CIS 251 • C++ Programming
Laboratory Assignment 9: Strings

In this laboratory assignment you will learn how to use C++ syntax to process several lines of text in an input file, displaying those lines of text that meet a certain criterion.

Background

A college maintains a database of information on students currently enrolled in classes on its campuses, and it exports a subset of this information to the file `studentInfo.csv` with each line containing the last name, first name, middle initial, student identification number, and major (separated by commas) of a single student. An example line follows:

```
Smith,Judy,D,X120987,English
```

Practically every modern spreadsheet program supports exporting a worksheet to a comma-separated values file (CSV), a plain text file with each row on a separate line and the columns separated by commas. These programs can be used to generate a quick sample data file.

The college wants a program that allows a user to enter a major (single or multiple words) and displays all of the students enrolled in that major, with a count of these students at the end of the list. This program will make use of the `string` class and its member function `find`.

Step 1: Launch Visual Studio and Set Up an Empty C++ Project with Code

Open Visual Studio from the Start Menu. Start a new project of installed template / type **Visual C++**, subtype **Win32**, template **Win32 Console Application**, project name and location as desired (usually the Visual Studio **Projects** folder), and **Create directory for solution** box **unchecked**. In the **Win32 Application Wizard**, go to **Application Settings**, **check** the box for **Empty project**, and click **Finish**.

In the **Solution Explorer**, right-click **Source Files**, and select **Add > New Item…** from the shortcut menu. Select the category **Code** and the file type / template **C++ File (.cpp)**. Enter the file name **FLast_Lab09.cpp** (with your **first initial** and **last name** replacing **F** and **Last**, respectively). Click **Browse…** to select the location where you want to save your code (flash drive preferred).

Step 2: Begin Your Code

Add a single block comment or a set of line comments including your name, the course number, the laboratory assignment number, the date (optional), and a brief description of the program's contents. Place each of these items on a separate line. As you continue your code, you may go ahead and add comments to the statements or groups of statements that you believe deserve additional explanation, or you may wait until after you have written your entire program to add comments.

Next, enter the standard items that are part of your program: the `include` and `using` directives, the header for the `main` function, the opening curly bracket, a blank line after the bracket, the `return` statement, and the closing curly bracket.

Remember the `include` directives that support file input and the C++ standard `string` class (which is not the same as the C-based `char` array function library). Also, in case the input file does not exist, include the library that provides the `exit` function.

Step 3: Declare Your Variables

The following table describes the variables needed in your main function (one per table row):

Description	Data Form	Initial Value
Variable for `studentInfo.csv`	Input file stream	n/a
Major for which to search	Text	From user input
The line of data for one student	Text	From user input
Count of students in that major	Whole number	0

Add a declaration for each variable to the `main` function, initializing any variables that have an initial value specified.

Do not declare a separate variable for each value on a line in the text file. Your program will not be reading each value into a separate variable; instead, it captures the entire line in one input statement.

The output generated by this program is displayed in the console; there is no file output in this program.

Step 4: Open the Input File

Before you can use the input file, you must invoke the `open` method on the input file stream variable. Then, check to make sure that there were no problems opening the file by using the `fail()` method; if it returns `true`, you should display an error message in the console and force the program to end by calling the `exit` function (which may require the additional library `cstdlib`).

Step 5: Prompt the User for the Major

Display a prompt that instructs the user to enter the major whose students the program should display. Read this major as a line of text from the **console**, not from the input file.

The major could be multiple words separated by spaces. Instead of using a normal `cin` statement with the input insertion operator, you should use the form of `getline` used for `string` input.

Step 6: Use a Loop to Read and Process the Input File Data

Just as a normal insertion-operator-based input statement involving an input file stream returns `false` once the end of the file is reached, the `getline` function also returns a `bool` value indicating the success or failure of the input statement. Thus, you can use a call to `getline` as the expression to control a `while` loop. Call `getline` in this manner to store the next line from the **input file** in the `string` variable for the student's data line.

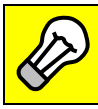
Make sure the first argument to each call to `getline` is the stream from which you want to capture the next line of input. In step 5, this was the console; in step 6, use the input file variable instead.

If you prefer that the program loops using the `eof()` member function, you can structure this loop in the same way. Make sure that your program has the priming call to `getline` just before the loop header and an identical call to `getline` as the last statement in the loop body.

Inside this loop, the program should determine if the line just read from the input file contains the major entered by the user. The syntax of the `find` method uses the `string` to be examined as the calling object (before the dot and method name) and the `string` to find as the argument:

```
theStringBeingExamined.find(theStringBeingSearchedFor)
```

The method returns the position where the search `string` begins in the examined `string` if it is found; if not, it returns a special value, `string::npos`. Thus, if the `find` method returns a value not equal to `string::npos`, the program should display the line from input and increment the count of students in that major. Nest this decision structure inside the body of the loop.



Call the `find` method in the Boolean expression for the `if` statement, comparing the method's return value to `string::npos`.

Make sure that you position the `string` variables in the right order. The `string` variable used as the calling object should be the longer `string` to search within; the `string` variable used as the argument should be the shorter `string` for which you want to search.

The output displayed on the screen will still have commas between the values (see the example run on the next page). Additional processing would be required to separate these values with some form of whitespace instead.

Step 7: Close the Input File and Display the Count

After the loop, close the input file using the input file stream variable; then, use an output statement to display the number of students whose major matched the search major.

Step 8: Build Your Project and Run the Executable

Build your project. If there are syntax errors, double-click each error description to get an idea of where the error is. Once the build succeeds, you'll need to create an input file for the program to draw from (or download the sample file from the assignment page); remember that Visual Studio looks for this file in the project folder, which may be different from the location of your source code file. Once you've placed this file in the correct location, select **Start Without Debugging** from the **Debug** menu. Run the program several times, searching for different majors and verifying that the students for that major are displayed as expected. You may also choose to modify the input file to include additional students or change the majors of selected students.

Example Run (your formatting and spacing may vary):

```
Student Major Search

Enter the major to search for: Computer Science

The following are the students who match that major:
Andrews,Lilian,C,X107293,Computer Science
Harris,Martin,E,X104501,Computer Science
Johnson,Anne,I,X108764,Computer Science
Vaughn,Veronica,L,X117032,Computer Science

Total of Computer Science majors: 4
```

This is not a perfect solution; for example, if someone's last name happens to be English, he / she will be counted as an English major. It is possible to force the `find` method to start at a specific position.

Step 9: Add Inline Comments and Submit Your Code

Make sure that your code has **three or more inline comments** near the statements that you believe warrant the additional description (especially those that pertain to `string` variables and methods). These can be simple line comments at the end of a line of code that needs explanation, or block comments where each precedes a series of statements described in the comment.

In Blackboard, return to the Assignments page for the current assignment. Under the **Submission** section, click **Add Attachments**. Click **My Computer**; then, click the first **Browse…** button. Navigate to where you saved the source code file **FLast_Lab09.cpp** (with your name), select it, and then click **OK**. You do not have to attach your input output file. Verify that your file appears in the Submission section, and click the **Submit** button at the bottom of the page.

If you do not click the Submit button, your attachment will not be saved!