CS 240 Summer 2016

Lab 1: Getting acquainted with C and its system environment (145 pts)

Due: 06/20/2016 (Tue.), 11:59 PM

Objective

The objective of this introductory lab is to familiarize you with the system environment of C programs under Linux.

Reading

Read chapter 1 from Kernighan&Ritchie (textbook). The introduction provides an overview of C programming. It is a bit dense. You are not expected to understand all the material. What you need to understand is the material discussed in class, with the textbook serving as a supplementary resource.

Lab1 code base

The C code base for lab1 is available as a tarball at

/u/data/u3/park/pub/cs240/lab1.tar

on the Linux machines in LWSN B148. Copy it to your working directory and untar it using

% tar xvf lab1.tar

Under lab1, you will find 7 subdirectories that contain versions of the example code discussed in class. As noted in class, always compile and run the final version on the pod machines in LWSN 148.

Problems [145 pts]

Problem 1 (20 pts)

Compile the code in directory v1 by running

% gcc main.c

What is the nature of the output a.out generated by gcc? What are the three steps performed by gcc that results in a.out? What does each of the steps accomplish?

How does gcc react if you omit the type declaration of main()? Will gcc generate an executable? What happens if you declare main()'s type as void?

Problem 2 (20 pts)

The version in directory v2 is essentially the same as the one in v1 except that printing of the addition result is facilitated by the libary function printf(). printf() outputs to standard output which is, by default, the terminal (i.e., display or screen) associated with a PC in our lab. Where is the header file stdio.h located on our Linux lab machines in LWSN B148?

What part of "result of %d + %d is %d" is literal (just characters to be printed as is) and what part is reserved, i.e., part of formatting control of C that determines how content of a variable is output? What happens if you run gcc with option -c? Can you execute the output generated by gcc? Explain.

Problem 3 (30 pts)

The version of v3 makes a further enhancement such that the two numbers to be added are provided as input via keyboard input which, by default, is the standard input of a Linux PC. What is the role of & (i.e., ampersand) in

scanf("%d %d",&x,&y)

and why is

scanf("%d %d",x,y)

incorrect? Compile the code as is, and run it to verify that it works correctly. Modify the code by removing the two ampersands and compile main.c using gcc. What does gcc say after the modification? What happens when you run a.out? Explain.

Problem 4 (20 pts)

Why does printf() in v3/main.c not use ampersand & before the variables x, y, z? What happens if you add & before the three local variables? Explain what gcc does and what happens when you execute the executable produced by gcc.

Problem 5 (40 pts)

v4 contains a floating point (i.e., real number) variation of v3. Compile and test with real numbers (say, 7.111 and 4.222) to check that it works correctly. In v5, the code is made more modular by implementing the addition part of the app code as a separate function, myadd(). Since myadd() is essentially a one-liner, the separation doesn't buy much in terms of clean design. However, the principle is clear: if the calculations were more involved, putting the instructions in a separate function makes the design more modular. myadd() takes the values to be added as its two arguments and returns the result to the calling function. Hence the caller is main() and the callee is myadd(). When passing the two arguments to myadd(), why do we not use ampersands, that is, myadd(&x,&y)?

Suppose instead of communicating to the caller the subtraction result using a function return, we want the callee to directly update the variable z in main() with the computed result. One way to do it is to change

```
z = myadd(x,y);
```

to

```
myadd(x,y,&z);
```

Why does adding &z as the third argument of myadd() enable the callee to update the value of variable z which belongs to main()? If this method of communicating between caller and callee is employed, C requires that the code of the callee be updated as follows:

```
int myadd(float a, float b, float *c)
{
  *c = a + b;
}
```

The * symbol preceding argument c in the callee specifies that c contains an address (not value). Thus we are telling the callee myadd() where in RAM (i.e., memory) z is located, i.e., its address. Since the callee has this information, it can now, if it wants to, go to the address and update the content at that address. This is done by prepending c with * and performing the addition *c = a + b, instead of c = a + b. Make the modifications and compile the updated main.c. What does gcc say? What additional modification do we need to make to have a correct code? Figure it out on your own and if you get stuck the TAs will help. Since myadd() does not return a value, what type declaration can we assign myadd() to reflect/highlight this feature? Compile the updated code and test that it runs correctly.

Problem 6 (15 pts)

Describe what the modification performed in v6 is. How should the code of v6 be compiled? What final modification is carried out in v7? Explain what the difference between

```
#include "mydecl.h"
```

and

```
#include <mydecl.h>
```

is. What happens if you replace by angle brackets of stdio.h by double quotes? What happens if you replace the double quotes of mydecl.h by angle brackets?

Bonus problem (15 pts)

Modify the version in v5 so that all arguments are passed as pointers. Explain the changes you are making in myadd().

Turn-in instructions

Electronic turn-in instructions:

i) For problems that require answering/explaining questions, submit a write-up as a pdf file called lab1.pdf. You can use your favorite editor subject to that it is able to export pdf files which many editors, including freeware, do. The TAs need to be able to spend their time evaluating the content of your submission, not switching between editors or hunting down obscure document formats which wastes valuable time and is in no one's interest. Thus only pdf files are accepted and other formats receive zero credit, even if a pdf file is submitted after the due date.

ii) For the Bonus Problem which is entirely optional (the bonus points count toward the 50% that lab assignments contribute to the course grade), create a tar file of your v8 directory by running

```
% tar cvf v8.tar v8/
```

and submit the tar file v8.tar.

iii) We will use turnin to manage lab assignment submissions. Create a directory named mylab1 and put your pdf file lab1.pdf and v5.tar which contains your modification to the code using pointers in mylab1. Include v8.tar if you have done the bonus problem. Go to the parent directory of mylab1 and type the command

```
turnin -c cs240 -p lab1 mylab1 -v
```

Please note the assignment submission policy specified in the course home page.

Back to the CS 240 web page