Programming Assignment 2
Phoney Baloney

Time due: 9:00 PM ~~Thursday, October 13~~ Saturday, October 15

Before you ask questions about this specification, see if your question 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 Surprizin' Wireless is to produce the bill for a customer given their mobile phone usage. The program you write must accept as input the number of minutes used (an integer), the number of text messages sent and received (an integer), the customer's name, and the billing month.

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

```
Minutes used: 385
Text messages: 423
Customer name: Alex Bell
Month number (1=Jan, 2=Feb, etc.): 11
---
The bill for Alex Bell is $48.46
```

According to Surprizin's usage plan:

- The basic charge is $45, which includes up to 400 free minutes and up to 250 free text messages.
- The charge for minutes beyond 400 is 40¢ per minute.
- Each text message from the 251st to the 500th cost 2¢ if the month is October through May. However, Surprizin' offers a summer discount: from June though September, each such message costs only 1¢
- Text messages beyond the 500th cost 11¢ apiece.

As an example, Alex above would be charged the base $45 and no minutes charge, since he didn't use more than 400 minutes. He'd be charged $3.46 for the 173 texts after his first free 250; those are 2¢ apiece in November.

Here's another example:

```
Minutes used: 417
Text messages: 531
Customer name: Elisha Gray
```

```
Month number (1=Jan, 2=Feb, etc.): 7
---
The bill for Elisha Gray is $57.71
```

Elisha's 251st through 500th texts were billed at the July rate.

You can test your understanding of the rate schedule by experimenting with the phone charge calculator we found at the Surprizin' Wireless website.

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), 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. That line you write must be in one of the following five forms; the text must be **identical** to what is shown, except that *italicized* items are replaced as appropriate:

- If the number of minutes used is negative:
  `The number of minutes used must be nonnegative.`
- If the number of text messages is negative:
  `The number of text messages must be nonnegative.`
- If nothing was provided for the customer name:
  `You must enter a customer name.`
- 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 bill for` *`customer`* `is $`*`amount`*

In the last case, *customer* must be the customer name as entered, and *amount* must be the correct answer, shown as a number with exactly exactly two digits to the right of the decimal point. 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 printing a line with exactly three hyphens in *all* cases.
- Writing more than one line after the line with three hyphens. Don't, for example, add a gratuitous "Can you hear me now? That's Surprizin'!"
- Writing lines like these:

`You must enter a customer name`	*missing period*
`Teh bill for Alex Bell is $48.46`	*misspelling*
`The Bill for Alex Bell is $48.46`	*wrong capitalization*
`The bill for Alex Bell is $ 48.46`	*extra space*
`The bill for Alex Bell is 48.46`	*missing dollar sign*
`The bill for Alex Bell is $48.46.`	*extra period*

- `The bill for Alex Bell is $48.460` *extra digit*
- `The bill for Tom Edison is $63` *missing decimal point and digits*

Your program must gather the minutes used, the number of texts, the customer name, and the month number in that order. However, if you detect an error in an item, you do not have to request and get the remaining items if you don't want to; just be sure you print 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 any 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 input item.

You will not write any loops in this program. This means that each time you run the program, it handles only one bill. 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 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 your program does **not** have to deal with (because you don't yet know enough to know how to do this nicely) is the case of not finding an integer in the input where an integer is expected. Our grading tool will not, for example, supply the text `Bell` or the number `307.8` when your program requests the minutes used.

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 **bill.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.doc** or **report.docx** (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 correctly. (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:

    Positive miles and gallons, typical mpg (400, 15)
    Positive miles and gallons, good mpg (400, 9)
    Positive miles and gallons, bad mpg (400, 25)
    Negative miles (-300, 12)
    Negative gallons (300, -12)
    Zero miles (0, 10)
    Zero gallons (300, 0)
    Boundary case for bad mpg (400, 10)
    …

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

By October 12, there will be a link 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 (g++ if you're doing your primary development using Visual C++, or Visual C++ if you're doing your primary development using g++ (e.g., with Xcode on a Mac)). 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.