Programming Assignment 2
Sales Tax Payment Calculator

Time due: 09:00 PM Sunday, ~~January 26rd~~ February 1st.

Here is an example of a dialog for the program I am asking you to create (user input is in **boldface**):

```
State Name: Texas
Purchase amount: 30.56
Provide the month: January
Provide the day: 1
Provide the year: 2021
Please pay a total of $33.06
```

In addition to correctly calculating the total payment answer, which includes all necessary taxes, your program needs to handle errors as shown below:

```
State Name: texas
Purchase amount: 30.56
Provide the month: January
Provide the day: 2
Provide the year: 2020
Invalid state!

State Name: Texas
Purchase amount: -30.56
Provide the month: January
Provide the day: 2
Provide the year: 2020
Invalid amount!

State Name: Texas
Purchase amount: 30.56
Provide the month: January january
Provide the day: 2
Provide the year: 2020
Invalid month!

State Name: Texas
Purchase amount: 30.56
Provide the month: February
Provide the day: -10
Provide the year: 2000
Invalid day!

State Name: Texas
Purchase amount: 30.56
Provide the month: March
Provide the day: 3
Provide the year: -101
Invalid year!
```

```
State Name: Texas
Purchase amount: 30.56
Provide the month: 10
Provide the day: -101
Provide the year: -101
Invalid month!

State Name: Soth dakota
Purchase amount: 30.56
Provide the month: January
Provide the day: 2
Provide the year: 2020
Invalid state!

State Name: Texas
Purchase amount: 0
Provide the month: January
Provide the day: 2
Provide the year: 2020
Invalid amount!

State Name: Georgia
Purchase amount: 30.56
Provide the month: march
Provide the day: 2
Provide the year: 2020
Invalid month!

State Name: north Carolina
Purchase amount: -9
Provide the month: February
Provide the day: -10
Provide the year: 2000
Invalid state!

State Name: Texas
Purchase amount: 30.56
Provide the month: March
Provide the day: 3
Provide the year: 2028
Invalid year!

State Name: Texas
Purchase amount: 30.56
Provide the month: jan
Provide the day: -101
Provide the year: 2028
Invalid month!
```

These are the data validation rules I want your program to enforce:
- the first letter of the state name must be capitalized
- **For state names with two words, both words need to be capitalized.**
- the purchase amount must be a positive number greater than 0 (zero).

- the first letter of the month must be capitalized
- the day must be between 1 and 31 (you do not have to validate if the day is correct for the corresponding month. For example, February 31 is OK).
- the year must be between 1 and 2025

Your program must gather the state name, purchase amount, month, day and year, in that specific order. **You should gather all inputs first before checking for any errors.** If you detect more than one error, write only the error message for the first erroneous input item, as shown above.

A sales tax table is provided for your reference to develop your algorithm to calculate the total payment. The total payment should be the purchase amount plus the required state, local, and levy add-on sale taxes. The State column on the table has the name of all 50 states. The stateTaxRate column has the state sales tax rate (in percentage), the avglocalTaxRate column has the local sales tax rate (in percentage). You can assume that all cities and counties in that state use the avglocalTaxRate to collect their sales tax shares. The Levy (add-on) column has the extra sales tax (in percentage) that the local government wants to collect. The Free Tax Dates column specifies dates that no sales tax is required (including the levy add-on). The total payment should be the same as the purchase amount on free tax dates. Keep in mind that not all states have free tax dates.

Your program must collect the information for one purchase in the manner indicated by the examples and then write to cout exactly one line in the format required below. We will judge the correctness of your program by examining your output. Your program's output must be in one of the following four forms; the text must be **identical** to what is shown (except that the *italicized* portions described below):

- If the user enters a state name that is different than what is in the sales tax table:
  `Invalid state!`
- If the user enters a purchase amount that is not a positive number (with or without decimal places) that is greater than 0 (zero):
  `Invalid amount!`
- If the user enters a month value which is not "January", "February", "March", "April", "May", "June", "July", "August", "September", "October", "November" or "December":
  `Invalid month!`
- If the user enters a day value less than 1 or greater than 31:
  `Invalid day!`
- If the user enters a year value less than 1 or greater than 2025:
  `Invalid year!`
- Otherwise:
  `Please pay a total of $`*`xx.xx`*
  xx xx is the calculated total payment amount with two decimal places. You will need to have two decimal places regardless (for example, 36.00 should not be output as 36).

- Your program's output 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:

- Writing any extra spaces on the output line.
- Writing more than one line of output. Don't, for example, add a gratuitous "Thank you for using my great sales tax program!"
- Writing lines to `cerr` instead of `cout` .
- Writing lines like these:

```
Please pay a total of 33.06                missing $ sign
Please pay a totl of $33.06                misspelling
please pay a total of $33.06   lower case p in the word please
Please pay a total of $ 33.06              extra spaces
Please pay $33.06                         missing words
```

You will not write any loops in this program. This means that each time you run the program, it handles only one purchase calculation. 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. Also, no user-defined functions.

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 two files and nothing more:

1. A text file named **payment.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 **your name** :
    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.)

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.