Lab 3

Lab 3 assignment is due on Friday, March 6, by 9pm in your svn repository. For help, please check the Labs FAQ first. Specifically, please read the Terminal Troubleshooting and SVN Troubleshooting. Also, refer to C Programming Reference and Coding Conventions for C language details.

Please ask all questions about this assignment during the office hours or post questions on piazza.

This lab is to be done on **a Linux workstation or** in **VM** during **any** time of the week. Plan your time accordingly since late submissions will not be accepted.

Introduction to C programming

In this exercise you are going to work with different data types available in C programming language. You will see the effect of using different data types on the same computation and will understand when to use a particular data type. Along the way you will also learn about the various input and output mechanisms available in C. Don't fret, we are sure you will have lots of fun!

Data Types in C

C has a concept of '*data types*' which are used to define a variable before its use. The definition of a variable will assign storage for the variable and define the type of data that will be held in the location. There are several integral data types, floating point data types for holding real numbers and more. In addition you can define your own data types using aggregations of the native types (you'll learn more about that in ECE 220). Some of these that you will be using are

int

The **int** data type represents a signed integer. For most of your simple computations, this will be your go-to data type. On the lab machines running on 64-bit Intel x86 processors, the **int** type is 32 bits, so it represent any integer between -2147483648 and 2147483647.

Notice that we say **on 64-bit Intel x86 processors**. This is because the size of the integer data types can vary from processor to processor. For example, on the LC-3 (a 16-bit processor), an **int** is 16 bits. In this author's opinion, lack of consistency of integer data types is a significant deficiency of the C language, and is something to be aware of if you write code that you want to run on different processors. (Floating point types are consistent and don't have this problem). For exercises in this class, we can assume that ints are 32-bits. The interested reader can also learn about a header <inttypes.h> , which provides a set of types that are guaranteed to be consistent across platforms.

float

The **float** data type allows you to store floating point numbers which the **int** data type cannot handle. Remember the IEEE 754 floating point representation of numbers? **float** denotes the 32-bit, single precision format, which has a range of -3.4×10^{38} to $+3.4 \times 10^{38}$. Many real world computations result in a floating point result. If you use the **int** data type to hold such results, you will not get accurate answers. This is because the **int** data type truncates everything after the decimal position. So if your answer is 3.14 and you try to hold this in a variable defined as **int**, you will get 3 as your answer. The 0.14 part will be truncated thereby resulting in inaccurate results.

char

This is an 8-bit signed integer (range -128 to 127) which is typically used to store simple characters that are available in the ASCII table. All the characters that can be represented as an ASCII number can be stored in the **char** data type.

bool

Many times, it is required that your program behaves like a switch. It is important in these situations that your program gives either a **yes** or **no** as the result of some computation. This is very similar to the binary logic that you have been studying. To represent and hold such a result, C provides the **bool** data type. The bool data type allows you to hold either *true* (value 1) or *false* (value 0) as the result. *true* can represent yes and *false* can represent no. To use the bool data type, you'll need to include the header <stdbool.h>

Its not over yet !

C provides different data types within the categories of the basic data types. For example you have different kinds of integer data types, each having different properties but having one common link that they are all integers.

Integer data types

- **unsigned int** - Like the name suggests, can hold only positive numbers. Can hold a higher value than normal int as the same number of bits are used to represent only positive numbers now.
- **long** - On the lab machines, this a 64-bit value with a much bigger range when compared to the simple int data type.

- **unsigned long** - On the lab machines, 64-bit unsigned integer.
- **short** - 16-bit integer
- refer to C Programming Reference for the complete list

Float data type

- **double** - provides more precision than the simple float data type. Up to 10 decimal places of precision.

Numerical computations

Before starting to work on this part, make sure to update your local svn copy. You should have a new directory, called lab3 in your ece120 directory. In this directory you should find file lab3.txt which you will need to modify per instructions provided below and to commit the changes to the Subversion repository. Also, download the program num_comps.c and save it in your lab3 folder. You will modify this code during the following exercise and you will submit this file for grading at the end.

Let's learn to compile and run a program written in C!

Step 1: Compile

There are many steps in the compilation of a program (which you will learn about in other classes), but in ECE 120 we can do them all with one invocation of the compiler "gcc" like this:

```
gcc -g -std=c99 -Wall -Werror -o num_comps num_comps.c
```

What does this mean?

- **-g** means compile with debugging symbols. This will allow the debugger to find the C source line corresponding to machine code (you'll learn more about the debugger later)
- **-std=c99** means use the ANSI 1999 C standard. This is the modern dialect of the C programming language.
- **-Wall** means warn about all possible problems with your code. **Pay attention to warnings the compiler outputs.**
- **-Werror** means fail if there are any warnings. We recommend using this option unless you are absolutely sure warnings you are getting are harmless.
- **-onum_comps** indicates that the program to be built should be called num_comps
- **num_comps.c** is the input file implementing your program, written in C

It is not necessary to give the -o flag when you compile your C code. If you do not specify the name of your output file using the -o flag then the compiler gives the output in a file named **a.out**. Try compiling "num_comps.c" without the -o flag to see for yourself.

Step 2: Execute

Okay so after compiling, go ahead and run the program you produced:

```
./num_comps
```

This will load the program into memory and run it. You now have created a program that you can run just like any other Unix program.

Why the ./ ?
Recall that the Unix command conventions is:

- program [arg1] ...

While the program can be a *path* (containing slashes), it can also just be the name of a program (no slashes) to make a command line user's life easier. When you just specify a name, the operating system searches through standard locations for binaries (such as /usr/bin) to find the program with that name. By default, most Unix systems will not search the current working directory (which is generally a good thing, because you don't want to be accidentally executing something you happen to be in the directory of). So you need to provide a path (something with slashes) to execute a program not in the standard locations. Adding ./ is usually the easiest way to do this. If you move to the parent directory, this would also work:

- lab2/hello-world

Now that you know how to compile and run the program, let's do the Lab!

Modify num_comps.c to perform the following three computations. You perform each of the computations below 4 times, each time using a different data type. Write your answers to each computation in your lab3.txt file. **Don't forget to commit it to your svn when you are done**, otherwise we will have nothing to grade.

1. -22 * 33
2. 12 / 3

3. 10 / 3

The "*" operator in C multiplies two numbers. The "/" operator divides two numbers.

Carry out each of the above mentioned computations by storing the first and the second values and the result as the following data types. To do that, you will need to update num_comps.c accordingly. Make sure that you update the printf command to match the data type!

1. **int**
2. **unsigned int**
3. **float**
4. **double**

Perform the following two computations once using the **int** data type. Write your answers to each computation in your lab3/lab3.txt file

1. 12 % 3
2. -11 % 3

Make sure to update the printf specifiers like "%d" to match the data types you are using. For *float* and *double* use %f for *unsigned int* use %u

That's it for this Lab!

Hand In Instructions

Submit both **lab3.txt** and your last version of **num_comps.c**. Since num_comps.c was not under revision control, you will need to add it first:

```
cd ~ece120sp20/lab3
svn add num_comps.c
```

and only then commit your work:

```
svn commit -m "done with lab 3"
```