Part III: Polishing the Output

This assignment continues the bank account application from Bank Part 2. In this part, we will polish up the appearance of the monthly statement. To do this we will have to play with output formatting.

Output formatting is not a strong part of C++ and often requires a combination of several strategies. It was easier in C and is again easier in Java. But in this exercise, we will do it in C++.

In Part 2, our output looked like the following:

```
1  500  deposit
3  -50  ATM on Elm Street
7  -67.43  Sprint Wireless
17  -8.99  Burger King
373.58
```

In Part 3, we want the output to look like the following:

```
day   amount  description
        0.00  opening balance
  1   500.00  deposit
  3   -50.00  ATM on Elm Street
  7   -67.43  Sprint Wireless
 17    -8.99  Burger King
      373.58  closing balance
```

To make these changes, we must plan the steps. What should we do, and in what order? From looking at the difference between the current (as-is) and the new (to-be) we can identify the necessary changes (gap). The differences are an opening balance, column labels, and evenly aligned output for the transactions. Since the statement is now a document associated with the Account, let us start by moving the code to produce the above examples into the Account class as a displayStatement() method. The opening balance won't change anything else, so it is a logical next step. Then we can change the transaction display with uniform columns. Lastly, when we know the actual column widths, add the line with the column labels.

Step 11

Create a new method in the Account class called displayStatement, and use it to replace the two lines currently being used in main for displaying the transactions and the current balance. For starters, just move the two lines into the new method, without the object name. In the main function, use account1.displayStatement(). Account.cpp needs to include <iostream> to make this work. The output should look the same as it did before. Finish this step by adding the text " closing balance" to appear after the balance when it is printed at the end.

Step 12

Add an openingBalance property to the Account class. The opening balance needs to be set to the same value as the balance at the beginning. We could change the constructor to accept an opening balance. For now, it is sufficient to leave the constructor as it is, setting balance to 0.0. Either way, once the balance has been set in the constructor, add another line to set the openingBalance = balance.

Add a line to the displayStatement() method to show the opening balance. Include the text " opening balance" afterwards. At this point, your output should look like the following:

```
0  opening balance
1  500  deposit
3  -50  ATM on Elm Street
7  -67.43  Sprint Wireless
17  -8.99  Burger King
373.58  closing balance
```

Step 13

Instead of 0, we would like the output to show 0.00 for the opening balance. To fix that we must address the first formatting problem. There are a number of formatting options for use with cout. To make cout include the decimal part, add the keyword fixed in the cout statement before the amount to be displayed.

```
cout << fixed << openingBalance << " opening balance" << endl;
```

Using the "fixed" qualifier displays the decimal places. But for currency we only want two decimal places. For the number of decimal places, we need to put another statement before the above line:

```
cout.precision(2);
```

To be safe, use the fixed qualifier in the line that outputs the closing balance, as well. Then do the same, put the cout.precision(2) statement and use fixed for the amounts, in the display method of the Transaction class. The output should appear as below. The columns in the output look aligned, but as we shall see that's just a coincidence.

```
0.00  opening balance
1  500.00  deposit
3  -50.00  ATM on Elm Street
7  -67.43  Sprint Wireless
17  -8.99  Burger King
373.58  closing balance
```

Step 14

In the output shown above, the day numbers are left justified. We want them to be right justified. There is another feature of cout that we can use for making all integers use the same number of characters. This feature is a function called setw() that takes an integer for the number of spaces, and is used in the stream the way we used the keyword "fixed." To use setw(), we must include <iomanip> right after we included <iostream>.

```
    cout << setw(2) << dayOfMonth;
```

The output should now appear as follows:

```
0.00 opening balance
 1  500.00  deposit
 3  -50.00  ATM on Elm Street
 7  -67.43  Sprint Wireless
17  -8.99  Burger King
373.58  closing balance
```

Step 15

We can also use setw() to align the second column. The largest number, 500.00 uses 6 characters. There are also 2 characters between the day and the largest amount. Combine the leading spaces with the size needed for the largest amount, and add one more character to be safe, for a total of 9 spaces. Precede the amount output with setw(9), before “fixed” in the stream. Do the same for the opening and closing amounts. To line them up with the other amounts, add two more to match the 2 spaces used for the days. The amount should now look like:

```
       0.00  opening balance
 1   500.00  deposit
 3   -50.00  ATM on Elm Street
 7   -67.43  Sprint Wireless
17    -8.99  Burger King
     373.58  closing balance
```

Step 16

The final step is to create a line with column labels. For the column labels, add another method to the Transaction class, called displayLabels(). Note that the output of displayLabels will be the same regardless of the value of any Transaction properties. Hence we can make it a class method instead of an instance method, using the static keyword: static void displayLabels(); Look up static methods in the Gaddis textbook. To call a static method, use the class prefix followed by :: instead of an object.

Because the word “day” has three letters, increase the size of the first column from 2 to 3. Make any other adjustments needed to again align everything with the columns. Call the Transaction displayLabels method as the first line in the Account::printStatement method.

Now the output should look as follows:

```
day   amount  description
        0.00  opening balance
  1   500.00  deposit
  3   -50.00  ATM on Elm Street
  7   -67.43  Sprint Wireless
 17    -8.99  Burger King
      373.58  closing balance
```

You may leave the other parts of main as they were, in which case, the output will include the lines with 1 0 1 1 1 and the holder name and address.

Conclusion

This exercise was intended to show you how to design a solution by dividing the problem into separate concerns, by forming separate classes and by solving one small problem at a time. When we started, the idea of creating a bank account application with transactions and a monthly statement would have seemed hopelessly complicated. But we did it with much less complication, once concern at a time.

The next time you have a programming problem, do not try to solve the problem all at once. Remember to think about dividing the problem into smaller concerns. Imagine an architecture of objects, using classes where each class takes care of its own concerns.

What to turn in:

There are still 7 .cpp and .h files to turn in. You may also want to make a copy of the output and include that in the zip file when you turn this assignment in.