CSCI 1300
Assignment 1: Change Maker

Assigned: Tuesday Sep 9, 2003
Due: Tuesday Sep 16, 2003

Honor Code Reminder

Please adhere to the policy presented on the first day of class: All work you submit to be graded must be entirely your own. Breaking this rule will result in an F grade for the entire course. You may ask friends and family for help in understanding the syntax of the C++ language, for working through examples in the text, and for understanding compliation and run-time errors, but they may not write or help you write the code for any homework assignment. If you are stuck, see the professor, TAs, or undergraduate gurus.

The Assignment

The assignment is presented in three parts. Each part builds on the previous part. You are to complete part 3 and hand it in. However, if you do not manage to finish part 3, then get as far as you can, even if you only complete parts 1 or 2.

Part 1

Suppose you make a purchase for $2.54 and you pay with a $5 bill. You should then receive $2.46 in change. Write a program that figures out how many dollars, quarters, dimes, nickels, and pennies should be given back in change so as to minimize the number of coins. For example, one way to produce $2.46 in change is with 246 pennies, but that would be far more change than anyone would want to carry around. Sample output from your program might look like (user input is underlined):

```
Enter amount to be given in change: $2.46
The correct change would be:
     2 dollar(s)
     1 quarter(s)
     2 dime(s)
     0 nickel(s)
     1 pennies
```

The way to minimize the amount of change given is by making change in the larger denominations when possible. For example, if the amount to be produced in change is between $1 and $2, then a dollar bill should be given. If the dollar isn't given, making correct change would require using 4 quarts, 10 dimes, 20 nickels, or 100 pennies, and all of those possibilities result in more change than necessary.

Part 2

Modify your program from part 1 so that it has the following features:

- Do not print out lines for which the number of coins is zero (e.g., the nickel line in the example above).
- If the number of coins is exactly one, then print the singular form of the coin name; if the number of coins is two or more, print the plural form, e.g.,

```
The correct change would be:
     2 dollars
     1 quarter
     2 dimes
     1 penny
```

Part 3

Suppose that your change program is being used to dole out money from a register. The register has an initial pool of cash, e.g., 1 dollar, 8 quarters, 1 dime, 4 nickels, and 55 pennies. Change must be made from the available pool. For this pool, our example of $2.46 in change would produce:

```
The correct change would be:
     1 dollar
     5 quarters
     1 dime
     2 nickels
     1 penny
```

Rather than asking the user to enter the initial register contents when you run the program, you can define those values in your program as constants. You can assume that the register has an infinite supply of pennies, so that the correct change can always be produced.

Bells and Whistles

After you have done Parts 1-3, if you want to embellish your program further, you could (a) allow for multiple withdrawals from the register (this will require a looping construct which we haven't discussed yet in class; read ahead in the text), and (b) allow for the possibility—not considered in part 3—that the register has insufficient cash to supply the change, in which case the user should be told that the correct change cannot be made.

Advice

As you advance from one part of the assignment to the next, save a copy of your working program to that point. Then, if you can't complete the next part, at least you have working code to hand in. In general, it is a good idea to save your program at various points. The professor and TAs will not have much sympathy if you come to us and say, "I had it running a few days ago and then I changed something and it no longer works."

One important decision that you must make early on is how to represent a certain amount of money. One possibility is to use a floating point variable, which would allow you to encode $2.46 as the floating point value 2.46. Another possibility is to use an integer variable, which would require that you encoded $2.46 as the number of cents, or 246. The pitfall of using floating point is you will have to be concerned with rounding errors. The trick of using integer values is that you have to keep in mind that you are operating in units of one hundredth of a dollar, not dollars. Before you start writing code, you will have to pick one unit of measurement or the other (i.e., dollars or pennies). After you see the disadvantages of the way you have chosen, you may decide to switch to another unit of measurement.

Test your program thoroughly. Worry about every possible case. A program that works on only some inputs isn't much use. The *boundary conditions* are the cases that often break a program. For example, in this assignment, I'd worry about cases where exact change can be made with high denominations, and all lower denominations should be zero (e.g., making change for $2.00). Another boundary condition might be when most of the answers are zero, e.g., making change for $0.01 or $0.00. And for Part 3, a case to consider is when the bank has nothing but pennies.

Handing in Your Program

Follow these instructions of your program will not be graded.

(1) By 11:59 p.m. of the assignment due date, you must send email to your TA. See the course home page for the email address to use. (Note: This address may have changed over the past few weeks!) The subject of the message must be "Assignment 1". The message must contain no text in the body, but must include an attachment which is the *source code* (the .cxx file, not the .exe file) for your program. The program must be named with your 9 digit student ID number, e.g., `123456789.cxx`.

(2) The first line of your program should be a comment with the assignment #, your student ID and name, e.g.,

```
// Assignment 1 Student ID 123456789 Michael C. Mozer
```

(3) If you have any sort of note to your TA (e.g., you weren't able to get some part of the program to work), write this note in comments below the first line above.

(4) Late assignments will not be accepted.