CIS 251 • C++ Programming
Laboratory Assignment 11: Classes

In this laboratory assignment you will learn how to use C++ syntax to develop a class definition and create objects from it. (See **Step 8** for an example of the program's execution.)

Background

The information about an item to be sold in a store includes a description of the item, the quantity in stock, and the price. You will create a class definition to store these values in `private` member variables and provide `public` member functions to set up the objects, obtain the values in the individual variables, modify the values of an existing object's variables, and obtain input and generate output for the object.

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_Lab11.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 needed for console I/O and the `string` data type, 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 Class Definition

Below the `include` and `using` directives and above the header for `main`, write the definition for a class named `StoreItem`. The following items should be declared in the `private` section of the class:

- A variable to store the item's description (text)
- A variable to store the item's quantity in stock (a whole number)
- A variable to store the item's price (a real number)

The following items should be declared in the `public` section of the class:

- A default constructor (no parameters)
- A second constructor with parameters for each member variable
- Three separate mutator member functions (one for each member variable) with a single parameter
- Three separate accessor member functions (one for each member variable)
- An input member function
- An output member function

Declare these members in your class definition, with separate sections for the `private` and `public` members.

The `private` and `public` sections of a class may be listed in any order. Additionally, a class definition can contain multiple `public` and `private` sections.

Remember that a class definition generally contains only the prototypes of the member functions. You'll write the member function definitions in your code after the `main` function.

Don't forget the semicolon at the end of the class definition!

Step 4: Write the Definitions for the Constructors

After the closing bracket for the `main` function, write the header for the default constructor (no parameters).

Don't forget to put the type qualifier (the class name) and the scope resolution operator (two colons) in the header of each member function definition written outside the class definition.

A constructor member function has no return type, and its name is the same as the class name (its syntax differs from that of other functions).

In the body of the default constructor (or in an initialization section in the header), assign the following default values for the member variables: description, "Not set"; quantity in stock, 0; and price, 10.00.

Next, write the header for the second constructor, including a parameter for each member variable (description, quantity, and price) that matches the type of that member variable.

Don't use the same names for the parameters as you used for your member variables. You'll need to be able to assign the parameters to the member variables, which is difficult if they have the same name.

In the body of this constructor, assign the description parameter to the corresponding member variable. Then, if the quantity parameter is not negative, assign it to the quantity member variable; otherwise, display an error message and assign the default value (0) to the quantity member variable as in the default constructor. Similarly, assign the price parameter to the price member variable only if the parameter is positive; otherwise, assign the default value (10.00) to the price member variable.

Step 5: Write the Definitions for the Mutators and Accessors

For each mutator member function, write the header including a parameter that matches the type of the member variable being changed. The body of each mutator should be similar to that of the constructor; the only difference is that if the quantity or price parameter values are invalid, these mutators don't assign anything to their respective member variables (the error message should indicate that the program is keeping the existing member variable's value when the parameter value is invalid).

You can also define a single mutator that allows a program to change all of the member variables at once, but separate mutators allow more flexibility.

Next, write the three accessor member function definitions. Each accessor member function should have a return type that matches the member variable being accessed, and the name of the mutator should include the name of that member variable. No parameters are required. The body of each accessor should be a `return` statement including the member variable for that accessor.

Step 6: Write the Definitions for the Input and Output Member Functions

The input member function has no parameters and does not return a value. The member function should prompt the user for a new value for each member variable and store it in a local variable declared inside the member function (one variable for each member variable). Then, these local variables should be passed as arguments to each mutator member function (called by name – no calling object required) so that the member function can verify the input values before they are assigned to the member variables.

In case the user enters multiple words for the item's description, you might want to use `getline` rather than a standard input statement for that member variable.

Don't store the user's input directly in the member variables. Instead, store each input value in a local variable; then, pass the local variable as the argument to a call to the mutator for that member variable. This way, the mutator can validate the user input and reject any invalid values.

The output member function has no parameters and does not return a value. The member function should display the value of each member variable with a label.

Don't forget to format the output (fixed notation, decimal point required, two digits after the point).

You are allowed to define the `input` and `output` member functions each with a stream reference parameter that will work either with the console or with a file, but you are not required to do so.

Step 7: Create and Use Objects in the `main` Function

Return to the `main` function. Declare two objects of the class type: use the default constructor for one, and use the second constructor for the other (create your own values for the description, quantity, and price). Invoke the output member function on each object, with each call preceded by an output statement that indicates which object is being used to call the member function (see the next page). Then, invoke the input member function on one of the two objects (preferably the one that was set up using the default constructor) and invoke the output member function for that object again. You do not have to use the accessor or mutator member functions in `main`, but you are free to do so.

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, select **Start Without Debugging** from the Debug menu. Run the program several times, mixing valid and invalid input for each member variable.

Example Run (user input in **bold**; your formatting, labeling, and spacing may vary):

```
Item 1:
Description:         Not set
Quantity In Stock:   0
Price:               10.00

Item 2:
Description:         Left Shoe
Quantity In Stock:   5
Price:               22.50

Getting more information for Item 1:
What is the item's description? Right Shoe
How many do you have in stock? 12
What is the item's price? $27.75

New Values in Item 1:
Description:         Right Shoe
Quantity In Stock:   12
Price:               27.75
```

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 classes and objects). 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_Lab11.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.