Programming Assignment 2
Royalty Reckoning

Time due: 9:00 PM Thursday, October 15

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.)

Getpix is a company that provides streaming movies and DVDs by mail, and for a given title, needs to pay the licensor royalties based on the number of units sent out each month. You've been hired to write a program to do the appropriate computation.

Your program must accept as input the number of units sent out during a given month, the movie title, the base price, and whether the movie is a premium item. The output will tell how much in royalties must be paid.

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

```
Units sent: 558
Title: Cretaceous World
Base price: 19.15
Premium item? (y/n): y
---
Cretaceous World earned $1173.51 in royalties.
```

Here is Getpix's standard royalty schedule:

- For the first 400 units sent, the royalty rate per unit is 9% of the base price.
- In addition, for the next 800 units sent (beyond the initial 400), the royalty rate per unit will be 13% of the base price if the product is not a premium item, or 16% of the base price if the product is a premium item.
- In addition, for units sold beyond the initial 1200, the royalty rate is 14%.

As an example, for Cretaceous World above, the base price is $19.15. The royalty per unit for the first 400 units is 9% of $19.15, which is $1.7235, so those 400 units earn $689.40. For the next 158 units, the royalty per unit is 16% of $19.15, which is $3.064, for another $484.112. (It's 16% instead of 13%, because this is a premium item.) The total royalty earned is $1173.512. (A requirement below tells you to display it rounded to the nearest cent.)

Here's another example:

```
Units sent: 200
Title: Straight outta Westwood
Base price: 15.90
Premium item? (y/n): n
---
Straight outta Westwood earned $286.20 in royalties.
```

You can test your understanding of the royalty schedule by experimenting with this royalty calculator.

Your program must collect the information for one movie 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. 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 units sent is negative:
  `The number of units sent must be nonnegative.`
- If an empty string was provided for the title:
  `You must enter a title.`
- If the base price is negative:
  `The base price must be nonnegative.`
- If the premium status is not `y` or `n` (must be lower case):
  `You must enter y or n.`
- If the input is valid and none of the preceding situations holds:
  `title` `earned $`*`amount`* `in royalties.`

In the last case, *title* must be the title as entered, and *amount* must be the correct answer, shown as a number with at least one digit to the left and exactly two digits to the right of the decimal point, rounded to the nearest cent. (See pp. 32-33 in 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 "Report royalty income on Schedule E."
- Writing lines to `cerr` instead of `cout`.
- Writing lines like these:

```
The Martian urned $123.40 in royalties.   misspelling
The Martian Earned $123.40 in royalties.  wrong capitalization
The Martian earned $123.40.               missing text
The Martian earned $ 123.40 in royalties. extra space
The Martian earned 123.40 in royalties.   missing dollar sign
The Martian earned $123.40 in royalties   missing period
The Martian earned $123.400 in royalties. extra digit
The Martian earned $123 in royalties.     missing decimal point and digits
```

Your program must gather the units sent, the title, the base price, and the premium status 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 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

movie. 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 title consisting of seventeen spaces and nothing more, treat that as you would any other non-empty string: as a valid title. (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 a number in the input where a number is expected. Our grading tool will not, for example, supply the text `too much` when your program requests the base price. You do not have to deal with not finding an integer in the input for the units sent; our tool will not, for example, supply `838.7` for the units sent.

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 **royalty.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:

        More surveyed than the total for keeping and returning (1000, 413, 382)
        Fewer surveyed than the total for keeping and returning (500, 413, 382)
        Zero surveyed (0, 100, 100)
        Data giving a non-integer percentage (1000, 413, 382)
        More reponses for keeping than returning (1000, 413, 382)
        Equal reponses for keeping and returning (1000, 500, 500)
        …

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 14, 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 (g++ if you're doing your primary development using Visual C++, or Visual C++ if you're doing your primary development using clang++ or 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.