Computer Architecture
CS 429, Fall 2014
Unique Numbers:
52915, 52920, 52935, 52940, 52945, 52960, 52965

Lab Assignment 0: Introduction to C
Assigned: Wednesday, September 3, 2014
Due: Wednesday, September 17, 2014, 9:00 am

Introduction

The purpose of this assignment is for students to gain basic familiarity with the C programming language. Using a subset of C, this lab asks that you solve various programming problems.

Logistics

You are to work alone on this project, but you may discuss programing tricks and bit-manipulation code with your peers. The entire "hand-in" will be electronic. Any clarifications and revisions to the assignment will be posted on the course Web page.

Hand Out Instructions

> You will find the file `casm-handout.tar` referenced on the homework page of the class website. You will need to download this file so you can use its contents.

Start by copying `casm-handout.tar` to an empty, protected directory in which you will to do your work. Make sure that your directory has permissions that restrict access to this directory to only yourself. For example, you can create an empty directory by issuing the command `mkdir -m 700 lab0`, which will create a new directory immediately in the directory where you are "standing." Then, give the command: `tar xvf casm-handout.tar`. This will cause several files to be unpacked. The only file you will be modifying and submitting is `problems.c`.

By looking at the file `problems.c` you'll notice a place to include your name and UTCS UserID into which you should insert the requested identifying information. Do this right away so you don't forget.

The `problems.c` file also contains a skeleton for your code. Your assignment is to implement the functions described by adding code to the file `problems.c`. You are to use only *straight-line* code (i.e., no loops, no IF statements) and a limited number of C arithmetic and logical operators. Specifically, you are *only* allowed to use the following nine operators:

```
 ! ~ & ^ | + << >> <=
```

You are not allowed to use any constants longer than 8 bits. See the comments at the beginning of the file `problems.c` for detailed rules and a discussion of the desired coding style. Compile the updated (by you) version of the file `problems.c` with the command: `gcc -O2 -m32 problems.c main.c`

Coding Challenge

You have been given a number of templates for a variety of integer coding problems. In the `main` procedure, you may wish to implement a specification for your code. We will certainly check your code on a very large (likely exhaustive) number of tests. Also, for some of the problems, you are expected to include the number of X86 instructions produced by the "gcc" compiler for each routine where there is a comment: "Number of X86 instructions:". To get the number of instructions, you need to compile `problems.c` with the `-S` flag; that is, issue the `gcc -S -O2 -m32 problems.c` command and inspect the assembler produced, which can be found in file `problems.s`.

Do not write lots of code! Pay attention to the maximum number of statements allowed. The main thing you need to do is to think carefully about what is asked and then code it in a dense manner. These problems should be fairly straightforward, but they will cause you to think carefully about using C to answer simple questions. You will only submit the updated `problems.c` file – insert your code there.

The file `dlc` is a (binary) program that you can use to check your solutions for compliance with the coding rules. Note, that this program does not check that your solution meets the number of steps nor that you have used the specific operators listed.

In addition to the problems given, you are asked to create one problem of your own and extend file "problem.c" with this example. It shouldn't be too tricky – but, it should be hard enough that when you give it to a fellow student that it takes your colleague at least a few minutes to solve your problem. Note, the "dlc" program checker does not have any rules for this problem, so you might get some error messages – just ignore them.

Evaluation

Your code will be compiled with `gcc` and run and tested on one of the public Linux machines. Your score will be computed out of a maximum of 100 points based on the following distribution:

90 Correctness of code running on one of the public Linux machines.

10 Style points, based on your grader's subjective evaluation of the quality of your solutions and your comments.

Regarding performance, our main concern at this point in the course is that you can get the correct answer. However, we want to instill in you a sense of keeping things as short and simple as you can.

We have reserved 10 points for a subjective evaluation of the style of your solutions and comments. Your solutions should be as clean and straightforward as possible. Your comments should be informative, but they need not be extensive.

Advice

You are welcome to pursue your code development using any system or compiler you choose. Just make sure that the version you turn in compiles and runs correctly on our UT CS public Linux machines. If it doesn't compile and run – we can't grade it – and your score will reflect that.

Hand In Instructions

We are working on the hand-in instructions. UT is using a new system call Canvas. We will provide hand-in instructions when we have a suitable procedure. Stay tuned to the class Webpage.