Lab 3: Released Wednesday, week 3 and due Tuesday, week 4

Note: This lab gives you all of the instructions for writing your code in Step 2, and then the instructions for compiling in Step 3. You are not required to write all three programs before compiling, feel free to write one, compile and test it, and then move on to the next.

Functions and Command-Line Arguments

Step 1: Getting Ready

Recall what environment you set up in lab 1. Navigate there, and then (same as last week), clone the Github repository. Go to the course Github (https://github.com/ucsb-cs16-1-u22) and make sure you can see a repository named Lab03-yourgithubusername. If you can see that repository, open it up in your browser and go to the green “Code” button near the upper right. Copy the SSH link for this repository (for more instructions on this step, and future steps involving Github, check out https://sites.cs.ucsb.edu/~maradowning/cs16/refs/gitbasics.html).

From here, run the following command in the folder you’d like your Lab03 folder to end up in:
`git clone <link you just copied>`

Check afterwards with `ls` to make sure the repository has been cloned properly, you should see a new folder named Lab03-yourgithubusername. You can run the following command to enter this directory and see the starter code files:
`cd Lab03-yourgithubusername`

Step 2: Edit your C++ Programs

This assignment consists of 2 problems, each of which is described below. Each is worth 50 points and should be solved in its own file. Both must be submitted for full assignment credit.

Important: Do not plagiarize.

Program 1: change.cpp
Write a program that tells what coins to give out for any amount of change from 1 cent to 99 cents. For example, if the amount is 86 cents, the output would be something like the following:

86 cents can be given in 3 quarters, 1 dime, 1 penny.

Use coin denominations of 25 cents (quarters), 10 cents (dimes), and 1 cent (pennies) only. Do not use nickel and half-dollar coins.

Your program should use the following function (it doesn't have to be the only function, but you have to use this one). Your program must use the `compute_coins` function declaration shown here:

```
void compute_coins(int amount);
// Precondition: 0 < amount < 100
// Postcondition: The function prints out the number of
// quarters, dimes, and pennies needed to make the
// amount value.
```

The program should verify that the value of `amount` is between 1 and 99 (inclusive). If this is not the case, the function should print out an error message (see example below).

A loop in the program should have the user repeat this computation for new input values until the user enters a zero to quit.

A session should look exactly like the following example (including whitespace and formatting) with all manner of different numbers for the input.

```
Enter number of cents (or zero to quit):
86
86 cents can be given in 3 quarters, 1 dime, 1 penny.
Enter number of cents (or zero to quit):
99
99 cents can be given in 3 quarters, 2 dimes, 4 pennies.
Enter number of cents (or zero to quit):
1
1 cent can be given in 1 penny.
Enter number of cents (or zero to quit):
101
Amount is out of bounds. Must be between 1 and 99.
Enter number of cents (or zero to quit):
0
```

Important: Note that the words and the punctuation (commas, especially) can be different for different kinds of input values. They have to match the numbers, so singulars for “quarter”, “dime”, and “penny”, and plurals for “quarters”, “dimes”, and “pennies”. The autograder will not tolerate bad grammar!

For this program, feel free to print error messages to `cerr` rather than `cout`, but it is not required. It will be required in the second program for this lab (calculate.cpp).

Hints: It will be helpful to use lots of if-else statements and you will have to include the <string> library to make variable messages to meet the singular/plural and comma/no-comma requirements.

Program 2: calculate.cpp
You will write a program that mimics a simple calculator that can do one of three operations on two integers: **addition**, **multiplication**, or **modulo**.

The program takes 3 arguments **at the command line**: an integer, a character, and another integer. All of these arguments must be separated by a space character. The middle character can only be 1 of 3 choices: +, x, or %. The program then returns either the **sum**, the **product**, or the **modulo** of the 2 integers, respectively.

The program should be able to verify that:

1. the user has provided exactly 3 command-line arguments
2. the operator used is one of the 3 allowed operators and nothing else, and
3. when using modulo, the second integer is not zero (otherwise you would divide by zero)

For each of these conditions, the program should print out a specific error message (called the "usage" message) and exit. For each condition listed above, the error messages should respectively be:

1. Number of arguments is incorrect.
2. Bad operation choice.
3. Cannot divide by zero.

You have to use `cerr` instead of `cout` to output the usage messages. Do not use `cerr` for any other outputs the program makes. While cerr and cout are alike in that they direct values to standard output, we usually use cout for the standard output, but use cerr to show or report *errors* to the user and the system.

You may use the exit(1) statement when exiting from giving some of the error messages. This statement is similar to return or break, but even stronger—it ends the entire program immediately, regardless of where you are.

Here is a skeleton program to help get you started.

```
#include <iostream>
#include <cstdlib>
using namespace std;

// Usage: ./calculate int char int
// char can either be: + x or %
int main(int argc, char* argv[]) {
      // PART 1: Check to see if the number of arguments is correct
      //    Hint: use "if (argc ...)" to check this,
      //    use cerr to output any messages

      // PART 2: Convert arguments into integers (only those that need it!)
      //    Hint: this means using atoi().

      // PART 3: Check for illegal operations like divide by zero. . .
      //    use cerr to output any messages
      //    You can assume the two integer arguments, if they exist,
      //    will be integers (so you don't need to error check here!)

      // PART 4: Do the appropriate calculation,
      //    outputs using cout

      return 0;
}
```

This program does **not** loop back to take inputs again. Your outputs should match the ones in the example run shown below. (The dollar sign indicates that this is a line you type into the terminal (without $) to run the program).

```
$ ./calculate 4
Number of arguments is incorrect.

$ ./calculate 1 + 2
3

$ ./calculate 7 x -5
-35

$ ./calculate 4 - 3
Bad operation choice.

$ ./calculate 14 % 0
Cannot divide by zero.
```

Hint: If the second argument (the operation) exists, you can assume it has only one character. To get the first character from an array of characters like the elements, you can use [0]: `argv[index][0]` gives the first character of the element of argv at `index`.

Step 3: Compile your Code

This time you'll use a Makefile to compile your code. Create a new file named `Makefile` (no extension) and copy the following code into it:

```
all: change calculate

change: change.cpp
	g++ -std=c++11 -Wall change.cpp -o change

calculate: calculate.cpp
	g++ -std=c++11 -Wall calculate.cpp -o calculate

clean:
	rm change calculate
```

Once you've saved that file, to compile an individual program, you can run either
`make change`
or
`make calculate`

Additionally, if you want to compile both programs at once, you just have to run:
`make`

If you need to delete your executable files, running
`make clean`
will delete both of them

You can run each of the executable files as normal:
`./change`
`./calculate`

If you encounter an error, use the compiler hints and examine the line in question. If the compiler message is not sufficient to identify the error, you can search online to see when the error occurs in general.

Remember to re-compile after you make any changes to your C++ source code.

Step 4: Submit your Code

There is a more detailed (and more general) set of instructions on the course webpage, under Quick References: https://sites.cs.ucsb.edu/~maradowning/cs16/refs/gitbasics.html. If you are new to git/github, I suggest reading through this guide as well when you submit your assignment.

Once you have finished your code (or any time you want to save your code to Github and take a break), first run `git status` to see which files you have modified/created. From there, add any files you want to end up on Github by running the following command:

```
git add <filename>
```

For this lab, most likely you will run two add commands:

```
git add change.cpp
git add calculate.cpp
```

Please do not add any executable files you created by compiling your code. However, if you do end up accidentally adding them, don't worry about it for now. In later labs I will take off a couple points for adding these files, but in these labs I'll just mention it to make sure you're aware.

Once you've added all the files you want, you will need to commit them. Run:

```
git commit -m "Your commit message here"
```

Please write a descriptive commit message within the quotes, something like "Finished Lab02" if you are done, or something more specific if you're pushing code partway through.

Finally, run:

```
git push
```

to push your changes to Github. Once you've run this command, make sure to go to your repository in your web browser and check that the updated files are there (you might have to reload).

Finally, once you're completely finished with the lab, fill out the Lab03 submission on Gradescope. It only asks you for a list of people you discussed the lab with (remember, discussion of the lab is ok but do not give classmates answers), the number of hours you spent, and has a checkbox to indicate that you're done. I will use this as an indication that your code is pushed to Github and I can begin grading.