CSC 101 Programming Assignment 1: Making Change

ISSUED:	Beginning of Week 1
DUE:	On or before 11:59:59PM Friday 6 April, via `handin` on unix1
POINTS POSSIBLE:	100
WEIGHT:	3% of total class grade
READING:	Textbook chapters 2 and 3

Overview

For this assignment, you are writing a simple C program that makes change, similar to how a teller makes change at a retail store. Writing the program requires that you understand the following concepts:

- the basic structure of a C program
- compiling and executing a C program
- computations with assignment statements and arithmetic expressions
- using simple functions in a C program
- integer input and output

Program Specification

The program inputs two integer values: the amount of a purchase and an amount tendered. Both inputs are in cents. The program outputs the total amount of change due, which is the difference between the amount tendered and the purchase amount. Following the total change value, the program outputs the change in five specific denominations of money: dollars, quarters, dimes, nickels, and pennies. The change in these denominations is computed to be the maximum whole number of units in each denomination that add up to the total change due.

The program may assume the following:

a. The inputs are correctly entered as positive integer values.
b. The amount tendered is greater than or equal to the purchase amount.

Here is a sample run of how the program behaves given a purchase amount of 216 (i.e., $2.16) and an amount tendered of 500 (i.e., $5.00):

```
Input the amount of the purchase, in cents: 216
Input the amount tendered, in cents: 500

Total change due = 284

Change in dollars through pennies is:
    2 dollars
    3 quarters
    0 dimes
    1 nickels
    4 pennies
```

Your program must prompt for input and write output in precisely the same format as this sample. Note that you do not need to worry about ungrammatical plurals in the output. E.g., the output "1 nickels" is correct for this program, even though the more correct English output would be "1 nickel".

Program Structure

You are required to implement and use the following functions in your program:

Function	Specification
`int get_dollars(int cents)`	Return the correct number of dollars in change for the given amount of cents. For example, if `cents` = 284, `get_dollars` returns 2.
`int get_quarters(int cents)`	Return the correct number of quarters in change for the given amount of cents. For example, if `cents` = 284, `get_quarters` returns 3.
`int get_dimes(int cents)`	Return the correct number of dimes in change for the given amount of cents. For example, if `cents` = 284, `get_dimes` returns 0.
`int get_nickels(int cents)`	Return the correct number of nickels in change for the given amount of cents. For example, if `cents` = 284, `get_nickels` returns 1.
`int get_pennies(int cents)`	Return the correct number of pennies in change for the given amount of cents. For example, if `cents` = 284, `get_pennies` returns 4.
`int main()`	Perform input, call the other functions to perform the computations, and output the results.

Each of the five "`get_`" functions need only be one line long. Functions in a C program are often longer than a single line, but we're starting out with some very simple functions, to get you introduced to them.

Program Development and Testing

Throughout the quarter, your instructor will stress the utility of *incremental development*. This means that you start by writing a simple version of the program and test it with some simple values. When that part works, you add some more computation and test some more. The idea is to build a working program in a step-by-step fashion, starting with the simple functionality, and incrementally adding harder functionality. We will discuss further details of incremental development and testing in the first weeks' lectures and labs.

When you have a working program, you need to test it with enough sample inputs to convince yourself that it works properly. In providing test inputs, you may rely on assumptions a and b above to be true. Therefore, you need not test your program with inputs that are negative values, or with inputs where the amount tendered is less than the purchase amount. In upcoming assignments, we will learn how to enhance programs to check for anomalous input conditions such as these, and to output appropriate error messages when the bad inputs are detected.

To help you with testing, there is a pre-compiled version of the program in the program 1 directory. The program is named "`make_change`". When you run it from the command line, it prompts for input and produces output in precisely the form your program must do.

There is a detailed test plan for Program 1, in the testing directory. This plan describe the detail of how your program will be run and how its execution will be scored.

Other Requirements

Your program is stored in a single file named `make_change.c`. You are free to develop and test your program on any computer(s) of your choice. Once it runs to your satisfaction, you must copy it to the CSL computer named "unix1", and compile and execute it there.

The program must compile error free using the following compilation command on unix1:

```
gcc -ansi -pedantic -Wall -Werror make_change.c -o make_change
```

The "`-o`" argument to `gcc` puts the compiled program in the file named "`make_change`", instead of the default file "`a.out`".

Once compiled, your program must run correctly on unix1. In all likelihood, your programs for CSC 101 will compile and execute the same on unix1 as they do on other computers. However, to avoid any subtle differences, we will always use unix1 as the single target computer on which your programs must compile and run successfully. If your program does not run successfully on unix1, it won't do you any good to say "*but it ran perfectly at home*". ***You must always compile and test on unix1 before you hand in.***

Grading Criteria

Your program will be graded in two phases:

1. program execution, based on the program 1 test plan
2. adherence to 101 programming standards, based on the programming-conventions handout

As described in the test plan, the execution is worth a total of 100 points, specifically 4 points for each of 25 test cases. As described in the programming conventions handout, points are deducted from your execution score for any convention violations. The scoring section conventions handout has details for how the deductions are computed.

Collaboration

NO collaboration is allowed on this assignment. Everyone must do their own individual work.

Program Turnin Procedure

For this and all future programming assignments, you will use the UNIX `handin` command on unix1. You run `handin` from the unix1 command line as follows:

```
handin gfisher 101_prog1 make_change.c
```

Run the `handin` command from the unix1 directory where your copy of `make_change.c` is stored. Note that you must only run `handin` from unix1, not the lab computers. This means that you must first login to unix1 before you run handin.

You can resubmit your file as many times as you like, up to the submission deadline. Each new submission completely replaces the previously submitted file(s). If you follow an incremental development strategy, you can submit as many partially-working versions as you like, as each step is completed.