OOP Practice Task

In the heyday of the British empire, Great Britain used a monetary system based on pounds, shillings, and pence. There were 20 shillings to a pound, and 12 pence to a shilling. The notation for this old system used the pound sign, £, and two decimal points, so that, for example, £5.2.8 meant 5 pounds, 2 shillings, and 8 pence. (*Pence* is the plural of *penny*.) The new monetary system, introduced in the 1950s, consists of only pounds and pence, with 100 pence to a pound (like U.S. dollars and cents). We'll call this new system *decimal pounds*. Thus £5.2.8 in the old notation is £5.13 in decimal pounds (actually £5.1333333). Write a program to convert the old pounds-shillings-pence format to decimal pounds. An example of the user's interaction with the program would be

```
Enter pounds: 7
Enter shillings: 17
Enter pence: 9
Decimal pounds = £7.89
```

In most compilers you can use the decimal number 156 (hex character constant `'\x9c'`) to represent the pound sign (£). In some compilers, you can put the pound sign into your program directly by pasting it from the Windows Character Map accessory.

Remember the `sterling` structure? We saw it in Exercise 10 in Chapter 2, "C++ Programming Basics," and in Exercise 11 in Chapter 5, among other places. Turn it into a class, with pounds (type `long`), shillings (type `int`), and pence (type `int`) data items. Create the following member functions:

- no-argument constructor
- one-argument constructor, taking type `double` (for converting from decimal pounds)
- three-argument constructor, taking pounds, shillings, and pence
- `getSterling()` to get an amount in pounds, shillings, and pence from the user, format £9.19.11
- `putSterling()` to display an amount in pounds, shillings, and pence, format £9.19.11
- addition (`sterling + sterling`) using overloaded `+` operator
- subtraction (`sterling - sterling`) using overloaded `-` operator
- multiplication (`sterling * double`) using overloaded `*` operator
- division (`sterling / sterling`) using overloaded `/` operator
- division (`sterling / double`) using overloaded `/` operator
- operator `double` (to convert to `double`)

To perform arithmetic, you could (for example) add each object's data separately: Add the pence, carry, add the shillings, carry, and so on. However, it's easier to use the conversion operator to convert both `sterling` objects to type `double`, perform the arithmetic on the doubles, and convert back to `sterling`. Thus the overloaded `+` operator looks like this:

```
sterling sterling::operator + (sterling s2)
   {
   return sterling( double(sterling(pounds, shillings, pence))
                    + double(s2) );
   }
```

This creates two temporary `double` variables, one derived from the object of which the function is a member, and one derived from the argument `s2`. These `double` variables are then added, and the result is converted back to `sterling` and returned.

Notice that we use a different philosophy with the `sterling` class than with the `bMoney` class. With sterling we use conversion operators, thus giving up the ability to catch illegal math operations but gaining simplicity in writing the overloaded math operators.