CIS 251 • C++ Programming
Laboratory Assignment 7: File Input and Output

In this laboratory assignment you will learn how to use C++ syntax to read student data from a file and write it to four separate files based on certain criteria.

Background

A local college stores information about students pursuing a degree in Computer Information Systems in a file named `cisdegree.txt` (a sample is included with this laboratory assignment). Each line in the text file includes a student's identification number (eight digits), last name, first name, middle initial, grade point average, and a single character representing one of the following degree options:

A – applications N – networking P – programming W – web development

An example line:

```
10203846 Hart Bartholomew S 3.27 P
```

The college wants the data from this file divided into four separate files, one for each degree program – `appmajors.txt`, `netmajors.txt`, `progmajors.txt`, and `webmajors.txt` – with each file listing the students in that particular major. Since each file will only include the students from one major, you don't need to include the character for major. For example, the aforementioned student would be included in the file `progmajors.txt` with the following line:

```
10203846 Hart Bartholomew S 3.27
```

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_Lab07.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.

Because the program deals with `string` data, file input and output, performing an immediate exit, and the potential of console output to display an error message for a nonexistent input file, it requires `include` statements for the libraries that handle these functions.

Step 3: Declare Your Variables

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

Description	Data Form	Initial Value
Student number	Whole number	From input
Last name	Text	From input
First name	Text	From input
Middle initial	Single character	From input
Grade point average	Real number	From input
Major code	Single character	From input
Variable for `cisdegree.txt`	Input file stream	n/a
Variable for `appmajors.txt`	Output file stream	n/a
Variable for `netmajors.txt`	Output file stream	n/a
Variable for `progmajors.txt`	Output file stream	n/a
Variable for `webmajors.txt`	Output file stream	n/a

Add a declaration for each variable. Remember – file names and stream variable names are different!

You must have a separate file stream variable for each file you use in your program. Make sure that you declare an `ifstream` variable for each input file and an `ofstream` variable for each output file.

Step 4: Open Each File

Before the program attempts to read from or write to the files, it must invoke the `open` method for each file. For the input file, use the `ifstream` variable to open the file; 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.

Some compilers require you to include an extra library, `<cstdlib>`, to use the exit function.

The program opens the four output files using the `ofstream` variables without using the `fail` method.

By default, a program overwrites an output file if it already exists (which is what you want for this program); however, to append to the existing file instead, add the argument `ios::app` to each `open` method call (but don't add this argument to the `open` method calls in this assignment).

Step 5: Use Number Formatting with Each Output File

Chapter 2 mentions a set of three method calls that formats the output of any decimal value. This program should write the grade point average of each student in fixed notation, with the decimal point and two digits after the decimal point required. To make this work for all four files, the program must invoke all three methods on each `ofstream` variable (a total of **twelve statements**).

The formatting methods applied to one output stream (`cout` or an `ofstream` variable) do not automatically apply to the others; a program must apply them to each output stream individually!

Step 6: Create the Input-Driven Loop

There are two very different ways to read all of the records in a file: either make the input statement the Boolean expression that controls the loop (in parentheses), or invoke the method `eof()` on the `ifstream` variable as the Boolean expression (in which case, the program contains the input statement both before the loop header and as the last statement inside the loop body). Refer to the lecture notes for the syntax in either case. Make sure that the input statement includes **all six variables** that are part of the student's record.

As with multiple items in a `cout` statement, separate the six input variables with the input insertion operator >>.

If a program uses the `eof()` method, it must contain the input statement twice: once before the loop, and once as the last statement inside the loop. Otherwise, the program may become stuck in an infinite loop, or it may end up processing one of the input records zero or two times.

When reading input from a file, a program should not display prompts; the file does not need to be "told" to send the values into the program.

Step 7: Process an Input Record in the Loop Body

To determine to which of the four files to write the student's data, use a decision structure (a series of `if` statements or a `switch` structure). Compare the variable storing the character representing the student's major to each of the four values, one in each `if` statement or `case`. The statement executed for each `if` statement or `case` should be an output statement that uses the `ofstream` variable for the corresponding file and each of the five variables to be written for the student.

When the variable being tested in a decision structure is a `char`, you must put single quotation marks around the character values (e.g., `'P'`) in each Boolean expression or `case` value.

If using `if` statements, don't forget to use **two equal signs** for the comparison operator in a Boolean expression.

The values in the output file should be separated by spaces. Include these in quotation marks between the variables. Example:

```
outFileVar << value1 << " " << value2 << endl;
```

Nest the decision structure inside the loop body.

If you are using the `eof()` form of the loop, make sure to place the decision structure **before** the input statement inside the loop. Don't read another set of values without first processing the previous values.

At this point the input loop is complete.

Step 8: Close the Files

To make sure that there are no "buffered" lines of output that have yet to be written to the output files, and to prevent damage to any of the files, invoke the `close()` method on the `ifstream` variable and each of the four `ofstream` variables (a total of five calls to `close`). The `close()` method **does not require any arguments**.

The statements that close the files should be placed **after** the loop, not inside the loop body. If you close a file inside the loop, you won't be able to use it in later iterations of the loop.

Step 9: Build Your Project and Run the Executable

Before you try to build and run your project, you'll need to place the input file in a location where the program will be able to open it. Visual Studio usually looks for this file in the project folder. If the checkbox for "Create directory for solution" is checked when you create a project, Visual Studio creates a separate folder with the same name inside the solution folder for the project; if not, the project folder will be the first folder you come to with the project name. Save the sample data file provided on the assignment page to the project folder, or create your own in a plain text editor (Notepad, TextPad, etc.).

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, select **Start Without Debugging** from the Debug menu, and observe the program's run. Open each of the four output files (in the project folder) to verify that the output written to them is correct. You will need to alter the contents of the input file and select the Start Without Debugging command several times to test the program with different values, examining the results in the output files each time.

The only output to the screen should be the `Press any key to continue` message (or an error message if the file cannot be opened). If you see any other output on screen, you've probably sent some of the output to `cout` instead of to the appropriate `ofstream` variable.

Step 10: 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 file input and output). 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_Lab07.cpp** (with your name), select it, and then click **OK**. You do not have to attach any of your input or output files. 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!