

Making the implementation modular:**Invoice Program with Functions**

For this assignment, you will modify and extend the simple invoice program you wrote for Project 4. The resulting program will still read a simple order form, with an arbitrary number of items possibly including incorrect entries as before, and produce a customer invoice for that order.

From an operational perspective, the only change is that the program will now also calculate sales tax and include it in the final total due. The big change is in the implementation. Now, you will decompose your design into separate functions, making the implementation more modular and reusable. A correct solution to Project 4 would almost satisfy all of the operational requirements of this project, without demonstrating the ability to design and implement C++ functions. Since that is really the reason for this assignment, when the TAs evaluate your submission they will reduce your score to zero if you have not broken your design into separate functions.

Functions:

You must decompose your implementation into separate functions. At minimum, you must satisfy these requirements:

- You must implement at least 5 functions, not counting `main()`. My solution has 10.
- You must implement at least one function (besides `main()`) that reads data from the input file and returns that data to the calling function using reference parameters. My solution has 2.
- You must implement at least one function (besides `main()`) that writes data to the output file. My solution has 6.
- You must implement a function that calculates and returns the discount as an `int`.
- You must implement a function that calculates and returns the sales tax as an `int`.

You may implement additional functions if you like... there is no limit on how many you can have, although it's difficult to come up with more than 10 good candidates. Each function should focus on carrying out one coherent task. Remember that it's not necessary for every calculation to be delegated to a separate function.

You should start with your solution to Project 4 and re-write it to incorporate functions. Then add a function to calculate the sales tax and the corresponding output code. Document your functions as you write them (see the Evaluations section).

This project represents a big step up from the previous one. That's why you're being given more time. Take advantage of that by not putting this off.

The input file:

The input file for this program is still named "Order.txt", and the format is unchanged from Project 4. Therefore no sample input file is included here.

Note that since the input format hasn't changed at all, you should be able to reuse the input design logic and most of the input code from your solution to Project 4. The only changes you'll need to make will relate to the function requirements stated below.

What must be calculated:

Everything that had to be calculated in Project 4 must still be calculated, according to the same rules. So, most of the design and implementation for the calculations in Project 4 can be recycled into Project 5.

The only change is that you must now also calculate a sales tax amount and add it to the final total due. The sales tax rate will be fixed at 4.5%. The sales tax is applied to the amount due after the discount, if any, has been subtracted. The sales tax amount will be truncated (not rounded) just as the discount is. So, in the sample output below, the sales tax is computed on the amount \$243.50 (why?) and works out to be \$10.95 even though the exact tax would be \$10.9575.

The output file:

The output file must still be named "Invoice.txt". The only change in the output file is that a line is added to show the computed sales tax amount. Again, that means you should be able to reuse most of the design and implementation for output in your solution to Project 4. The tail of a sample output file is shown below. Complete input/output examples will be posted on the course website.

. . .		
White Bread, 16 oz loaf	1	2.19
Buttermilk Waffles, 16	5	14.95
Red Kidney Bean	9	7.11
White Bread, 20 oz loaf	8	20.40

	subtotal	251.11
	discount	7.61
	sales tax	10.95
	total	254.45

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.
- You must include a header comment with the implementation of each function you write. The header comment should be formatted to match the sample function header posted on the *Programming Standards* page of the course website.

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 must use an input-failure controlled loop to manage the input for this program, as in Project 4.
- You may not use an array or structures in this program.
- You must not use global variables in this program. Global function declarations and global constants are OK.
- Each function parameter should be passed using an appropriate protocol. Parameters should only be passed by reference if the called function needs to modify the actual parameter. Potentially large parameters, such as strings, should be passed by constant reference instead of by value (unless they need to be passed by reference).

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.