CIS 251 • C++ Programming
Laboratory Assignment 6: Reference Parameters

In this laboratory assignment you will use a `void` function with a reference parameter and a value parameter to adjust one exam score for each student based on a curve.

Background

An instructor wants a program that will add the same number of points to each exam score provided as input. Although the grade-curving logic could be written in the middle of the `main` function, this syntax might fit better in its own function.

The function accepts the existing exam score as an argument to a reference parameter and the points for the curve as an argument to a pass-by-value parameter; it then uses a simple assignment statement to modify the value of the score based on the curve.

The `main` function prompts the user for the curve and each student's exam score; after obtaining a single exam score, it passes the score and the curve value to the function. Since the score argument is passed by reference, any changes to the parameter are also applied to the argument, so the `main` function can display the new, curved score.

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_Lab06.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.

Step 3: Write the Prototype for the Other Function

On a line between the `using` directive and the header for the `main` function, write the **prototype** for a function with the following attributes (see Display 5.6 in the textbook, or your chapter 5 notes, for an example of a function prototype with a mixture of value and reference parameters):

- Return Type: `void` (the function does not return a value)
- Function Name: `curveExamScore`
- Parameters: one **reference** parameter of type `int` to receive the exam score, and one **value** parameter of type `int` to receive the points for the curve

WARNING: Don't put an ampersand on a value parameter! Only reference parameters have ampersands.

NOTE: In a function prototype, the parameter names are optional, but it may be helpful to provide these names to identify the purpose of each parameter.

Step 4: Write the Definition for the Other Function

On a new line below the closing bracket of the `main` function, begin by writing the **header** of the second function (identical to the function prototype except that **must have** parameter names and it does **not** end with a semicolon). After an opening curly bracket, write an **assignment** statement according to the following formula:

exam score = exam score + curve points

WARNING: Make sure your variable names in the assignment statement match the parameter names.

Since the function is a `void` function, it does not require a `return` statement; the assignment statement can be the only statement in the body of this function. Write a closing curly bracket to end the function.

NOTE: This function does not produce any output on the screen, so you should not write any output statements in this function.

NOTE: You have the option of writing a `return` statement in a `void` function, but it may not contain a value (`void` functions do not return a value).

Step 5: Declare Variables in the `main` Function

The following table describes the variables needed in the `main` function:

Description	Data Form	Initial Value
Curve Points	Whole number	From input
Exam Score	Whole number	From input

After the opening bracket for the `main` function, add a **declaration** for each variable.

NOTE: The variable names in different functions may be the same or different. There is no relationship between variables with the same name in different functions.

Step 6: Obtain Initial Input from the User

At this point, the `main` function needs to prompt the user (using an **output** statement followed by an **input** statement) to enter the curve to be applied to each exam score:

```
Exam Score Curve Program

How many points do you want to add to each exam grade?
```
(user enters value here)

Your `main` function should also prompt the user for the first exam score (**output** followed by **input**), letting the user know what value can be used to exit the program:

```
Enter the first exam score (-1 to exit):
```
(user enters value here)

Step 7: Call the Other Function Inside a Loop

This program requires a sentinel-controlled `while` loop. Write a **loop header** so that the loop will continue as long as the user has not entered a value of -1 for the exam score. Follow this with an opening curly bracket.

The first statement in the loop body should **call** the function. The call to this function will be the function name with the exam score variable and the curve points variable as arguments inside the parentheses and a semicolon after the closing parenthesis.

WARNING: Don't place a call to a `void` function inside an assignment statement, an output statement, or any other statement! The function will not return a value that can be used in another statement!

WARNING: Make sure the order of the arguments matches the order of the parameters!

NOTE: When calling a function, neither the return type nor the parameter types are required.

Step 8: Display the Results and Ask for More Input

After the function call (and still inside the `while` loop body), display the value of the exam score variable with a label; then, ask the user for another exam score to curve (**output** followed by **input**):

```
Exam Score after Curve:   exam score variable
Enter another exam score (-1 to exit):   (user enters value here)
```

HINT: Sentinel-controlled loops typically require an input statement right before the loop header and another input statement as the last item inside the loop body. If you forget one of these, you may run into problems.

NOTE: Do not ask the user for the curve points inside the loop. The curve points entered at the beginning of the program should be applied to every exam score entered thereafter.

Close the loop body. You may want to add an output statement with an "end of program" message to be displayed after the loop ends.

Step 9: 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, select **Start Without Debugging** from the Debug menu, and enter some sample data for each choice in the console, verifying that the output is correct for each exam score. You will need to select the Start Without Debugging command several times to test the program with different curves.

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 the statements related to functions). 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_Lab06.cpp** (with your name), select it, and then click **OK**. Verify that your file appears in the Submission section, and click the **Submit** button at the bottom of the page.

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