Lab #1 Assignment

CSE 1320-003

Fall 2013

Design Document Due Date: **September 3, 2013 at noon (12pm)**

Lab Due Date: **Sept. 5, 2013 at noon (12pm)**

(see instructions on website for how to turn this in - “lab submission info”)

Grade value: 4% out of 100% for all grades

Lab #1 has three parts, I, II, and III – parts I and II are worth 1% each and part III is 2%

Objective: The goal for this lab is to provide an opportunity for practice of C program development and the C programming topics covered in Chapters 1–4 and parts of later chapters in Foster and Foster as covered in class. It is assumed that the student is familiar with the programming concepts but not the C syntax. This assignment is designed to help student practice the skills they will need for all the labs in this class. The three parts of the assignment are:

Part I: Practice in accessing and using the omega Linix (Unix-based) operating system including creating a secure shell, transferring files to omega, creating directories and navigating in them, compiling a C file that was transferred to omega, running the compiled program, and recording it all in a script file.

Part II: Practice in editing a C program file that I give you using a line based editor on the Unix system, compiling the C file, running the program and then debugging it. This program will have intentional errors that you must fix.

Part III: Adding to the program from Part II by writing additional functions that will allow the program to do more actions.

Every lab assignment allows students to practice program development, debugging, and testing. All of these skills are crucial to success in Dr. T’s class. The format of this assignment is not similar to the assignments that will be required for the rest of the semester but labs from previous semesters may be reviewed to get an idea. A student who is unsure of their skills should start early and should plan to ask the instructor or TA about any concepts they are unsure of.

Topics:

Program design

Modular programming structure

Error checking

Programming style

Compiling code,

Executing code,

Debugging and testing.

Plan: Dr. T gives an overall problem that students work on all semester. Each lab assignment implements specific concepts. Succeeding assignments will modify and extend previous assignments.

Part I Assignment:

In this part of Lab #1 you are going to practice using the tools you need for the rest of the semester. After Lab #1 is complete, you will not be required to use ONLY these tools but you are expected to ALWAYS use some of them. (More on this later.) For Part I, here are the steps you must complete:

1. On whatever computer you are planning use for your programming assignments during the semester you must have a way to securely connect to “omega” which is the Unix-based system that you will be running your programs on. If you will be using a PC, you must download SSH Secure Shell Client from the UTA OIT software download page. If you are using a Mac, you can connect to omega using Terminal to create a secure shell connection.

2. Once you have ssh downloaded (or Terminal open) you should open an ssh connection window to your account on omega. You should be able to log in with your 7 character NetID (abc1234) and then you will be asked for your password. If you have a problem with this, check with the OIT Help Desk in the main library.

3. Once you are logged in to omega you should list your directory (type: ls) to show the files that are in your top level directory.

4. Now start a script file session to capture the rest of the actions that you do for Part I. See the “lab submission info” on the class website for instructions on how to do this. Your script file will be turned in as your proof that you did Part I.

5. Now that the script file session is running, list your directory with details (ls -l) .

6. Back on your PC or Mac use WordPad or TextEdit or some similar simple editor to type the example file called output.c into a plain text file (Don't cut and paste – just type it). You may shorten the comments but you must include the same number of comment lines. You should print out another statement instead of “C by Discovery” so put your own statement in double quotes and do not add any lines. DO NOT use MS Word. (It makes your life complicated for this lab assignment.) Save the file with the name abc1234lab1output.c and make sure it is saved on your desktop. (“abc1234” is your NetID)

7. Now using the ssh (or Terminal) open an sftp (secure file transfer protocol) connection window to omega. Keep the ssh window open too. You’ll see your computer’s files on the left and the omega directory on the right in the sftp window. In the left window, make sure you are showing the files on the desktop – you should be able to see abc1234lab1output.c in the list. Drag abc1234lab1output.c from the left window to the right window to transfer (copy) the file from your computer to omega. On the Mac you can also open another Terminal window, do the command

>*sftp* [*yourlogin@omega.uta.edu*](mailto:yourlogin@omega.uta.edu)

Enter your password when asked. Then you can transfer the file with

>*put abc1234lab1output.c*

8. Go back to the ssh window now and do ls again. You should see that the file abc1234lab1output.c is now in the directory.

9. Just for practice now we are going to create a subdirectory on omega. This is just the same as making a subfolder inside a folder on a PC or Mac. To make the subdirectory, we will use the command to make a directory (mkdir) and we’ll call the directory C1320. So in the ssh window type: mkdir C1320 to create a new subdirectory.

10. Do ls again to see that C1320 is now listed.

11. We’ll move the abc1234lab1output.cfile to the new subdirectory with the command: mv abc1234lab1output.c C1320/ (The slash tells Unix that C1320 is a directory)

12. Do ls one more time and you’ll see that abc1234lab1output.c has disappeared from the current directory.

13. Now let’s change our location from the current directory to the subdirectory. We do that with the change directory commend (cd). So type: cd C1320

14. Do ls again and now you’ll see that we only have abc1234lab1output.c in this directory.

15. Two more tasks and we’re done with Part I. We are ready to compile our little C program. I hoped you typed it very carefully so that it will compile right the first time. Before you compile it, let’s just take a look at what you typed in. Unix has a command that will print out a file to the screen. For various reasons the command is called the concatenate command but you only have to type cat. To see what your abc1234lab1output.c file has in it type: *cat abc1234lab1output.c* This will print the entire file to the screen.

16, To compile a C program on omega you will be using the GNU C Compiler (gcc). The compiler will only work with files that have the extension .c so if you try to compile something else it won’t like it. To compile your program type: *gcc abc1234lab1output.c* If you get some sort of message, that means it did not work and you had an error. If it works, it does not give any message. (I’ll get back to the error case.) [NOTE: All programs in Dr. Tiernan’s 1320 class all semester MUST be compiled on omega using gcc and must compile and run without errors or warnings.]

17. Do ls again. If the compilation worked you should now have a file called a.out in your directory. This is the executable file the gcc creates that has your program in it ready to run.

17error. If you got a message from gcc and/or there is no a.out, go back to your original file on your PC/Mac and make sure you typed it exactly as it shows in the book. Fix anything that needs it, eliminate any blank lines at the bottom of the file, and do the same steps again.

18. Once you get an a.out file then you need to run your program. The a.out file is the executable file so you run it by just typing its name: *a.out* Once you do this, the program should print whatever message you substituted for C by Discovery.

19. When that is done, you should end your script file session and you are done with Part I.

Part II Assignment:

In this part of Lab #1 you are going to practice using one of the editors on omega. You must use this editor for this lab but you will not be required to do so for the remaining labs. The editors on omega that you will access through ssh are line oriented editors. You will use the keypad to do your editing tasks rather than the mouse. There are many Unix editors but the most common ones are vi (or vim), pico (or nano), and emacs. You will find links to manuals for all of these editors on the class website. You will not be able to guess your way through using these editors so I highly recommend having a webpage open with some reference info or doing a tutorial online before you do this part of Lab #1.

The goal here is twofold: practice in editing a C program file that I give you using a line based editor on the Unix system and then debugging the program that will have intentional errors that you must fix. So here are you steps to follow:

1. I will have put a file called CLab1Fall13err.c on the Course Materials page of the Blackboard site for this class. The file should be saved to your PC/Mac desktop. Don’t open the file on the PC/Mac. Use the same process as part I above to transfer the file to omega with sftp and open an ssh window to omega. If you haven’t ever logged in to Blackboard, this is your chance to practice that too.

2. Start a script file session on omega and do ls to show the CLab1Fall13err.c file in your directory.

3. Since it is a C file try compiling it with gcc just like before. If everything went as expected, you should have two lines of warnings and four lines of errors. So now you’re going to do some debugging.

4. First figure out how many things it is really complaining about. Is it six things or are there six lines of info but really less things to deal with? What does the first warning say? “Incompatible implicit declaration of…” – OK, incompatible means it doesn’t agree with you, “implicit declaration” means that you tried to use it yourself without telling C where you got this thing. And what doesn’t it like? “built-in function ‘printf’ “ – hmm, it liked printf before so what did we do before that is missing in this program? Go look at abc1234lab1output.c and see if there is some info that would tell C where we want it to find ‘printf’ <You can check the book to help you too.> OK, I know you’ve figured out what is missing for the printf problem so let’s go put it in the program.

5. Now instead of editing the program on the PC/Mac and moving it each time, we are going to edit it on omega directly. Now, I want you to edit this and then go back and fix the other errors before you recompile so you should open a second ssh window and use that window to do your editing. Open the second ssh window connected to omega and then open CLab1Fall13err.c in the editor you wish to use. For example, to use emacs I would type: *emacs CLab1Fall13err.c*

6. Once your editor opens you will see the contents of the file CLab1Fall13err.c and you can edit it. Use the arrow keys to move the cursor to the location you want to edit. Use your reference manual to figure out the editing commands. When you have added what you need to fix the printf warning from step 4, then save the edited file but keep it open in the editing window so you can make more changes for the other errors.

7. OK, back to the ssh window with the gcc errors. Check out the second warning. Do you think you will have fixed this one too? Edit the file if you think you need to.

8. Now the first error – at line 15 “expected ; before sq” If you look at ‘sq’ in line 15, what comes right before it? Is anything missing, like maybe a ‘;’? Remember that C doesn’t care about the physical line something is on, it looks for the right symbols to tell it when things end. Decide what needs to be done to fix this and go make that change in the edit window. Don’t forget to save the file after you edit it.

9. Now the next error – ‘sq’ undeclared” on line 17. So if ‘sq’ is undeclared and that causes an error, what will fix it? Where should that fix go? Go to the editing window and make that change.

10. The next error line – “(Each undeclared identifier is reported only once” followed by “for each function it appears in)” . It also refers to line 17. Is this a new error or a message about something that it already told us? Do we need to do anything else to fix this?

11. Once you’ve made the changes you think you need and saved the file, what do you have to do to see if your changes worked? Do that.

12. If you changes worked, then run the program.

13. Once the program has run successfully, then see if you can enter numbers that will make the program break (not give correct results or crash). This is called ‘testing’. ☺ After you’ve broken it (or tried three times and didn’t figure out how to) then end the script session and you are done with Part II.

Part III Assignment:

Now, I’m going to stop giving steps and just give the requirements for this third part. You are going to take the program CLab1Fall13err.c that now works and you are going to expand it. First, you should make a copy of the file so that the original file that you have fixed doesn’t get messed up. To copy on Unix you give the copy command cp, then the name of the original file and then the name of the file you want to copy it into. So we’ll use: *cp CLab1Fall13err.c CL1P3.c* Now you will use CL1P3.c to do the remainder of Part III.

For Part III you will be changing your program into one that is more meaningful and useful than just a sum and square. You are going to

work up an estimate for what it would cost to start a small garden store based on the size of the store, the kind of stock, the start up costs, and staff costs. (If you really own a business, you will quickly realize that this is not going to be a very realistic example but the idea is for you to practice your programming skills. ;) You will ask the user for the number of rooms in the store building to be rented; the width of a room in feet (we will assume each room is the same size and is square for our example); the type of flowers to stock represented by a single number with 0 meaning annuals, 1 meaning high water use perennials, and 2 meaning xeriscape perennials; the fixed start up costs such as the cost of your business license and any fees you pay at the start; and cost for staff in terms of how many hours per month you expect your full-time employees to work, the number of employees you have and the hourly wage you will pay the employees. You must let the user do multiple sets of calculations if they want, i.e. keep asking for new costs and wages until they are done. You must also check that all the input they give is valid, i.e. there will not be a negative number of work hours.

Simplifying assumptions for Lab 1 Pat III:

1) If your instructions tell the user to enter numbers, then you do not have to check that the input might be a letter instead of a number. If the user does not follow your instructions to enter a number then any data type input error is the USER’s fault and you will not be penalized in this lab.

Given your starting program from Part II, you will need to make the following changes:

1. Give the user a welcome message when the program starts.

2. Change the variable names so that one variable is for the number of rooms and another variable is the total estimated garden store cost. Then add other variables with meaningful names as needed. Declare all your variables for the function at the top of the function.

3. Ask the user to enter the number of rooms then ask the user to enter the average room width in feet.

4. Verify that the values that were entered were positive numbers. If the number of rooms is not positive, use 5. If the width is not positive, use 16.

5. Ask the user to enter the rental rate per square foot per year. If this number is positive, then calculate the total yearly rental for the rooms of the store. If the number is not positive, print a message for the user and ask them to re-enter the number until they enter a positive value.

6. Call a function to calculate the fixed startup costs. The function should ask the user for the cost of a business license, the total cost of the utility connection fees, the cost of a business permit, and an amount to have as a start up fund.

7. In the function verify that all of the values that were entered were positive numbers. If any cost is not positive, use $1000 for that cost.

8. The function should add the costs up and return the total fixed start up costs.

9. Ask the user for the code number identifying what kind of stock they will have, verify this value and save it in a variable with a meaningful name. Then ask the user for the average price of one stock item in dollars and cents. Multiply this price times 10,000 and save this as the starting stock cost.

10. Ask the user how many hours per month the average garden store employee will work, the expected number of employees, and the average hourly wage for an employee in dollars and cents. Use this information to calculate an expected yearly staff cost.

11. Calculate the garden store startup costs as the sum of the total yearly rental, the fixed startup costs, the starting stock cost, and the expected yearly staff cost.

12. Print out all the intermediate and resulting values in one or more meaningful messages.

13. Ask the user if they would like to calculate another startup cost and read in a response from them. If yes, loop back to the top and repeat. If no, end the program with a thank you message. You will need a variable to use for this loop so don’t forget to declare it.

Once your program is modified, compile it, run it and test it. Your program must compile and run without warnings or errors when you turn it in.

You should design your program in advance before you begin writing code. In this lab and all other labs you will document your design and turn in the design document before the lab assignment is due. The goal of the design document is to assist you in developing the actual program.

Read ALL of the assignment before you start trying to develop the program.

Be sure to check the DEDUCTIONS section at the end of this assignment to avoid penalties. You may NOT use global variables, the exit command, goto, break (except in a switch), continue, arrays, structs, string variables, passing by reference, pointers, or linked lists. Ask Dr. Tiernan BEFORE you use any concept not discussed in class or in the material covered in the book.

Implementation Requirements:

For Part III WRITE A DESIGN DOCUMENT FIRST. The design must include

a) all the functions you expect to write,

b) brief (one line) descriptions of each function, and

c) some indication of which function calls what other functions.

The design document may be written as lines of text or as diagrams (such as a diagram that start with the main function at the top and all others below it) or as some combination of those, but it must include the information listed for a), b), and c) above. Each function should accomplish one main purpose and each function at a lower level should have a more specific purpose than the function that calls it. Be sure to include all the functions that are described in this lab assignment. This design documentation will normally be turned in for the lab and a design will be required to be turned in for all labs after Lab 1. See the website for the DESIGN DOCUMENT due date. It is PRIOR to the lab due date.

The program should use the following data structures:

Single variables

The program should use the following control structures:

Function calls to perform tasks

A while or for loop to allow user to give multiple inputs

If, if-else, nested ifs, or switches to error check and implement the menu options

The program should NOT use:

arrays

string variables

structs

global variables

exit

break (except in a switch)

continue

pointers

any topic not covered in class before the lab DUE date unless approved by the instructor

Programming practices:

Comment your code! Use headers as described below, use line comments and use block comments. Line and block comments should explain the meaning of the code. As an example compare the following examples of code with comments:

Example 1:

/\* Find A by multiplying L times W \*/

A = L \* W; // A is the product

Example 2:

/\* Find the area of the rectangle by multiplying length times width of the sides \*/

A = L \* W; // A is area, L is length of one side, W is length of perpendicular side attached to L

While Example 1 has comments, the comments do not tell us anything extra about the code. The code itself tell us that A is the product of L\*W. In Example 2, the comments are meaningful and explain the goal of the code and the meaning of the variables. Make your comments like Example 2 not Example 1.

The program should have a program header which gives, at least, your name, the number of the lab assignment, your class and section, the assignment date, the due date, and a description of the program. If multiple files are used, each file should contain a similar header.

Each programmer-defined function, i.e. each function you write, should have a function header similar to those used in the examples in the textbook. This header should include at least the function name, the purpose of the function, and its inputs and outputs.

The program should be implemented as a set of functions with a main routine and at least one other function such as one that calculates the yearly rental cost. The purpose of functions is to divide the problem into small tasks, each one assigned to its own function and then to call the functions from main() or from another function when appropriate. Do not code the entire program in main!

The program should perform the following actions in the given order:

Declare and initialize the variables

Print a welcome screen for the user that introduces the system

Get the needed input values from the keyboard or file

Print the appropriate outputs

Let the user make additional choices until the user indicates that they are finished.

Output requirements:

The program must be run and the output recorded in a script file from OMEGA using the gcc compiler. No Exceptions! If you do not know how to create a script file, it is your responsibility to ask the TA, look for help on the class website, or OIT how to use this function.

Testing:

This program must be run with two different sets of test data for the input data. You must create two different data sets and run your program with both of them. I will give you a small amount of sample data at the end of this lab for you to use as a model. You may run your program two times within a single execution or you may execute the program two different times so that you have a total of two different data sets. The sample data sets that you create must meet the guidelines given in the problem definition. Your test runs should demonstrate all the choices that are available in your program.

NOTE ABOUT ERRORS:

Programs turned in for credit MUST compile and run WITHOUT any compilation errors or runtime errors using the gcc compiler on omega. No other compiler may be used for the compilation for credit. No other operating system may be used for the compilation and/or execution for credit.

*Compilation errors* occur while the program is being developed and they prevent the program from compiling correctly. Programs compile correctly when running

>gcc myprogram.c

gives no error messages.

Programs may be turned in for credit when they are partially complete but all completed functions must run without any errors. A program completes without runtime errors if it ends only when the user selects for the program to end and it correctly prints the exit message. Partially complete programs that run correctly for all implemented menu choices will receive partial credit.

Any other type of ending is a *runtime error* or a *"crash"*. A program has a runtime error if it compiles and runs but then crashes in any situation, i.e. if there is any set of choices the user can make that will make the program crash. It is your responsibility to test all possible choices in your program to make sure that none of them cause a runtime error. The goal of creating input test data and running your program with it should be to test all of the various choices in your program to make sure all of them terminate correctly. (Remember that for this lab, you do NOT have to check if the user entered letters instead of numbers but you do have to check that the numbers that are entered are valid, i.e. meet the requirements given.)

**Labs which have errors in them and do not terminate normally will receive an overall grade of 0 (zero)**

Grading Scale:

**Tasks to accomplish: (50%)** Parts I and II

Part I - (20%)

Script file initially shows ls -l without file abc1234lab1output.c in directory (1 point)

Script file then shows ls with file abc1234lab1output.c in directory (1 point)

Script shows:

mkdir C1320 (2 points)

ls //should show C1320 now (1 point)

mv abc1234lab1output.c C1320/ (2 points)

ls //should show that abc1234lab1output.c is gone (1 point)

cd C1320 (2 points including the ls command after)

ls //to show that abc1234lab1output.c is here

cat abc1234lab1output.c // print the program to the screen (2 points)

gcc abc1234lab1output.c //(2 points including the ls command after)

ls //to show a.out

a.out (2 points)

//program output should be shown (2 points)

Everything for Part I done in one continuous script file (2 points) - this script can have typos but is all one single script session instead of multiple sessions.

Part II - (30%)

Script file initially shows ls to show the CLab1Fall13err.c file (1 point)

Script shows:

gcc CLab1Fall13err.c // with two warnings and four errors (2 points)

// script might show editing commands but does not have to

gcc CLab1Fall13err.c // with fewer warnings and errors (repeat until no more warnings or errors) (12 points)

a.out (5 points)

// run with test data

a.out // or if program allows user to enter new data, this counts as second test (3 points)

// try to "break" the program with numbers

a.out // or third iteration (3 points)

// may have discovered that very large inputs will cause "inf" output which "breaks" the program (2 points)

Everything for Part II done in one continuous script file (2 points) - this script can have typos but is all one single script session instead of multiple sessions.

**Code: (34%)** Part III

Program header and function headers for all functions (3 points)

Comments (line comments and block comments) (3 points)

Modularity (division of the problem into small tasks, each one assigned to its own function and called from main() or from another function when appropriate--do not code the entire program in main!) (5 points)

Style (indentation, consistency, meaningful identifiers, lateral separation of code from line comments, etc.) (4 points)

Correct declaration and manipulation of the variables (5 points)

Correct use of required control structures - loop to let user run program again etc.(4 points)

Correct function structure as required - main and one other function (4 points)

Proper implementation of data input and input error checking (6 points)

**Output: (16%)** Part III

User clearly understands what is being requested for input (3 points)

Calculate tasks perform correctly (4 points)

Input verification shows valid values and list of inputs correctly saved and printed (2 points)

Output gives clear information to explain the values to the user (4 points)

Output contains all the given test data and one additional data set (3 points)

**Grading Deductions:**

Use of global variables, arrays, structs, or pointers will result in an overall grade of 0 (zero) [-100 deduction]

Use of the exit, break (outside a switch), or continue command will result in an overall grade of 0 (zero) [-100 deduction]

Labs which have errors in them and do not terminate normally will receive an overall grade of 0 (zero) [-100 deduction]

Late submission of softcopy of code and/or script file to Blackboard will result in an overall grade of 0 (zero) UNLESS

student has obtained prior instructor approval [-100 deduction]

Use of C language elements not yet discussed in class by the lab due date will result in potential deduction of points - discuss with instructor before using.

Miscellaneous: If you have questions, e-mail Dr. T ([tiernan@uta.edu](mailto:tiernan@uta.edu)) and the TA (see website for contact info)

Sample input:

Number of rooms: 4

Width of room: 40 feet

Yearly rental rate for one square foot: $85

Cost of a business license: $120

Total cost of the utility connection fees: $250

Cost of a business permit: $500

An amount to have as a start up fund: $10,000

Code number identifying what kind of stock they will have: 2

Average price of one item of stock: $3.50

Employee hours per month: 100

Number of employees: 3

Average hourly pay rate: $15.45