

Putting the basics together:**Simple Invoice Program**

It's finally time to write a complete program. This project will use many of the C++ features that were illustrated in the source code you were given for the first and second programming assignments. The resulting program will read a simple order form and produce a customer invoice for that order. This will require reading a relatively simple input file, doing simple calculations, and writing a simple output file. In addition, your program will have to use a C++ selection mechanism to make a decision regarding a possible discount.

The input file:

The input file for this program is named "Order.txt". A sample input file is given below. The input file begins with four lines that list the name and address of the customer. The address will always consist of three lines, the last of which will contain three tab-separated items: city name, state and zip code. You should treat the customer name and the first two address lines as strings, and you should treat each "token" on the third address line as a separate string. A blank line will follow the third address line, then a line of labels and then a line separating the labels from the order data.

Each order will specify exactly five items. Your program should be designed with that restriction in mind. Each of these order lines will have the following format:

```
<item description>      <number of units>      <unit price>
```

The item description is just a string, and will never be more than 40 characters long (useful information for formatting your output). There will be a tab character immediately after the description. There will be zero or more spaces following the tab. The number of units will be a positive integer, and will be followed by one or more spaces. The unit price will be a decimal number, with two digits following the decimal point. There will be a newline character immediately after the unit price.

Hannibal Lecter Suite B2 Chesapeake Institute Baltimore MD 21230			I widened the spacing here.
Item	Units	Price	

Homestyle Waffles, 16	1	2.99	
Cheddar Singles 12 oz	4	3.09	
Beefsteak Tomato, 4 pack	1	4.49	
Whole Kernel Corn 14.5 oz	1	0.75	
Scallops 16 oz	9	10.99	

The item descriptions are strings of characters, which may contain spaces but not tabs. Note that the appearance of tabs is somewhat unreliable, and so the perfect alignment shown above will not always be achieved in the input file. That shouldn't matter to your implementation anyway.

Some advice on handling monetary amounts:

The prices are decimal values and therefore pose some problems that we will discuss in class later. For now, you are advised to store monetary amounts as integers, not as doubles. To achieve precisely correct answers, you should read the dollar amount and the cents amount separately and then store the total price in cents. All internal calculations involving money should operate on cents and produce results that are stored as integers. When the time comes to print a monetary amount, you should compute the correct dollar amount and cents amount, as integers, and print those separated by a decimal point. If you do not do this, some of your answers may differ from the correct results in the last digit.

What must be calculated:

For each ordered item, the program must calculate the total cost of the specified number of units of that item. The program must also calculate the total cost of all the ordered items.

Some orders may qualify for a discount. Specifically, if an order totals more and \$100.00 then the customer will receive a discount of 5% of the amount of the order in excess of \$100.00. For example, if an order totaled \$125.00 then the customer would receive a discount of \$2.50 (5% of \$25.00). The discount is truncated to an integer if it doesn't come out to an exact integer. The program must determine what discount, if any, the customer should receive, and then calculate the actual total due.

The output file:

The output file must be named "Invoice.txt". An output file, which corresponds to the given input file, is shown below. As usual, the first two lines specify the programmer and the assignment, and the third is blank. Lines four through eight give the shipping address for the order, as given in the order file. Note the formatting that is used for the last address line; that's why you must read the city name, state and zip code separately. Remember, you must use exactly the same labels shown here. The ninth line separates the shipping address from body of the invoice.

The invoice begins with a line of labels and then a line of delimiters separating the labels from the table body. The table body contains one line of output for each ordered item, displaying the item description, units ordered and total price for that many units of that item. The last item data line is followed by another line of delimiters.

The next line shows the total cost of the order, before any discount. The next line shows the discount (0.00 if none), and the last line shows the grand total the customer must pay.

Programmer: Bill McQuain		
CS 1044 Summer I 2001		
Ship to:		
Name:	Hannibal Lecter	
Address:	Suite B2	
	Chesapeake Institute	
	Baltimore, MD 21230	
Item	Units	Price

Homestyle Waffles, 16	1	2.99
Cheddar Singles 12 oz	4	12.36
Beefsteak Tomato, 4 pack	1	4.49
Whole Kernel Corn 14.5 oz	1	0.75
Scallops 16 oz	9	98.91

	subtotal	119.50
	discount	0.97
	total	118.53

I widened the spacing here.

If you have read the description of how the Curator scores your program in the *Student Guide*, you know that is important that you use the same spelling and capitalization for all the labels shown above. The horizontal spacing does not effect scoring unless you combine things that should be separate or separate things that should be combined. You are free to experiment with the horizontal layout but you should try to align the columns neatly.

Implementation plan:

You should design your solution piece by piece. Begin by identifying the major tasks that have to be done, and then add detail for each of those tasks. Your implementation should be developed in the same manner. Here's a suggested order of implementation.

First, implement the input code to read the name and shipping address, and the output code to print them back out in the specified format. This will require declaring the appropriate stream variables and setting up the file streams, and declaring and using variables to store the input values. Test this code and make sure it produces correct results. Once you know this part works you should never have to worry about it again.

Second, implement the input code to read the first item record. Add code to just echo out the values you read in, to be sure you're reading them correctly. Then add the code to calculate the total price for the first item and to print it with the specified format. Test all of this code now to be sure it works. Hint: be careful of the newline character at the end of the data line. If your output appears to be double-spaced you've probably forgotten this.

Since reading the first item is (almost) logically the same as reading each of the other four, it's now easy to add the rest of the code to handle the last four items. Do that next and test to verify that you're producing the correct values in the table part of the output file.

Now add the variables and code you need to calculate the subtotal. Test that. Then add the variables and code to calculate the discount. Test that. Finally add the code to calculate the total and test that.

By implementing the program feature by feature you let yourself concentrate on solving one small part of the problem at a time. You also should only have to test one part of it at a time as well. The plan above can be broken down further. For instance, you might add code to read the description of the first item and test that, then add code to read the quantity and test that, then finally add code to read the unit price and test that.

Evaluation:

Everything that was said in the specification for Project 1 about testing still applies here. Do not waste submissions to the Curator in testing your program! There is no point in submitting your program until you have verified that it produces correct results on the sample data files that are provided. If you waste all of your submissions because you have not tested your program adequately then you will receive a low score on this assignment. You will not be given extra submissions.

Your submitted program will be assigned a score, out of 100, based upon the runtime testing performed by the Curator System. We will also be evaluating your submission of this program for documentation style and a few good coding practices. This will result in a deduction (ideally zero) that will be applied to your score from the Curator to yield your final score for this project.

Read the *Programming Standards* page on the CS 1044 website for general guidelines. You should comment your code in the same manner as the code given for the first two programming assignments. In particular:

- You should have a header comment identifying yourself, and describing what the program does.
- Every constant and variable you declare should have a comment explaining its logical significance in the program.
- Every major block of code should have a comment describing its purpose.
- Adopt a consistent indentation style and stick to it.

Your implementation should also meet the following requirements:

- Choose descriptive identifiers when you declare a variable or constant. Avoid choosing identifiers that are entirely lower-case.
- Use named constants instead of literal constants when the constant has logical significance.
- Use C++ streams for input and output, not C-style constructs.
- Use C++ string variables to hold character data, not C-style character pointers or arrays.

- You are explicitly forbidden to write any kind of loop for this program. This will make the program somewhat repetitive, and physically longer than the alternative. To some extent, that's the reason for this restriction.

Understand that the list of requirements here is not a complete repetition of the *Programming Standards* page on the course website. It is possible that requirements listed there will be applied, even if they are not listed here.

Submitting your program:

You will submit this assignment to the Curator System (read the *Student Guide*), and it will be graded automatically. Instructions for submitting, and a description of how the grading is done, are contained in the *Student Guide*.

You will be allowed up to five submissions for this assignment. Use them wisely. Test your program thoroughly before submitting it. Make sure that your program produces correct results for every sample input file posted on the course website. If you do not get a perfect score, analyze the problem carefully and test your fix with the input file returned as part of the Curator e-mail message, before submitting again. The highest score you achieve will be counted.

The *Student Guide* can be found at: <http://ei.cs.vt.edu/~eags/Curator.html>

The submission client can be found at: <http://spasm.cs.vt.edu:8080/curator/>

Pledge:

Each of your program submissions must be pledged to conform to the Honor Code requirements for this course. Specifically, you **must** include the following pledge statement in the header comment for your program:

```
// On my honor:
//
// - I have not discussed the C++ language code in my program with
//   anyone other than my instructor or the teaching assistants
//   assigned to this course.
//
// - I have not used C++ language code obtained from another student,
//   or any other unauthorized source, either modified or unmodified.
//
// - If any C++ language code or documentation used in my program
//   was obtained from another source, such as a text book or course
//   notes, that has been clearly noted with a proper citation in
//   the comments of my program.
//
// - I have not designed this program in such a way as to defeat or
//   interfere with the normal operation of the Curator System.
//
// <Student Name>
```

Failure to include this pledge in a submission is a violation of the Honor Code.