Project 1, Program Design

Write a C program that asks the user to enter a U.S. dollar amount and then shows how to pay that amount using the smallest number of $20, $10, $5, and $1 bills.

Hint: Divide the amount by 20 to determine the number of $20 bills needed, and then reduce the amount by the total value of the $20 bills. Repeat for the other bill sizes. Be sure to use integer values throughout, no floating-point numbers.

If the amount entered is less than zero or greater than one billion (1000000000), output an error message and abort the program.

Before you submit:

1. Compile with –Wall. –Wall shows the warnings by the compiler. Be sure it compiles on ***circe*** with no errors and no warnings.

   gcc –Wall dollar.c

2. Be sure your Unix source file is read & write protected. Change Unix file permission on Unix:

   chmod 600 dollar.c

3. Test your program with the shell script try_dollar on Unix:

   chmod +x try_dollar
   ./try_dollar

4. Submit dollar.c on Canvas.

Grading:

Total points: 100

1. A program that does not compile will result in a zero.
2. Runtime error and compilation warning 5%
3. Commenting and style 15%
4. Functionality 80%

Programming Style Guidelines

The major purpose of programming style guidelines is to make programs easy to read and understand. Good programming style helps make it possible for a person knowledgeable in the application area to quickly read a program and understand how it works.

1. Your program should begin with a comment that briefly summarizes what it does. This comment should also include your **name**.
2. In most cases, a function should have a brief comment above its definition describing what it does. Other than that, comments should be written only *needed* in order for a reader to understand what is happening.
3. Variable names and function names should be sufficiently descriptive that a knowledgeable reader can easily understand what the variable means and what the function does. If this is not possible, comments should be added to make the meaning clear.
4. Use consistent indentation to emphasize block structure.
5. Full line comments inside function bodies should conform to the indentation of the code where they appear.
6. Macro definitions (#define) should be used for defining symbolic names for numeric constants. For example: `#define PI 3.141592`
7. Use names of moderate length for variables. Most names should be between 2 and 12 letters long.
8. Use either underscores or capitalization for compound names for variable: `tot_vol`, `total_volumn`, or `totalVolumn`.