Fall 2018 CS 31

Programming Assignment 2
Rental Reckoning

Time due: 11:00 PM Thursday, October 18

Before you ask a question about this specification, see if it has already been addressed by the Project 2 FAQ. And read the FAQ before you turn in this project, to be sure you didn't misinterpret anything.

(Be sure you also do the homework accompanying this project.)

Your job at Yentl's Rentals is to write a program to determine how much to charge a car rental customer who has rented a car for a certain number of days and driven a certain number of miles. Your program must accept as input the odometer readings at the start and end of the rental, the number of days the car was rented, the customer's name, whether or not the car is a luxury car, and the month the rental started.

Here is an example of a dialog with the program (user input is shown here in **boldface**):

```
Odometer at start: 2417
Odometer at end: 2754
Rental days: 4
Customer name: Elon Musk
Luxury car? (y/n): n
Month (1=Jan, 2=Feb, etc.): 10
---
The rental charge for Elon Musk is $208.77
```

An odometer reading is an integer indicating the number of miles the car has been driven since being built. In the example above, Elon Musk drove 337 miles. The number of rental days is an integer.

Here is the Yentl's Rentals schedule of charges:

- The base charge for a rental is $33 per day for a non-luxury car, or $61 per day for a luxury car. In addition to the base charge, there is a charge per mile driven.
- The first 100 miles driven cost $0.27 per mile.
- In addition, for the next 300 miles driven, the charge is $0.27 per mile for a rental starting in a winter month (December through March), or $0.21 per mile for rentals starting in other months.
- In addition, for every mile beyond 400 miles, the charge is $0.19 per mile.

As an example, Elon Musk above would be charged a base charge of $132 (4 days at $33 per day for a non-luxury car) plus $27 for the first 100 miles driven, plus $49.77 for the remaining 237 miles driven (starting in October), for a total of $208.77.

Here's another example:

```
Odometer at start: 1885
Odometer at end: 1973
Rental days: 1
Customer name: John DeLorean
Luxury car? (y/n): y
Month (1=Jan, 2=Feb, etc.): 2
---
The rental charge for John DeLorean is $84.76
```

You can test your understanding of the schedule of charges by experimenting with this rental charge calculator we found at the Yentl's Rentals web site.

Your program must collect the information for one customer in the manner indicated by the examples, and then write to `cout` a line with three hyphens only (no spaces or other characters), followed by exactly one line in a format required below. Our grading tool will judge the correctness of your program by examining only the line following the line with three hyphens (and verifying that there are no additional lines). That one line you write must be in one of the following seven forms; the text must be **identical** to what is shown, except that *italicized* items are replaced as appropriate:

- If the starting odometer reading is negative:
  `The starting odometer reading must be nonnegative.`
- If the ending odometer reading is less than the starting reading:
  `The final odometer reading must be at least as large as the starting reading.`
- If the number of rental days is not positive:
  `The number of rental days must be positive.`
- If an empty string was provided for the customer name:
  `You must enter a customer name.`
- If the luxury status is not y or n (must be lower case):
  `You must enter y or n.`
- If the month number is not an integer between 1 and 12 inclusive:
  `The month number must be in the range 1 through 12.`
- If the input is valid and none of the preceding situations holds:
  `The rental charge for` *`customer`* `is $`*`amount`*

In the last case, *customer* must be the customer name the user entered, and *amount* must be the correct answer, shown as a nonnegative number with at least one digit to the left and exactly two digits to the right of the decimal point. (See pp. 32-33 of the Savitch book.) The lines you write must not start with any spaces. If you are not a good speller or typist, or if English is not your first language, be especially careful about duplicating the messages **exactly**. Here are some foolish mistakes that may cause you to get no points for correctness on this project, no matter how much time you put into it:

- Not writing to `cout` a line with exactly three hyphens in *all* cases.
- Writing any spaces on the line that is supposed to have three hyphens.
- Writing more than one line after the line with three hyphens. Don't, for example, add a gratuitous "Thank you for being gentle with your Yentl rental!".
- Writing lines to `cerr` instead of `cout`.
- Writing lines like these:

```
You must enter a customer name                    missing period
```

```
The rentel charge for Henrik Fisker is $123.40       misspelling
The rental Charge for Henrik Fisker is $123.40       wrong capitalization
The rental charge for Henrik Fisker will be $123.40  wrong text
The rental charge for Henrik Fisker is $ 123.40      extra space
The rental charge for Henrik Fisker is 123.40        missing dollar sign
The rental charge for Henrik Fisker is $123.40.      extra period
The rental charge for Henrik Fisker is $123.400      extra digit
The rental charge for Henrik Fisker is $123          missing decimal point and digits
```

Your program must gather the starting and ending odometer readings, the number of rental days, the customer name, the luxury status, and the rental month in in that order. However, if you detect an error in an item, you do not have to request or get the remaining items if you don't want to; just be sure you write to `cout` the line of three hyphens, the required message and nothing more after that. If instead you choose to gather all input first before checking for errors, and you detect more than one error, then after writing the line of three hyphens, write only the error message for the earliest erroneous item.

You will not write any loops in this program. This means that each time you run the program, it handles only one customer. It also means that in the case of bad input, you must not keep prompting the user until you get something acceptable; our grading tool will not recognize that you're doing that.

A string with no characters in it is the empty string. A string with at least one character in it is not the empty string, even if the only characters in it are things like spaces or tabs. Although realistically it would be silly to have a customer name consisting of seventeen spaces and nothing more, treat that as you would any other non-empty string: as a valid customer name. (Since you don't yet know how to check for that kind of situation anyway, we're not requiring you to.)

The one kind of input error that your program does **not** have to deal with, because you don't yet know how to do so, is not finding an integer in the input where an integer is expected. We promise that our grading tool will not, for example, supply the text `a whole lot` or `3256.5` when your program requests an odometer reading.

The correctness of your program must not depend on undefined program behavior. Your program could not, for example, assume anything about `n`'s value at the point indicated:

```
int main()
{
    int n;
    int m = 42 * n;  // n's value is undefined
    …
```

What you will turn in for this assignment is a zip file containing these three files and nothing more:

1. A text file named **rental.cpp** that contains the source code for your C++ program. Your source code should have helpful comments that tell the purpose of the major program segments and explain any tricky code.
2. A file named **report.docx** or **report.doc** (in Microsoft Word format) or **report.txt** (an ordinary text file) that contains:
   a. A brief description of notable obstacles you overcame. (In Project 1, for example, some people had the problem of figuring out how to work with more than one version of a program in Visual C++.)
   b. A list of the test data that could be used to thoroughly test your program, along with the reason for each test. You don't have to include the results of the tests, but you must note which test cases your program does not handle in the way this spec requires. (This could happen if you didn't have time to write a complete solution, or if you ran out of time while still debugging a supposedly complete solution.) For Project 1, for example, such a list, had it been required, might have started off like this:

      More voters than the total for Newsom and Cox (1000, 413, 382)
      Fewer voters than the total for Newsom and Cox (500, 413, 382
      Zero total voters (0, 100, 100)
      Data giving a non-integer percentage (1000, 413, 382)
      More votes for Newsom than Cox (1000, 413, 382)
      Equal votes for Newsom and Cox (1000, 500, 500)
      …

3. A file named **hw.docx** or **hw.doc** (in Microsoft Word format) or **hw.txt** (an ordinary text file) that contains your solution to the homework accompanying Project 2.

By October 17, there will be links on the class webpage that will enable you to turn in your zip file electronically. Turn in the file by the due time above. Give yourself enough time to be sure you can turn something in, because we will not accept excuses like "My network connection at home was down, and I didn't have a way to copy my files and bring them to a SEASnet machine." There's a lot to be said for turning in a preliminary version of your program, report, and homework early (You can always overwrite it with a later submission). That way you have something submitted in case there's a problem later. Notice that most of the test data portion of your report can be written from the requirements in this specification, before you even start designing your program.

The writeup Some Things about Strings tells you what you need to know about strings for this project.

As you develop your program, periodically try it out under another compiler (g31 on cs31.seas.ucla.edu if you're doing your primary development using Visual C++ or Xcode). Sometimes one compiler will warn you about something that another is silent about, so you may be able to find and fix more errors sooner. If running your program under both environments with the same input gives you different results, your program is probably relying on undefined behavior (such as using the value of an uninitialized variable), which we prohibit.