Lab Assignment 3: Branching, Loops and Functions

Due: Friday, September 21, 11:59 PM

Description:

The purpose of this assignment is to practice writing code that calls functions, and contains loops and branches. You will create a C program that prints a menu and takes user choices as input. The user will make choices regarding different rectangles that will be printed to the screen.

General Comments:

This program will be graded using a script file. This means that, for full credit, the specifications must be followed exactly, or else the input given in the script file may not match with the expected output.

Instructions:

The source file for this assignment will be named lab3_<username>_<labsection>.c and will include the required header information at the top:

```
// <Your Name and G Number>
// CS 262, Lab Section <Lab Section Number>
// Lab 3
```

Your code must contain at least one of ***all*** of the following control types:

- nested for() loops
- a while() ***or*** a do-while() loop
- a switch() statement

- an if-else statement
- functions (see below)

Important! Consider which control structures will work best for which aspect of the assignment. For example, which would best to use for a menu?

The first thing your program will do is print a menu of choices for the user. You may choose your own version of the wording or order of choices presented, but each choice given in the menu must match the following:

Menu Choice	**Valid User Input Choices**
Enter/Change Character	'C' or 'c'
Enter/Change Number	'N' or 'n'
Print Rectangle Type 1 (Border Only)	'1'
Print Rectangle Type 2 (Filled in)	'2'
Quit Program	'Q' or 'q'

A prompt is presented to the user to enter a choice from the menu. If the user enters a choice that is not a valid input, a message stating the choice is invalid is displayed and the menu is displayed again.

Your executable file will be named lab3_<username>_<labsection>

Your program must have at least five functions (not including main()) including:

- A function that prints the menu of choices for the user, prompts the user to enter a choice, and retrieves that choice. The return value of this function will be void. It will have one pass-by-reference parameter of type char. On the function's return, the parameter will contain the user's menu choice.

- A function that prompts the user to enter a single character. The return value of the function be a char and will return the character value entered by the user. This return value will be stored in a local variable, C, in main(). The initial default value of this character will be ' ' (blank or space character).

- A function that prompts the user to enter a numerical value between 1 and 15 (inclusive). If the user enters a value outside this range, the user is prompted to re-enter a value until a proper value is entered. The return value of the function be an int and will return the value entered by the user. This return value will be stored in a local variable, N, in main(). The initial default value of this character will be 0.

- Two "Print Rectangle" functions. Each function will take the current integer value N and character value C as input parameters. The return values of these functions will be void. The functions will print rectangles of N lines and columns using the input character C. The Border Only function will print the rectangle with the just the border. The Filled In function will print the rectangle as a solid rectangle. For example, if the integer value N = 6, and the character value C = '*' and the Filled In type is called, the following rectangle will be printed:

```
******
******
******
******
******
******
```

If the Border Only rectangle is to be printed, then the following rectangle is printed:

```
******
*    *
*    *
*    *
*    *
******
```

Your program must conform to the CS 262 Programming Style Guide. All variables must be declared within the body of a function, and if necessary, passed as parameters to other functions.

Suggested Steps to Complete the Assignment:

You are not required to use the following steps to write your program or even pay attention to them. However, following the steps will likely help you complete the lab faster and with fewer debugging issues. If you do use the suggested steps, you should test your program thoroughly to ensure each step works correctly before moving on to the next step.

1. Create a directory named "Lab3_<username>_<labsection> and make it your current working directory.
2. Copy your Makefile from Lab2 into this directory and modify it to work with the Lab3 executable
3. Create a source file with only your main() function and the standard #include files. Compile and run (it won't do anything).
4. Write a menu() function, called from main(), that prints the menu only (do not add any additional functionality or parameters at this point). Compile and test it to ensure it works properly.
5. Add code to the menu() function to prompt for retrieve user input (allow any character). Compile and test your code.
6. Add a pass-by-reference parameter to your menu() function that will retrieve the character input within that function and make it available for use within the main() function. This function must remain a void function and not return a value.
7. Enclose the menu() function within a loop that will exit the program when the Quit program choice is entered. Compile and test the logic of your code to ensure that the loop only exits on proper input.

8. Create four "stub functions" for the four other (non-Quit) choices. Put a print statement such as "This is function EnterChar()" or some other informative statement in the body of the function. For functions that return a value, return a specific character 'X' or number. This will be changed when the function is filled in.
9. Within the loop in main(), create the logic to call each of the four stub functions based on input from the menu choice (and handle incorrect input choices). Test this logic by observing the output of the stub function statements.
10. Fill in the logic and code for each function. Note that the Filled In Printing function is probably a little easier (logically) to write than the Border Only Printing function, so you may want to write it first. Once you have the Filled In function complete, think about how you would need to change it to print only the borders. This is the part of the lab (and the course) where you develop your skills to create algorithms that solve specific problems. These kinds of skills are not specific to C or any other language, but the methods used to implement these algorithms *are* language specific.
11. Test your program thoroughly.

Testing your completed program:

A sample input file is included with the assignment. To test your program, you can use Unix redirection to enter the choices for your program automatically. To do this, simply type the name of your program, the "<" character (Unix stdin redirection), and the name of the sample input file:

```
bash$ lab3_203_astudent < sample_input.txt
```

This will run your program with the input values provided in sample_input.txt. You can also redirect output to a file with the ">" character (Unix stdout redirection):

```
bash$ lab3_203_astudent < sample_input.txt
> test_run.out
```

The above command will run your program using the sample input choices and save the resulting output in the file *test_run.out*.

Make sure that your program runs correctly with the *sample_input.txt* file. When grading, the TAs will use a similar file to ensure your program works properly.

Adding to your Makefile:

You will now edit your Makefile so it is more generic. If you haven't already, copy your Makefile from Lab 2 to your Lab 3 directory. Then add the following line to the top of your Makefile (after the comments with your name, lab section, etc.)

```
# The compiler: gcc for C programs, g++ for C++ programs, etc
CC = gcc

You will now create a target that will reduce the amount of editing you need
to do to have your Makefile compile other programs. Add the following line
below the CFLAG line you added for Lab 2:
TARGET = lab3_<username>_<labsection>
```

Create a new line for your `all` target:

`all = $(TARGET)`

and replace your old all target with the following line:

```
$(TARGET): $(TARGET).c
```

Next, edit the old compile line so that it references the new variable names you created:

```
$(CC) $(CFLAGS) -o $(TARGET) $(TARGET).c
```

(Don't forget the tab before the $(CC)!)

Finally, edit the clean: target so that it removes the executable by its variable name rather than the explicit name:

```
rm $(TARGET)
```

(Don't forget the tab before the rm!)

lab3_<username>_<labsection>:
lab3_<username>_<labsection>.c
	gcc -o lab3_<username>_<labsection>
lab3_<username>_<labsection>.c $(CFLAGS)

Make sure you have placed a tab (no spaces!) before the gcc in the second line.

Once that is done, you can compile your executable by simply running the make command. You do not need to have the executable name as a parameter anymore. Test this to ensure that your Makefile works correctly.

Submission:

You will create a typescript file containing a listing of your code (showing that it compiles without errors) and a run of the program after compiling. Then create a tarfile containing your source code, Makefile and typescript for submission. Follow this procedure to produce the necessary files:

1) If you have not already done so, create a directory named lab3_<username>_<labsection>. Move your source file and Makefile to this new directory, and change directories to make it your current working directory.

2) Create a typescript file named lab3_typescript_<username>_<labsection>. To do this, type at the command prompt: script lab3_typescript_<username>_<labsection>

2) Show that you are logged onto Zeus. Type: `uname -a`

3) Show a listing of the current directory to ensure the source file and Makefile are present.

4) Show a listing of your code.

5) Remove any versions of the lab3_<username>_<labsection> executable that may appear.

6) Compile the code using your Makefile.

7) Show that the executable file was created from the compile command.

8) Run the code using the provided sample script.

9) Delete the executable using the *make clean* command

10) End the typescript by typing Control-d

11) Verify your typescript file is correct, then change (cd) to the directory above your Lab3_<username>_<labsection> directory.

12) Create a tarfile of your Lab3 directory

13) Submit the tarfile for your lab to Blackboard as Lab 3.

Points to Review:

- Learn a bit more about Unix Redirection: A straightforward page that discusses I/O redirection can be found at: http://www.ee.surrey.ac.uk/Teaching/Unix/unix3.html
- Note the differences between the while loop and the do-while loop. Which one is guaranteed to execute the code within the loop at least once?
- Be aware that the switch statement needs break statements to avoid "falling through" to the next case.

Congratulations! You have completed Lab 3.