary with the usage statement `./simplecalc 1 + 1`
5. Output the correct answer, or some descriptive failure message

1.2 Specific Requirements

1.2.1 Required Operations, Symbols, and Data Types

Addition	+	int32_t
Subtraction	-	int32_t
Multiplication	*	int32_t
Division	/	int32_t
Modulo	%	int32_t
Shift Left	<<	uint32_t
Shift Right	>>	uint32_t
AND	&	uint32_t
OR	|	uint32_t
XOR	^	uint32_t
Rotate Left	<<<	uint32_t
Rotate Right	>>>	uint32_t

NOTE: ROL and ROR are circular operations

2 Deliverables

Your code should have the following file structure:

```
1_SimpleCalc
├── src
│   └── source-files
├── hdrs
│   └── header-files
├── references
│   └── documentation
└── CMakeLists.txt
```

Your code should build and compile with the following shell script ran from the SimpleCalc directory. An example of this would be:

```
// build.sh
mkdir build
cd build
cmake ..
make
```

You are free to make this as fancy as you want, but your build script needs to:

1. be called build.sh
2. run properly when used with the command `bash ./build.sh`

3. live in the base folder of your project (ie 1_SimpleCalc/build.sh)

4. do whatever steps needed to produce a binary named 1_SimpleCalc/build/simplecalc

3 Notes to grader

The purpose of this assignment is NOT to write fancy C code. Use this assignment to achieve the following objectives:

1. Hammer down on the coding standard. By the end of the assignment your mentee should understand the coding standard.

2. Code organization. Ensure your mentee does not implement the solving functionality in main(). Encourage them to implement separate functions for each operator, and probably a separate source file for operations. This will help them throughout the next assignments.

3. Building projects with CMake. CMake is a powerful build system, but requires a learning curve. Use this project to introduce them to CMake.

4. Git tradecraft. Have your mentee branch off of devel, commit their work, submit a merge request to devel, etc.

5. Proper error handling (resource cleanup, printing errors to stderr, etc)

6. Ensure your trainee is taking proper care of completing their JQRs. They should be picking which JQR items are reasonable for a project

Although you ultimately control the trainees JQR completion status, allow them to complete the projects using whichever means they want (within reason). Engineers need to be able to break down problems, and iterate on solutions. If we say "you will do X and Y and Z to solve this" we risk putting the trainees in a box where they can't fully flex their problem solving muscles.

4 JQR Sections Potentially Covered - Mentor/Trainee dependent

NOTE: These projects are designed to give each trainee room to think creatively. Although the reference spec may target specific JQR requirements, it is up to the trainee to pick which JQR items they want to solve, and it should be indicated somewhere in the trainees documentation. It is up to the mentor to determine which JQRs have been satisfied and which need more work.

NOTE: These JQRs are satisfied in the reference solution, which is designed to be a sane and straightforward, but not particularly creative.

- 4.1.3 (all)

- 4.1.4 (all)
- 4.1.6 (all)
- 4.1.7 (all)
- 4.1.9 (all)
- 4.1.10
- 4.1.12 (all)