Project 4, Program Design

1. Modify `cents.c` (attached) that asks the user to enter a number for cents and then calculates the number of quarters, dimes, nickels, and pennies needed to add up to that amount

   So it includes the following function:

   ```
   void coins(int cents, int *quarters, int *dimes, int *nickels,
   int *pennies);
   ```

   The function determines the smallest number of quarters, dimes, nickels, and pennies necessary to add up to the amount of cents represented by the `cents` parameter. The `quarters` parameter points to a variable in which the function will store the number of quarters required. The `dimes`, `nickels`, and `pennies` parameters are similar. Modify the `main` function so it calls `coins` to compute the number of quarters, dimes, nickels, and pennies. The `main` function should contain the `printf` statements that display the result.

Important: this program will be graded based on whether the required functionality were implemented correctly instead of whether it produces the correct output, for the functionality part (80% of the total grade)

2. Complete the functions for the attached program *`complex.c`*. The program adds, subtracts, and multiplies complex numbers. The program will prompt the user for the choice of operations, read in the complex numbers, perform the operations, and print the results. The main program repeatedly prints the menu, reads in the user's selection, and performs the operation.

We use pointers only for those arguments that the function intends to change. In all the functions, `r` represents the real component and `i` represents the imaginary component. Study the program and complete the following functions. **Do not modify the function prototypes.**

1) Complete the function that reads in two complex numbers.

```
void read_nums(double *r1, double *i1, double *r2, double
*i2);
```

The function should prompt for the user to enter the real component and imaginary component of the first number and the second number and store the values in the variables pointed by `r1`, `i1`, `r2`, `i2`, respectively.

2) Complete the following functions that calculate the addition, subtraction, and multiplication of two complex numbers. The functions take real component and imaginary component of the first number and the second number and store the values in the variables pointed by `r3` and `i3`, respectively.

```
     void add(double r1, double i1, double r2, double i2, double
*r3, double *i3);

     void subtract(double r1, double i1, double r2, double i2,
double *r3, double *i3);

     void multiply(double r1, double i1, double r2, double i2,
double *r3, double *i3);
```

Before you submit

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

   gcc –Wall cents.c

   gcc –Wall complex.c

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

   chmod 600 cents.c

   chmod 600 complex.c

3. Test complex number program with the shell script *try_cents* and *try_complex* on Unix:

   chmod +x try_cents

   ./try_cents

   chmod +x try_complex

   ./try_complex

4. Submit *cents.c* and *complex.c* on Canvas.

Grading

Total points: 100 (problem 1: 40 point, problem 2: 60 point)

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. Information to include in the comment for a function: name of the function, purpose of the function, meaning of each parameter, description of return value (if any), description of side effects (if any, such as modifying external variables)
4. 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.
5. Use consistent indentation to emphasize block structure.
6. Full line comments inside function bodies should conform to the indentation of the code where they appear.
7. Macro definitions (#define) should be used for defining symbolic names for numeric constants. For example: `#define PI 3.141592`
8. Use names of moderate length for variables. Most names should be between 2 and 12 letters long.
9. Use underscores to make compound names easier to read: `tot_vol` or `total_volumn` is clearer than totalvolumn.