Lab 03: Functions and Command-Line Arguments

Assigned: *Tuesday, October 20th, 2020*
Due: *Monday, October 26th, 2020*
Points: *100 (normalized to 100 in gradebook)*

- There is NO MAKEUP for missed assignments.
- We are strict about enforcing the LATE POLICY for all assignments (see syllabus).

Introduction

The assignment for this week will utilize concepts of void functions and command line arguments.

Step 1: Getting Ready

First, open a terminal window and log into the correct machine. Change into your CS 16 directory, create a **lab03** directory and change into it. Remember that at any time, you can check what directory you are currently in with the command **pwd**.

Step 2: Create and Edit Your C++ Files

This week, you will need to create **2 files called change.cpp, and calculate.cpp**: Each corresponds to one of the problems listed below, which make up this lab. Each is worth 50 points and should be solved in its own file and both must be submitted for full assignment credit.

IMPORTANT NOTE: This lab will be also graded for use of comments and style. See below for details. In addition, we will take *major* points OFF if you use C++ instructions/libraries/code that either (a) was not covered in class, or (b) was found to be copied from outside sources (or each other) without proper citation.

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 as 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 0 and 99 (inclusive). If this is not the case, the function should print out an error message (see example run below for the exact wording).

A loop in the program should have the user repeat this computation for new input values until the user enters a zero to quit (again, see example below).

A session should look exactly like the following example (including whitespace and formatting - note that there is no whitespace at the end of each of these lines and each printed line has a newline at the end), with all manners of different numbers for inputs and the output. The user input is **bolded**. Note also the case shown below where the input is out of bounds.

```
$ ./change
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):
35
35 cents can be given in 1 quarter, 1 dime.
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):
25
25 cents can be given in 1 quarter.
Enter number of cents (or zero to quit):
76
76 cents can be given in 3 quarters, 1 penny.
Enter number of cents (or zero to quit):
45
45 cents can be given in 1 quarter, 2 dimes.
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):
22
22 cents can be given in 2 dimes, 2 pennies.
Enter number of cents (or zero to quit):
0
$
```

Challenge: Note that the words AND the punctuation (commas especially) can be different in different cases. They have to match the numbers, so singulars for **quarter**, **dime**, and **penny** are expected as appropriate for the case and plurals for **quarter<u>s</u>**, **dime<u>s</u>**, and **penn<u>ies</u>**, etc…
This can be trickier than it looks!

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

CALCULATE.CPP

You will write a program that mimics a simple calculator that can do one of 3 operations on 2 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 be 1 of only 3: '**+**', '**x**', or '**%**'. The program then returns either the **sum**, the **product**, or the **modulo** of the 2 integers, respectively.

For example, if you ran the program as (by typing the following on the Linux command line): `./calculate 1 + 2`, then you should get `3` as the answer.

The program should be able to verify that:

1) the user has exactly 3 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.

Hint: 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. The instructor will demonstrate the proper use of **exit(1)** and **cerr** in class in Lecture 6 (pre-recorded video).

Your outputs should match the ones in the example run shown in the next page.

Here is a skeleton program to help you get started:

```cpp
// Calculate.cpp
// By: <Your Name Here>
// Created on: <Date Here>

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

// Usage: ./calculate int char int
// char can be one of 3 things: + 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

    // PART 4: Do the appropriate calculations,
    //         outputs using both cout and cerr, etc...

    return 0;
}
```

This program does not loop back to take inputs again. See the sample runs examples below.

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

$ ./calculate 1 + 2
$ 3

$ ./calculate 6 + -19
$ -13

$ ./calculate 7 x -5
$ -35

$ ./calculate 8 % 3
$ 2

$ ./calculate 4 3 2 + 0
Number of arguments is incorrect.

$ ./calculate 5 - 4
Bad operation choice.

$ ./calculate 8 % 0
Cannot divide by zero.
```

Step 3: Create a makefile and Compile the Codes with the make Command

In order to learn another way to manage our source codes and their compilations, we will first create a **makefile** and put in the usual g++ commands in it. Afterwards, whenever we want to compile our programs, the Linux command is a lot shorter – so this is a convenience. The use of **makefiles** will reveal itself to be very useful the more complex our programs and CS projects become. We will discuss **make** and **makefile** in class soon!

Using your text editor, create a new file called **makefile** and enter the following into it:

```
all: change calculate

change:
        g++ change.cpp -o change

calculate:
        g++ calculate.cpp -o calculate
```

Then from the Linux prompt, you can do one of two things: either issue separate compile commands for each project, like so:

```
$ make change
```

Or, you can issue one command that will compile all the projects mentioned in the **makefile**, like so:

```
$ make
```

If the compilation is successful, you will not see any output from the compiler. You can then run your programs, for example:

```
$ ./change
```

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 the relevant files after you make any changes to the C++ code.

Step 4: Submit using GRADESCOPE

BEFORE YOU SUBMIT YOUR FINAL VERSION FILES: Make sure that you have **COMMENTS** in your program that would help explain what your program is doing to another person reading your program code. Also make sure that you **STYLIZE** your program appropriately to additionally make it easier on others reading your code. These 2 aspects (having appropriate comments and style) will be graded for an *extra 20 points* beyond the automatic grade you get from Gradescope.

Log into your account on **https://www.gradescope.com/** and navigate to our course site. Select this assignment (Lab 3). Then click on the "Submit" button on the bottom right corner to make a submission. You will be given the option of uploading files from your local machine or submitting the code that is in a github repo. Choose the first option and follow the steps to **upload BOTH (2) .cpp FILES** (they must be named **change.cpp** and **calculate.cpp**) to Gradescope.

You may submit this lab multiple times. You should submit only after you test it on your computer (or CSIL) and are satisfied that the programs run correctly. The score of the last submission uploaded before the deadline will be used as your assignment grade.

Step 5: Done!

Once your submission receives a score of 100/100, you are done with this assignment. REMEMBER that there are 20 additional points that will be scored according to your proper use of comments and styling. The total will then be normalized to 100 points again (i.e. 120 score will be 100%, 111 score is 92.5%, etc…) to grade this lab.

WE WILL BE CHECKING FOR PLAGIARISM – DO NOT COPY FROM OTHER STUDENTS OR FROM SOURCES ONLINE! USE PROPER CITATION OF CODE THAT YOU DID NOT WRITE! THE CONSEQUENCES WILL - AT MINIMUM - BE A ZERO ON THIS LAB AND POSSIBLY A ZERO IN THIS COURSE AND YOU WILL BE REPORTED TO THE UNIVERSITY.