Fall 2019 CS 31

Programming Assignment 2
One Fine Day

Time due: 11:00 PM Thursday, October 17

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.)

The federal judge in the college admissions scandal has devised a formula for determining a suggested amount for the fine to be imposed on a defendant convicted of fraud in the college admissions scandal. The amount of the fine depends on the amount the defendant paid to Slick Stinger, who arranged for cheating on the SAT or ACT or for faking athletic competition credentials, the latter being a more serious offense that should be penalized more. The job of writing the program that implements the formula has come to you.

Your program must accept as input the name of the defendant, the amount the defendant paid Stinger to facilitate cheating (in thousands), and whether or not the fraud involved faking athletic achievement.

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

```
Defendant: Horton Lapland
Amount paid (in thousands): 75
Fake athlete? (y/n): n
---
The suggested fine for Horton Lapland is $49.9 thousand.
```

According to the fine formula:

- The base amount of the fine is $20 thousand.
- For up to the first $40 thousand paid to Stinger, 66% of that amount is added to the fine.
- In addition, for up to the next $210 thousand paid (beyond the initial $40 thousand), the fine will be increased by of 10% of that additional amount paid. However, if athletic achievement was faked, the addition to the fine will be 22% of that additional amount paid instead of 10%.
- In addition, a further fine will be imposed in the amount of 14% of the amount paid to Stinger above $250 thousand.

As an example, the suggested fine for Horton Lapland above would be the $20 thousand base amount, plus $26.4 thousand for the first $40 thousand paid, plus $3.5 thousand for the next $35 thousand paid (10% of $35 thousand instead of 22%, since there was no athletic fakery), for a total suggested fine of $49.9 thousand.

Here's another example:

```
Defendant: Duplicity Muffin
Amount paid (in thousands): 15
Fake athlete? (y/n): n
---
The suggested fine for Duplicity Muffin is $29.9 thousand.
```

You can test your understanding of the fine formula by experimenting with this prototype suggested fine calculator we found.

Your program must collect the information for one defendant 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 four forms; the text must be **identical** to what is shown, except that *italicized* items are replaced as appropriate:

- If an empty string was provided for the defendant:
  `You must enter a defendant name.`
- If the amount paid is negative:
  `The amount paid must not be negative.`
- If the response to the fake athlete prompt 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:
  `The suggested fine for` *`defendant`* `is $`*`amount`* `thousand.`

In the last case, *defendant* must be the defendant the user entered, and *amount* must be the correct suggested fine, shown as a number with at least one digit to the left and exactly one digit 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 very few points for correctness on this project, no matter how much time you put into it, because the mistake will cause your program to fail most or all of the test cases we run:

- 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 "And there'll be some jail time, too!".
- Writing lines to `cerr` instead of `cout`.
- Writing lines like these:

```
The sugested fine for August Whopayus is $99.6 thousand.        misspelling
The suggested Fine for August Whopayus is $99.6 thousand.       wrong capitalization
The suggested fine for August Whopayus will be $99.6 thousand.  wrong text
The suggested fine for August Whopayus is $99.6.                missing text
The suggested fine for August Whopayus is $ 99.6 thousand.      extra space
The suggested fine for August Whopayus is 99.6 thousand.        missing dollar sign
The suggested fine for August Whopayus is $99.6 thousand        missing period
The suggested fine for August Whopayus is $99.60 thousand.      extra digit
The suggested fine for August Whopayus is $100 thousand.        missing decimal point and digits
```

Your program must gather the defendant, the amount paid, and the fake athlete 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 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 defendant. 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 defendant consisting of seventeen spaces and nothing more, treat that as you would any other non-empty string: as a valid defendant. (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 a number in the input where the amount paid is expected. We promise that our grading tool will not, for example, supply the text `a ridiculous amount just to get into USC` when your program requests the amount paid. Notice that the amount paid need not be an integer.

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 **fine.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 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 supporting and opposing (1000, 413, 382)
        Fewer surveyed than the total supporting and opposing (500, 413, 382)
        Zero people surveyed (0, 100, 100)
        Data giving a non-integer percentage (1000, 413, 382)
        More support than opposition (1000, 413, 382)
        Equal numbers supporting and opposing (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 16, 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.