

## A Realistic System:

## Invoice Program with Arrays and Structures

For this assignment, you will implement a simple invoice program. Each stocked item in the inventory has a unique identification code (called an SKU). The customer will enter an SKU and quantity for each item he or she wishes to order.

The *inventory file* will contain one line of data for each item in the inventory. That line will include the SKU, the item description, and the unit price for the item. The exact format is described later. Your program will read in the entire inventory file, and store the inventory data in an array. In order to do this, you must also declare a structured type of variable using the `struct` mechanism in C++. We'll call this array the inventory list.

Your program will still read a simple order form, with an arbitrary number of items possibly including incorrect entries, and produce a customer invoice for that order. Your program will read in the entire customer order, and store all those items in a second array (also using a structured type). We'll call that the order list. The customer may enter an incorrect SKU that doesn't correspond to any item in the inventory list, so you'll eventually have to check for that.

### struct variables:

Your design and implementation must treat an item as a "thing", not just as a bunch of separate variables. That means you must design and use a `struct` type. You should have an "item" type that aggregates the components of an item. In my solution, I used two different ones, one for inventory items that aggregated the SKU, description and unit price, and one for ordered items, that aggregated the SKU, description, unit price, quantity ordered, total price, and a status flag to keep track of whether the item was found in the inventory list or not. You don't have to follow that approach, but it may give you some idea of where to start.

As is usually the case, your declarations of the `struct` types should be global. Remember that you're declaring a type, not a variable here. Since these types will be used throughout the entire program, including in function parameter lists, the type declarations must be global.

### Arrays:

You must use two different arrays in this project, one to hold the inventory list and one to hold the order list. We'll guarantee that there will be no more than 100 items in the inventory list and no more than 25 items in the order list, so you can (and should) set your array dimensions accordingly.

You are required to initialize each of the arrays so that every cell holds some "dummy" data before you start reading the input files. That is a tremendous aid in debugging, especially when you add sorting later on. The arrays should be declared in `main()` and passed around as necessary. It's acceptable for the constants for the array dimensions to be global.

Be very clear on the distinction between the dimension of an array (how many records it can hold) and the usage (how many records it actually does hold at the moment). Be sure that you use the correct one in the correct place. Not doing this is a very good way to make your program produce incorrect results. Be sure you never write code that accesses either array at an illegal index. That's an excellent way to make your program blow up at runtime.

### Functions:

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

- You must implement at least 10 functions, not counting `main()`. My solution has 21.
- You must implement one or two functions to initialize the order list and the inventory list to hold dummy data before the input files are read. This is important for debugging your program.
- You must implement one function (not `main()`) that reads the inventory data from that input file and stores the data in the array of structures holding the inventory list.
- You must implement one function (not `main()`) that reads the customer order data from that input file and stores the data in the array of structures holding the order list.

- You must implement one function (not `main()`) that writes the data list of valid ordered items to the output file.
- 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 must implement a function that sorts the order list before it is printed. This function must use the Bubble Sort algorithm from the course notes, adapted to work with your data type.

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 25 good candidates. Each function should focus on carrying out one coherent task. Remember that it's not necessary for every calculation or every single output or input operation, to be delegated to a separate function. On the other hand, I wrote 8 different functions that manage parts of my output.

You should start by writing the necessary code to create and initialize an array of `struct` variables to hold the list of ordered items. You should then add a second array of `struct` variables to hold the inventory list, and add the function(s) to read in that data. It's very useful to write some "utility" functions that will take each of these arrays and just print out its contents in a nice format, even though you won't need to print the inventory list in the end.

Document your functions as you write them (see the Evaluation section).

This project represents the final step up in the course. Designing and implementing a program that uses arrays and structs takes some effort. But the final product will be a program that actually does something fairly realistic.

### The inventory list input file:

The input file containing the inventory data is named "`ItemData.txt`". A sample inventory data file is given at the end of the specification. The first two lines are just labels and should be ignored. Each remaining line contains the data for one item in the inventory:

```
<SKU><tab><description><tab><spaces><unit price><newline>
```

The SKU will always be five characters long. Descriptions will be no more than 40 characters long. The unit price will be a decimal value, as before. The unit price is guaranteed to be positive, so you don't need to check it this time. There will never be two items with the same SKU.

### The order input file:

The input file containing the order is named "`Order.txt`". A sample input file is included at the end of the specification. The file begins with the customer name, followed by three lines of address data, and followed by a header for the table of ordered items. The format of each item record will be:

```
<SKU><tab><spaces><units ordered><newline>
```

### What must be calculated:

The program must read in the inventory list data and store it in an array. It must then read in the invoice data, and store the order list in a second array. For each item in the ordered list, the inventory list must be searched for an item with a matching SKU. If one is found, then the entry in the ordered list must be updated with a description and unit price. If no match is found, that must be recorded in some way.

The subtotal must be calculated, reflecting only "good" items that were found in the inventory list. Then the discount, sales tax and final total must be calculated. The discount is computed as follows:

- Subtotals less than or equal to \$100 receive no discount.
- Subtotals between \$100.01 and \$250.00 receive a discount of 5% on the amount of the order above \$100.00.
- Subtotals greater than \$250.00 receive the 5% discount for the amount of the order between \$100.01 and \$250.00, and receive an additional 10% discount on the amount of the order greater than \$250.00.

The sales tax is 4.5% of the subtotal minus the discount (if any).

A hint: to calculate the subtotal, you must process the array of ordered items, looking at each item and updating the subtotal as needed. That's a good candidate for a function. It's also a good example of how much of this program will look. Once you've acquired the data by reading the two input files, you'll make a number of passes through each of the arrays you set up. Don't try to do everything in one pass. It's not possible. Even if it were, it's better to make extra passes if it makes the program easier to understand.

Before you print the order list to the invoice file, you must sort that list in ascending order by description. You are required to use (a modification of) the Bubble Sort algorithm (covered in the course notes). Note that you'll only print "good" items in the table in the invoice. You'll list the "bad" items separately at the end, in a different format. That looks like a good place to have two more output functions.

## The output file:

The output file must be named "Invoice.txt". The file begins with a label line, and then labeled lines containing the name and address of the person who placed the order. Following a blank line, a line of column headers and a line delimiting the beginning of the table, there is a table listing the ordered items, along with the quantity and total price for that item. The end of the table is marked by a line of delimiters.

The table is followed by four labeled lines, containing the item subtotal, the discount, the sales tax, and the total amount due. If the order included any invalid SKUs, there should be one blank line, and then a label and a list of the invalid SKUs.

See the sample output file at the end of the specification.

## Implementation plan:

This is, by far, the most complex program you'll write for this course. That makes it even more important that you approach the implementation sensibly. Unless you're awfully good, trying to write the entire program before testing any of it is a quick route to frustration.

The first thing to do is design the `struct` types you intend to use in this program. You can begin with the one to store an ordered item, and add the declaration of that type to your program. You can't really use that until you've also plugged in an array to store the list of ordered items. I'd implement the input code to use an array of `struct` variables, and print the order out exactly as it was read in to validate the input logic. This is really the biggest step in this project, learning how to use an array of `struct` variables.

Once that works, I'd add the second array (and possibly a second `struct` type for inventory items). Add the code to read the inventory data from its input file and store it in the second array. Print out the contents of that array to be sure you're reading everything correctly.

Now add the code to look up the ordered items in the inventory list. Concentrate first on handling only "good" items. Be sure you're getting everything right when you print the ordered items out (description, unit price).

Implement the calculation code needed to use the order list when calculating the subtotal. The calculations for the discount and sales tax are then relatively trivial. Add the sorting code. This should be easy. Just modify the Bubble Sort code from the course notes, and plug it in.

Finally you need to deal with "bad" items. I used a "flag" in my ordered item records to record whether the item was found in the inventory list; then I used that "flag" to make decisions later. The flag could be a `bool`, but I used an `enum` type to allow for additional choices, such as "out of stock" that were not needed for this project but could be useful later.

## Evaluation:

Everything that was said in class about testing 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 must use two arrays of `struct` variables in this program.
- You must not use global variables in this program. Global function declarations, type 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).
- Arrays should be passed by reference (which is the default for arrays) only if the called function needs to modify the actual array parameter. Otherwise, pass an array parameter by constant 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 ten 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* and other pertinent information, such as the link to the proper submit page, can be found at:

<http://www.cs.vt.edu/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.**

**Sample Inventory data file:**

SKU	Item	Price
V0011	Cut Green Bean 14.5 oz	1.05
V0013	Cut Green Beans 14.5 oz	0.75
F0001	Braeburn apples, 2 pack	1.79
F0006	Lite Peaches 15 oz	1.09
F0007	Bananas, ripe 16 oz	0.69
F0008	Strawberries 16 oz	1.49
F0011	Grapefruit, large pink	0.79
F0015	Navel Orange, large	0.69
F0016	Navel Orange, small	0.45
F0017	Cantaloupe	1.99
F0018	Honeydew	3.99
F0019	Beefsteak Tomato, 4 pack	4.49
F0020	Cherry Tomatos, 12 oz	3.99
F0022	Yellow Tomatos, 2 pack	3.99
F0023	Avocado	2.29
V0007	Whole Green Beans 14.5 oz	0.75
V0015	Whole Kernel Corn 14.5 oz	0.75
V0027	Sweet Peas 15.25 oz	0.75
V0034	Red Kidney Bean	0.79
V0040	Chopped Spinach, 10 oz	0.89
V0041	Creamed Spinach, 10 oz	1.79
M0010	Chicken Wings, 10 oz	2.99
M0017	Chicken thighs, boneless 1 lb	2.99
M0022	Choice Flank Steak 16 oz	7.99
M0023	Choice Chuck Eye Steak 12 oz	3.99
M0031	Choice Filet Mignon 8 oz	9.49
S0035	Crab Legs, Imitation 8 oz	1.98
S0023	Scallops 16 oz	10.99
S0040	Calamari, breaded rings 10 oz	5.99
S0041	Calamari, rings 16 oz	5.99
S0029	Orange Roughy fillets 8 oz	9.99
S0031	Sushi, Eel and Cucumber, 9 pc	4.95
D0001	Milk, 2% Gallon	3.05
D0002	Milk, 2% Half Gallon	2.19
D0013	American Singles 12 oz	3.09
D0014	Cheddar Singles 12 oz	3:09
D0030	Monterey Jack, 1 lb	6.99
D0032	Swiss, 1 lb	8.99
D0040	Extra Large Eggs, dozen	1.85
D0042	Jumbo Eggs, dozen	1.95
D0100	Old Fashioned Spread, 16 oz	0.89
FZ055	Blueberry Waffles, 10	2.39
FZ050	Buttermilk Waffles, 10	2.39
FZ051	Buttermilk Waffles, 16	2.99
FZ042	French Fries, 32 oz	2.29
FZ002	Chocolate Ice Cream, half gallon	5.49
FZ007	Coffee Ice Cream, half gallon	5.49
FZ004	French Vanilla Ice Cream, half gallon	5.49
BV020	Diet Pepsi, 6 pack 24 oz bottles	2.99
BV021	Diet Pepsi, 24 pack cans	5.79
BV085	Dannon Natural Spring Water, 1 liter	1.19
B0001	Whole Wheat Bread, 20 oz loaf	2.41
B0003	Honey Wheat Bread, 16 oz loaf	2.45
B0021	Bagels, Plain, 6 pack	2.39
B0022	Bagels, Natural Wheat, 6 pack	2.39
B0023	Bagels, Blueberry, 6 pack	2.39

**Sample Order file:**

Hannibal Lecter  
Suite B2  
Chesapeake Institute  
Baltimore MD 21230

Item	Units
S0041	4
V0011	5
B0031	4
D0014	7
S0023	2
V0034	9
R0001	1
F0023	4
M0010	2
M0023	6
D0001	1
D0013	2
D0030	1
D0040	1
D0035	3
FZ051	1
FZ002	1
BV020	10
B0001	1
B0021	1

**Resulting Invoice file:**

## Ship to:

Name: Hannibal Lecter  
 Address: Suite B2  
 Chesapeake Institute  
 Baltimore, MD 21230

Item	Units	Price
American Singles 12 oz	2	6.18
Avocado	4	9.16
Bagels, Plain, 6 pack	1	2.39
Buttermilk Waffles, 16	1	2.99
Calamari, rings 16 oz	4	23.96
Cheddar Singles 12 oz	7	21.63
Chicken Wings, 10 oz	2	5.98
Chocolate Ice Cream, half gallon	1	5.49
Choice Chuck Eye Steak 12 oz	6	23.94
Cut Green Bean 14.5 oz	5	5.25
Diet Pepsi, 6 pack 24 oz bottles	10	29.90
Extra Large Eggs, dozen	1	1.85
Milk, 2% Gallon	1	3.05
Monterey Jack, 1 lb	1	6.99
Red Kidney Bean	9	7.11
Scallops 16 oz	2	21.98
Whole Wheat Bread, 20 oz loaf	1	2.41
-----		
	subtotal	180.26
	discount	4.01
	sales tax	7.93
	total	184.18

The following ordered items were not found:

B0031 D0035 R0001

You only have these lines if there were "bad" items in the order file.

The SKU's are separated by whitespace; doesn't matter what kind.

However, the SKU's must be printed in alphabetic order or the Curator will take off points. There are several ways to accomplish this; any method you can think of will be acceptable.