This programming project is due on **Wednesday, October 23, 2019**.

File Format Conversion Program

This project involves designing and creating a C++ program that will convert a "pipe-delimited" text file to a file in CSV format. (The vertical bar character, " **|** ", is often called a "pipe" character. So a text file with vertical bar delimiters is often called a "pipe-delimited" file.)

The program must provide commands for the user to operate the program. At a minimum, the following commands **must** be supported:

Command	Description
i (input)	Allows the user to specify an input file name.
o (output)	Allows the user to specify an output file name.
p (process)	Process the data in the input file, writing the results to the output file. (The user will run the "**p**" command *after* running the "**i**" and "**o**" commands.)
h (help)	Output "help text" that describes the user commands.
q (quit)	Exit the program

The "**p**" command must repeat the following steps for **each** line of text in the input file:

- Read one complete line of text from the input file into a **string** object. (This string is called the *input record*.)
- Convert the *input record* to an array of **string** objects. Each "field" of the input should become one element in the array of strings.
- Modify each string in the array, according to the rules discussed later in this document.
- Write the array of strings to the output file, as one line of text, with each field followed by a **comma**.
- Repeat these steps for each line of text in the input file.

The following sections of this document discuss the input and output file formats and the data conversion process in more detail.

Comma-Separated-Values ("CSV" file format)

There are many discussions of "CSV" file format on the internet. It is a mechanism to represent multiple fields per line of text, without using any special, non-standard characters.

The obvious complicating factor with a file format like this is: How do you handle the case where the field delimiter (in this case, a comma) is present in the data? The short answer is to surround those fields with double-quote characters. But what if the original data *contains* a double-quote character? CSV addresses this issue by "escaping" the double-quote characters: it replaces each original double-quote character with TWO double-quote characters.

A further complication ***would*** be: What if the data field itself contains the record separator (new-line) character? Some CSV implementations address this, but for this project we do not consider this case.

Steps to Convert "pipe-delimited" data to CSV

The file processing should consist of the following steps for ***each*** data record (each line of text) in the input file:

1. First, read one complete line of text (from the input file) into a **string** object. This string is called an ***input record***.
2. Split the ***input record*** into separate fields, assuming the fields are delimited by the vertical bar ("**|**") character. That is, create an array of **string** objects, where each element of the array contains the contents of *one* data field from the input record.

   NOTE: the array of **string** objects does not need to contain the vertical bar ("**|**") character itself. The vertical bar ("**|**") character is needed when splitting up the ***input record*** into separate fields, but after that step is done, it is no longer needed.
3. Convert the contents of each field according to the following rules:

   - If there are any pre-existing double-quote characters in the field, then each of those should be replaced by TWO double-quote characters. (That is, insert another double-quote character.)
   - Any field that contains one or more commas or double-quote characters must be enclosed in double-quotes. (That is, if a field contains any double-quote characters or commas, then insert another double-quote character at the beginning of the field, and also at the end of the field.)
4. The modified fields for the current data record should be written to the output file, separated by commas. That is, write **each** field to the output file, and follow it with a comma. When all fields have been written to the output file, output a new-line character ("**`\n`**") to the output file.

Note that each line of text in the input file corresponds to one line of text in the output file. That is, the design of the program can focus on processing one line of text at a time.

Design Restriction

You should be able to complete this project using the **string** library, as described in Chapter 10 of our textbook. You may **not** use the stringstream class or any other prepackaged library other than the **string** class. The purpose of this project is go give you experience developing your own string manipulation software. If there is some other library you believe you need, get permission from the instructor first.

Manually Checking the Output Data

If the name of the output file has a suffix of "**`.csv`**", then there are two programs you can use to view the result:

- **First**: manually open the output file using a text editor such as *Notepad*. This will show exactly what text has been written to the file.
- **Second** (optional): manually open the "**`.csv`**" file using *Microsoft Excel*, and then save it in "**`.xlsx`**" format (as an Excel workbook).

The data shown in the following tables contains a representative sample of test cases:

Sample input file: **sample1.txt**

```
feet per mile|5,280|quotes in "middle" of field|0.00|0.00|1,000.00
"beginning" quote|quote at "end"|Boston, Massachusetts|123.29|0.00|876.71
|Text with comma, in the middle|"Quoted Text"|58,378.53|0.00|798.18
non-descript text|12/31/2012|||0.00|933.51
"Alpha Kappa Mu"|BHCC Honors Society|Recognition for all that hard work
```

Sample output file: **sample1_result.csv**

```
feet per mile,"5,280","quotes in ""middle"" of field",0.00,0.00,"1,000.00",
"""beginning"" quote","quote at ""end""","Boston, Massachusetts",123.29,0.00,876.71,
,"Text with comma, in the middle","""Quoted Text""","58,378.53",0.00,798.18,
non-descript text,12/31/2012,,,0.00,933.51,
"""Alpha Kappa Mu""",BHCC Honors Society,Recognition for all that hard work,
```

sample1_result.xlsx: (optional, manual test)

When the sample output file (above) is manually loaded into *Microsoft Excel*, it looks like the following:

	A	B	C	D	E	F
1	feet per mile	5,280	quotes in "middle" of field	0	0	1,000.00
2	"beginning" quote	quote at "end"	Boston, Massachusetts	123.29	0	876.71
3		Text with comma, in the middle	"Quoted Text"	58,378.53	0	798.18
4	non-descript text	12/31/2012			0	933.51
5	"Alpha Kappa Mu"	BHCC Honors Society	Recognition for all that hard work			

Note: the requirement is for your program to produce the **.csv** file *only*. If you wish to perform this (optional) final check, manually open your output file with Microsoft Excel and save it in **.xlsx** format.

Test Data

Two sample input files are provided with the project document: **sample1.txt** and **sample2.txt**. The program should work correctly with these files, as well as general files of similar format.

Think about what the program is trying to do at a conceptual level, and consider other test data which you think may be appropriate. What test data can you think of which is appropriate for the way you are designing your program?

Experiment with *Microsoft Excel*, and save various spreadsheets in different formats, including "**CSV**" format.

Sample Console Output

The following is a sample session from the solution to Project 2. (The user input is shown in **Bold** text.):

```
Command: h
Supported commands:
        CI              Close input file.
        CO              Close output file.
        h               print this help text.
        i               open file for input
        o               open file for output
        p               (lower case 'p') process data file.
        q               quit (end the program).

Command: i
Enter name of input file: sample1.txt

Command: o
Enter name of output file: sample1_result.csv

Command: p
Input data = feet per mile|5,280|quotes in "middle" of field|0.00|0.00|1,000.00
Input data = "beginning" quote|quote at "end"|Boston, Massachusetts|123.29|0.00|876.71
Input data = |Text with comma, in the middle|"Quoted Text"|58,378.53|0.00|798.18
Input data = non-descript text|12/31/2012|||0.00|933.51
Input data = "Alpha Kappa Mu"|BHCC Honors Society|Recognition for all that hard work
End of file encountered.

Command: CI

Command: CO

Command: q
Exitting program with status = 0
Press any key to continue . . .
```

Extra Credit Enhancement:

As always, before attempting the *extra credit* portion of this project, save your working solution for the *basic* version in a safe place. That way, if you run out of time before completing the *extra credit* version, you will still have a working program to submit when the deadline arrives. One convenient way to do this is to keep the *basic* and *extra credit* versions in separate **Visual Studio** projects.

The basic requirement for this assignment, as described in this document, is to convert a pipe-delimited text file to an equivalent CSV-format file. If you have time, and feel ambitious, you can add a feature that allows conversion in the opposite direction: **from** CSV-format **to** pipe-delimited format. With this enhancement, the program would accept a **.csv** file as input, split the data into its individual fields and then write the output file using pipe delimiters.

This enhancement requires re-thinking the file conversion process in the *reverse* order. For each line of the "**`.csv`**" input file, the program needs to split the file into separate fields. When starting from a **CSV** format, the input delimiter character is the comma, but you need to be careful how you design the program: any comma character enclosed in double-quote characters should **not** be interpreted as a field separator. For this reason, the program needs to maintain some "state" information, as it progresses through the text:

- If the *first* character in a data field is a double-quote character, then the program must scan for the corresponding double-quote character at the end of the field.
- But what if there are double-quote characters inside the data field? How can your program distinguish between a double-quote character at the end of the field and a double-quote character inside the data field?

Think about this design before you attempt to code it: try some "what-if" scenarios using a pencil and paper.

Implement this enhancement as a new "command" (the '**x**' command), which the user selects from the command prompt. (Be sure to add this new command to the output for the "**h**" command.)

Project Deliverables:

The project source file(s) should be submitted by Moodle, using the Moodle Activity:

CIT237_Project2

Submit your **`.cpp`** file(s) and any **`.h`** file(s) which you create. I will need to compile your code on my home computer in order to grade it. If you are submitting more than one file (**`.cpp`** and/or **`.h`**), do **not** enclose the files in a ZIP file. Moodle will allow you to submit multiple source files.
Do ***not*** submit the entire *Visual Studio* project.
Do ***not*** include the *Visual Studio* project folders, or any binary files.

If you have developed your program using some IDE / compiler *other* than the one we have on our classroom computers, be sure to compile and test your final version on one of the Windows 10 computers in our classroom before you submit it.

Grading Criteria

The project will be graded according to the following grading criteria:

Feature	Portion of grade
1. The program functions correctly.	65%

2. In the **main** function of the program, there is a loop which contains code to support *at least* the following input commands:
 	
`h` `print help text.`
`i` `open file for input`
`o` `open file for output`
`p` `(lower case 'p') process data file.`
`q` `quit (end the program).`
	3%

3. The "command loop" in the **main** function should continue until the user enters a '**q**' command.	3%
4. Each of the commands (except the '**q**' command) should call a separate function. That is, the "**main**" function should not be excessively long.	3%
5. The '**i**' and '**o**' commands must each call another function that asks the user to specify the name of the input or output file. That is, the project must work for files with any name.	3%
6. The program is clearly organized and **commented** so that it is easy to read and understand. At a minimum, there should be a comment at the beginning of each function that explains what that function does. Use your judgement regarding any additional comments that may be needed to make the program easy to understand, without over-commenting the program. (As you get more experience, your judgement about this will improve.) Do **NOT** put all of your code in the **main** function or any other function.	10%
7. Use good variable names and function names: • A variable name or function name should indicate something about what that variable or function does in the program. • Variable names and function names should be not too short and not too long.	5%
8. Cleanup any unused portions of code, such as "failed attempts" that you later replaced.	3%
9. Place a brief summary of the program in comments at the beginning of the source file(s). Also be sure these comments have your name and the due-date for the project.	3%
10. Cleanup any irrelevant comments	2%
Total:	100%