Event-controlled Iteration, Decisions, Functions, Numerical Accuracy

This programming assignment uses many of the ideas presented in sections 3 through 7 of the course notes, so you are advised to read those carefully. Read and follow the following program specification carefully. Again, this program is somewhat more complex than the previous project, so don't underestimate it.

You may receive two scores on this project. You will receive a score for runtime testing from the NAG. You may also receive a software engineering score for following the instructions in the Programming Standards section below.

The Program Specification: Installment Loan Payments

You are planning to borrow a certain amount of money, called the principal. For an example, let's say the principal is $10,000.00. You will repay this loan by making monthly payments until the remaining balance is zero.

Of course, it's never that simple. Your lender will charge you a certain amount of interest, specified as an annual interest rate. The monthly interest rate is found by dividing the annual interest rate by 12 (naturally). Let's say the annual interest rate is 15%, then the monthly interest rate would be 1.25%.

Call the amount of money you still owe the lender the balance. At the end of each month, the monthly interest rate is applied to the balance to calculate the finance charge for that month, which is added to the balance. (Yes, it would be preferable if your payment were deducted first, but you don't make the rules here.) Continuing our example, at the end of the first month the finance charge would be 1.25% of $10,000, which works out to be $125.00; that makes the balance you owe $10,125.00.

Now, about your monthly payment. You will plan to make a certain minimum payment each month, based on the personal budget you're planning. Let's say you plan to pay at least $200 each month. The lender will also specify a minimum amount due each month; this is usually based on a flat percentage of the current loan balance. Let's say that for your lender, the minimum amount due will be 3% of the loan balance. Then in our example, the minimum payment due at the end of the first month would be 3% of $10,125.00, which works out to be $303.75. When the minimum amount due is larger than your planned minimum payment, you will pay the minimum amount due; otherwise, you will make your planned minimum payment. So, in our example, you would make a payment of $303.75.

There is one special case, that of the last payment. Usually, when the time comes to make the last payment, the balance on your loan will be smaller than your planned minimum payment. Of course, in that case, you'll just pay the amount of the loan balance, reducing the balance to $0.00.

Input file description and sample:

Your program **must** read its input from a file named `loan.data` — use of another input file name will generally result in a score of zero. The input file will consist of five lines of data, each preceded by an appropriate label as shown in the example that follows. The data lines, in order, will specify:

- The initial balance of the loan, a positive real (decimal) number.
- The annual interest rate, a positive real number.
- The percentage used to determine the minimum amount due, a positive real number.
- The amount of your planned minimum payment, a positive real number.
- The step size you will use when printing your output table (explained in the output section), a positive integer.

Each input line will be terminated by a newline character. Note that each label ends with a colon ':', followed by whitespace, and then the specified value; that implies a simple way to get rid of the labels.

You may assume that all the input values will be logically correct (no negative or missing values, for instance), and that they will be given in the specified order. Here is an example:

```
Principal:            10000.00
Annual rate:              0.18
Minimum payment rate:     0.030
Planned payment:        150.00
Step size:                12
```

What to Calculate:

Your program must calculate a table showing all the payments you must make in order to pay off this loan. The table format is shown in the output section below. In particular, for each month, you must calculate the finance charge, the minimum payment due, the actual payment you will make, and the resulting balance.

Your program must detect when the last payment is made, and then print the total finance charge and the total amount paid, as shown below. All dollar amounts must be printed with precision two.

Output description and sample:

Your program must write its output data to a file named `payments.out` — use of any other output file name **will** result in a score of zero. A sample output file produced from the sample input file above is shown below:

```
Programmer:  Bill McQuain
Loan Payment Calculation

Initial Balance:  10000.00
Annual Rate:          0.18

Month   Finance   Minimum   Actual    New
        Charge    Due       Payment   Balance
-------------------------------------------
    1    150.00    304.50    304.50   9845.50
    2    147.68    299.80    299.80   9693.38
    3    145.40    295.16    295.16   9543.62
    4    143.15    290.60    290.60   9396.17
    5    140.94    286.11    286.11   9251.00
    6    138.76    281.69    281.69   9108.07

   12    126.39    256.57    256.57   8295.70
   24    104.85    212.84    212.84   6881.89
   36     86.98    176.57    176.57   5709.02
   48     72.14    146.44    150.00   4731.28
   60     56.91    115.52    150.00   3700.62
   72     38.69     78.55    150.00   2468.34
   84     16.92     34.35    150.00    995.01

   92      0.09      0.18      5.96      0.00
-------------------------------------------
Total   6835.34            16835.34
```

As usual, the first line of your output should identify you by name, as shown. The second line should include the title "Loan Payment Calculation" only. The third line should be blank. The fourth and fifth lines should specify the initial

loan balance and the annual interest rate, labeled precisely as shown above. The sixth line should be blank, followed by the specified column labels and a row of delimiters to mark the top of the table.

Next your output file will contain a table, with lines of output indicating the result of selected monthly payments. You will print the payment information for the first six months (every loan will require at least that many payments), and then print the payment information for every k-th month, where k is the step size given in the input file. In the example above, k = 12, so you print every 12^{th} payment. Note that you count steps starting from the first month, not the sixth.

In all likelihood, the final loan payment will not be "hit" by the step size. You must print the information for the last payment (and only print it once).

Just for fun, you will print a blank line after the sixth payment line, and again before the last payment line. There should be no other blank lines in your table.

After the table, your output file will display the total of all the finance charges and the total of all the monthly payments. Of course, if everything is right, the initial balance plus the total finance charge should equal the total amount paid.

You are not required to use the exact horizontal spacing shown in the example above, but your output must satisfy the following requirements:

- You must use the specified header and column labels, and print a row of delimiters before and after the table body, as shown.
- You must arrange your output in neatly aligned columns, with a label identifying the contents of each column. Use spaces, not tabs to align your output. Note that while the NAG doesn't deduct points for horizontal alignment, the TA who may grade your source code for programming standards may do so.
- You must use the same ordering of the columns as shown here, and print all dollar amounts with precision two.
- You must print a newline at the end of each line.

Programming Standards:

You'll be expected to observe good programming/documentation standards. All the discussions in class about formatting, structure, and commenting your code will be enforced. A copy of *Elements of Programming Style* is included with the course notes — if you don't have a copy I strongly suggest you read the on-line edition (available from the course web page). Some specific requirements:

Documentation:

- You must include header comments specifying the compiler and operating system used and the date completed.
- Your header comment must describe what your program does; don't just plagiarize language from this spec.
- You must include a comment explaining the purpose of every variable or named constant you use in your program.
- You must use meaningful identifier names that suggest the meaning or purpose of the constant, variable, function, etc.
- Precede every major block of your code with a comment explaining its purpose.
- Precede every function you write with a header comment, explaining in one sentence what the function does, and listing each parameter to the function with an explanation of how that parameter is used by the function. The return value, if any, should also be described.
- You must use indentation and blank lines to make control structures like loops and if-else statements more readable.

Coding:

- Use named constants instead of variables where appropriate.
- Use at one `bool` variable appropriately in this program.
- Use `double` variables for all monetary amounts.
- Write and use at least four user-defined functions, not counting `main( )` or the function `RoundToCents( )` given later in this specification. At least one of these must `return` a `double` value, and at least one must be `void`.
- Provide a valid C++ prototype for each function you write, excluding `main( )` of course. The function `main()` must be the first function defined in your source code file.
- Pass parameters to your functions in appropriate ways. That is, pass a parameter by reference only if there is a logical reason to do so; otherwise, pass it by value or constant reference.
- You must not use any globally-scoped variables. It is OK to use globally-scoped function prototypes.

Your submission that receives the highest score may be graded for adherence to these requirements, whether it is your last submission or not. If two or more of your submissions are tied for highest, the earliest of those will be graded. Therefore: implement and comment your C++ source code with these requirements in mind from the beginning rather than planning to clean up and add comments later. No exceptions will be made to this policy.

Incremental Development:

You'll find that it's easier and faster to produce a working program by practicing incremental development. In other words, don't try to solve the entire problem at once. First, develop your design. When the time comes to implement your design, do it piece by piece. Here's a suggested implementation strategy for this project:

- First, write the code necessary to read the entire input file, and write out what you're reading (and nothing else) to the output file.
- Second, add the code to compute the monthly finance charge, and calculate an output table assuming you make your planned minimum payment each month. This won't give the correct results shown above, but you can hand-check your first five or six lines to be sure you're computing the finance charge and balance correctly. At this point, just print out all the monthly payments.
- Third, add the code to compute the minimum amount due each month, and print a table showing those values.
- Fourth, add the code to select the correct monthly payment, between the minimum due and your planned payment. At this point, you should have the output table substantially correct. The only remaining issue with the table is to be sure you're handling the very last payment amount correctly.
- Fifth, straighten out the last payment if you don't have it right already.
- Sixth, add running totals for the finance charge and the total amount paid, and print those values.

Now, at this point your dollar amounts should be close to, but probably not equal to, the ones shown in the output sample above.

Why Aren't the Monetary Amounts Exactly Right?

That is due to numerical roundoff issues. We'll discuss this in class, but the simple explanation is that your computer will store too many decimal places for your dollar amounts.

For example, the in the second line of the output table above, the finance charge would actually work out to be

$$0.015 * 9845.50 = 147.6825$$

Of course, the lender wouldn't carry the finance charge out to that many decimal places. In our case, we'll assume that the lender will round off each monetary amount to the nearest cent (and .005 will round up to .01). So the finance charge will be 147.68 in this case. Similarly, when the minimum amount due is calculated (using the rounded finance charge) we get

$$0.03 * (9845.50 + 147.68) = 299.7954$$

which the lender would round to 299.80 as shown. The question now is: how can you make your program round to the nearest cent? One approach that works well enough for our purposes is described below.

Consider the following code fragment:

```
double x = 147.6825;

x = int(100.0 * (x + .005) / 100.0;
```

Assuming `x` is stored exactly in hardware, the multiplication should yield 14768.75. The `int( )` is an example of a C++ typecast operator —this one truncates the value within parentheses to an `int`, so it should yield 14768. Dividing that value by the `double` constant 100.0 should then yield the `double` value 147.68, which is what we want.

This won't always produce exactly correct results (because the value of `x` won't always be stored exactly in hardware), but it should be sufficiently accurate for our purpose in this program. To make the use of this idea more modular, we should wrap it up in a function, something like:

```
////////////////////////////////////////////////////////////////////////
// RoundToCents( ) takes a double value and returns its value, rounded to
// two decimal places (more or less).
//
// Parameter
//   Amount    the value to be rounded
//
// Return value:  the value of Amount, rounded to two digits after the decimal
//
double RoundToCents(double Amount) {

   return (int(100.0 * (Amount + .005)))/100.0;  // round to nearest cent
}
```

Now, the recalculation of `x` above could be accomplished by the statement: `x = RoundToCents(x);`

The results in the given output file were obtained by using this function to round each computed monetary amount before using it in another calculation or printing it.

Testing:

Obviously, you should be certain that your program produces the output given above when you use the given input file. However, verifying that your program produces correct results on a single test case does not constitute a satisfactory testing regimen.

At minimum, you should test your program on **all** the posted input/output examples given along with this specification. The same program that will be used to test your solution generated those input/output examples. You could make up and try additional input files as well; of course, you'll have to determine by hand what the correct output would be.

Pledge:

Each of your submissions to the NAG must be pledged to conform to the Honor Code requirements for this course. Specifically, you **must** include the following pledge statement in the header comment for your program:

```
//    On my honor:
//
//    - I have not discussed the C++ language code in my program with
//      anyone other than my instructor or the teaching assistants
//      assigned to this course.
//
//    - I have not used C++ language code obtained fr om another student,
//      or any other unauthorized source, either modified or unmodified.
//
//    - If any C++ language code or documentation used in my program
//      was obtained from another source, such as a text book or course
//      notes, that has been clearly noted with a proper citation in
//      the comments of my program.
//
//    - I have not designed this program in such a way as to defeat or
//      interfere with the normal operation of the Automated Grader.
```

Failure to include this pledge in a submission is a violation of the Honor Code.