CS 215 – Fall 2019
Project 2

Version 2 – Oct 13 – See added items in red below

Learning Objectives:

- implementation of a program using functions in a structured design.
- use of arrays of structures to solve a problem.

General Description:

You are to write an application that allows customers to order items from an inventory of products.

When the program begins, it reads the inventory list from a file, then displays the main menu including a logo that contains **your name**. Feel free to make up a name for your application.

```
+-------------------------------------------------------+
|                        AMOZON.COM                     |
|                      by Ada Byron                     |
+-------------------------------------------------------+
I - List our Inventory
O - Make an Order
L - List all Orders made
X - Exit
Enter an option: o
```

The menu options are:

- Display the inventory list
- Make an Order
- List Orders made
- Exit

The menu asks the user to enter an option. The user should be able to enter any string for the option. The first character of the string entered is upper-cased and used as the option. Examples:

Enter an option: o converted to **O** and accepted

Enter an option: only first character **o** is capitalized to **O** and accepted

```
I - List our Inventory
O - Make an Order
L - List all Orders made
X - Exit
Enter an option: what option?
Invalid option, enter I, O, L or X!
Enter an option: try again?
Invalid option, enter I, O, L or X!
Enter an option: ok this will work
```

When an invalid option is entered, the program prints `Invalid option, enter I, O, L or X!` and repeats the input until a valid option Is entered.

List Inventory:
Prints a box with the word "Products" followed by two lines of column headers as shown. A list of products is printed as shown in columns. *Product Codes* are strings 12 characters long with no spaces. *Prices* are floats, max value 999.99. *Descriptions* are strings with spaces up to 28 characters in length.

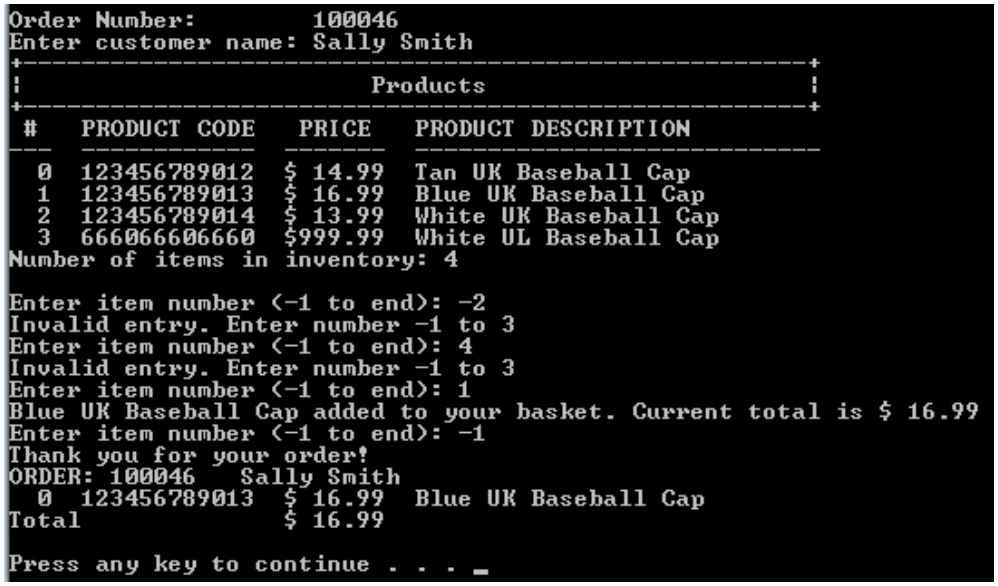
```
I - List our Inventory
O - Make an Order
L - List all Orders made
X - Exit
Enter an option: i

+-------------------------------------------------------+
|                       Products                        |
+-------------------------------------------------------+
 #   PRODUCT CODE   PRICE   PRODUCT DESCRIPTION
---  ------------  -------  ---------------------------
  0  123456789012  $ 14.99  Tan UK Baseball Cap
  1  123456789013  $ 16.99  Blue UK Baseball Cap
  2  123456789014  $ 13.99  White UK Baseball Cap
  3  666066606660  $999.99  White UL Baseball Cap
Number of items in inventory: 4
```

Make an Order:
The inventory data file contains the *Last Order Number* from the last order made the previous time the program was executed. This number is read from the file at the beginning along with inventory data.

When making an order, the *Last Order Number* is incremented and displayed on the screen. The program then asks the user to enter the name for the order. The customer name may have spaces.

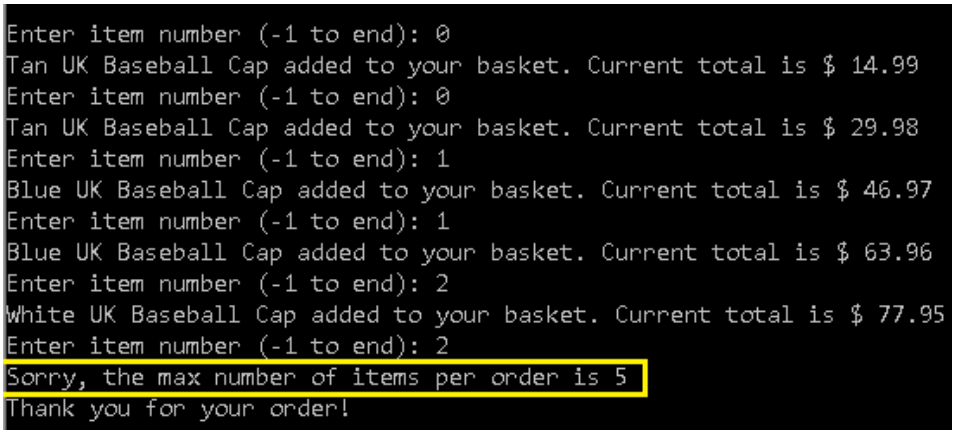
After this, the inventory is displayed from which the user may select products to order. A product is added to the order by entering its *order number.*



Note order numbers start at 0…they are really indices into the array of inventory items. The user enters -1 to indicate the order is complete. Thus the program validates the user's input to be between -1 and the number of the last product (3 in this example). You may assume the user enters an integer, but validate the value entered.

After each item is added to the order, a message is printed with the items description, along with the total cost of the order so far (sum of all prices of all items entered).

The program should also ensure that the Maximum Number of Items Per Order is not exceeded (5). When the user enters a 6th item, the program should print a message and end the order, as if the user had entered -1 to quit.



Once -1 is entered, the new order is displayed on the screen. Use a system pause to allow the user to view the order before returning to the main menu.

List Orders:
This will list all orders made so far. First a box with "Orders" is printed. For each order:

- on the first line, print the order number and customer name.
- list the items on the order. The numbers (0,1,2) are indices of the order's array, NOT the original order numbers from the inventory list. These should be printed in columns as shown. Note that the format is EXACTLY as used for the Inventory List function.
- Finally, the total amount for the order is printed, followed by a blank line.

After the last order is printed, a line with the total number of orders is printed.

Exit: nothing is printed, but the orders should be printed to an output file as detailed below.

Detailed Specifications:

Detailed specifications are written below describing how the code is to be written. You are required to follow these specifications *exactly*; unfortunately, there is little room for "creativity" when designing this project. This is how project specifications are sometimes presented to programmers in industry.

Each of the following sections describes **one function** that is to be written, including details on the function's arguments and return type. Again, not following these specifications will result in a lower grade.

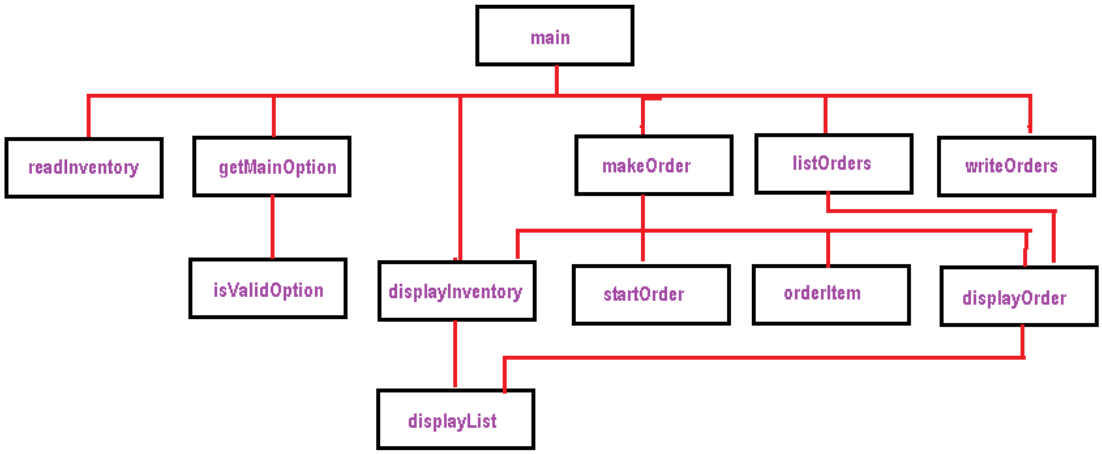
Data Structures: use the following structures in your program:

```
// describes a single item in the inventory, and a single item on an order
struct item {
	string prodCode;		// product code: length 12, no spaces
	string description;		// product description: max length 28, has spaces
	double price;			// price of the product, max 999.99
};

// describes a customer order or "basket"
const int MAX_ORDER_ITEMS = 5;
struct order {
	long   orderNumber;				// unique order number for this order
	string custName;				// customer name
	double totalPrice;				// price of all items purchased
	item items[MAX_ORDER_ITEMS];		// list of items purchased
	int numItems;					// number of items purchased (1-5)
};
```

In main, you will declare a partial array of *Items* (MAX_INV_ITEMS=10), which is called "the inventory", and a partial array of *order*s (MAX_ORDERS=7) called "the orders". These two arrays, along with other main variable's needed, will be passed into and out of other functions as detailed below.

Structure Chart:
The structure chart shows the "command structure" of the system. Each box is a function in the program. A line drawn from a higher box to a lower box indicates that the higher *invokes* the lower. Each invocation may involve data being *passed to* the lower function when it is first invoked, then (resulting) data *passed back* to the higher function.



Function List:

Name	**main()**
Given	
Modifies	
Returns	0

The main should declare the **Inventory** (a partial array of *item*), the **Orders** (a partial array of *order*), along with the *lastOrderNumber*, *menuOption* and any other local variables it may need.

It should first invoke *readIventory()*, populating the **Inventory** and the *lastOrderNumber*.

It should then repeat the main menu until the user enters the Exit code. Each iteration of the loop, it will invoke *getMainOption()* and then the chosen operation: *displayInventory(), makeOrder(), listOrders()*, or *writeOrders* (when exiting). It should end with the normal system pause.

Name	**readInventory()**
Given	
Modifies	inventory, lastOrderId
Returns	

This function has been written for you and a copy placed on the course website. Note that the design of this function is dictated by the design of the input file, which is:

- first line contains *number of inventory items* and *last order id*

Followed by a list of inventory items. For each item, there is one line of data, containing (in order):

- product id number, price, and description (possibly with spaces).

You'll need to write your own read file functions in the future, as well as a write file for this project. So study the function carefully to be sure you understand it. A sample input file is also provided on the course website.

Name	isValidOption()
Given	- an option character - a string of valid options
Modifies	
Returns	true or false

This utility function searches the given string as if it were an array. If it finds the given character, it returns **true**. If it does not find the character, it returns **false**. This can be used instead of a long list of conditional expressions. Ex: so we do NOT have to write:

```
if (c !='X' && c!='O' && c!='L' && …for ever & ever… && c !='Z')
```
instead of
```
if (!isValidOption(c,"XOL…Z"))
```

Name	getMainOption()
Given	
Modifies	
Returns	the main menu option (a char: 'I', 'O', 'L', or 'X')

Displays the "logo" with your name, followed by the menu options. Repeats a *validation loop* to force the user to enter one of the four valid options. It should invoke *isValidOption()* as part of the validation instead of a very long conditional expression with a list of or'ed or and'ed expressions. (See the first two screen shots above).

To make the program more "user friendly": the user may enter an entire string, possibly with spaces. The program should extract the first character of the string entered, convert it to upper case, and validate that value. We basically ignore anything entered after the first character. Here is code to show how to accomplish this:

```
string userInput;
char   option;
cout << "Enter an option: ";
if (cin.peek() == '\n') cin.ignore();  // just in case it's needed
getline(cin, userInput);
option = toupper(userInput[0]);
// userInput[0] is the first char of the string userInput
// toupper is a C++ function that converts a lowercase char to uppercase
```

Name	displayList()
Given	Inventory
Modifies	
Returns	

Prints the items in the given Inventory in the following format, for each inventory item:

- array index — in a width of 3, right justified, followed by 2 spaces
- product code — in a width of 12, left justified, followed by 2 spaces and a $
- item price — in a width of 6, right justified, 2 digits past the decimal point, followed by 2 spaces
- item description

Note this function demonstrates **internal reuse:** it will be invoked to print the inventory list, *and* it will be invoked to print the product list of an order (basket), since the contents and formats of both are identical. You can see this on the Structure Chart as there are two lines coming into the box for displayList().

Name	**displayInventory()**
Given	Inventory
Modifies	
Returns	

Prints the box with "Product" as shown in the screen prints above, followed by the two lines of column headers. It then invokes displayList() [do NOT repeat/copy/paste the for loop!!] and finally the "number of items" as shown above.

Name	**displayOrder()**
Given	a single basket
Modifies	
Returns	

Prints a line with "ORDER: " followed by the order number, and customer name.
It then invokes displayList(), passing it the item list from the basket. [do not write a for loop and repeat/copy/paste the code from displayList()!]. Finally it prints the total for the basket on one line, followed by a blank line. See the screen shots above for more detail.

Name	**startOrder()**
Given	
Modifies	- partial array of *Orders (baskets)* - the *LastOrderNumber*
Returns	

This function initializes a basket for the next available order. First, it increments the *LastOrderNumber* and copies it into the next available basket. It initializes the basket's number of items and total price to 0.
It displays the order number as shown above for starting a new order, then asks the user to enter their name, storing it in the basket. Finally, it increments the number of orders in the array of Orders.

Name	**orderItem()**
Given	Inventory partial array
Modifies	a single basket
Returns	true or false

This function handles the ordering of a single item in a basket.
It asks the user to enter an item number, -1 to the number of inventory items. It uses a validation loop to force the user to enter a correct answer.

If the user enters -1, indicating they do not want to order more items, the function returns **true**, indicating the "quit" option was entered.

If the user enters an item number from the inventory (assumed to already be displayed), it copies the product code, price and description from the indicated (item number) inventory item into the next available (purchased) item in the basket. The item number entered is the index into the Inventory array.

It then adds the price of the selected item to the total price for the basket. This will require some "clever" array indexing. Next it prints the description of the item just added to the basket along with the new total price for the basket (see screen shot above). It will need to increment the number of items in the basket as well. Finally it returns **false** indicating this was NOT the quit option, but instead a new item was added to the basket.

When the user enters one too many items, the program does not add the item, but instead prints "Sorry, the max number of items per order is *nn*" where *nn* is the max number of items per order. The function returns true to end the adding of items.

Name	**makeOrder()**
Given	Inventory partial array
Modifies	Orders partial array (a new element will be added) Last Order Number (will be incremented)
Returns	

This is the "main" function for making a new order. It's ultimate goal is to get information from the user on the new order, and "append" a new basket to the partial array of Orders (baskets).

First, the function should ensure that there is enough room in the Orders array to add a new order; if not, the functions should print `Sorry, we can not take more orders today.`

When there is room in the array for a new basket, it then invokes **startOrder()** and **displayItems()** to get the initial information and show the list of items to the user. Next it repeats **orderItem(),** passing it the last basket in use (the new one) until that function returns true.

Finally, it prints `Thank you for your order!` then invokes **displayOrder()** so the user can see the order just made. Use a system pause to allow the user to review this order before ending the function.

Name	**listOrders()**
Given	partial array of Orders
Modifies	
Returns	

Prints the "Orders" logo as shown above, then invokes displayOrder() on each of the Orders in use in the partial array. [do not recode/copy/paste from the displayOrder() function!!]

Finally it prints the total number of orders on one line as shown in the screen shot above, followed by a blank line.

Name	writeOrders()
Given	partial array of Orders
Modifies	
Returns	

This opens a file called "orders.txt" for writing. When the open() fails, the function should print a failure message and return from the function. When the open is successful, the function should write the Orders in the following file format:

- first line: number of orders
- for each order:
    - one line with the following, with a space between each
        - Order number
        - Number of Items
        - Total Price
        - Customer Name
    - For each item ordered:
        - one line with the following, with a space between each
            - product code
            - price
            - description
            -

Don't forget to **close()** the file when done…otherwise all of the data may not be written to the file.

Example orders.txt:

```
3
100046 4 62.96 Freddy Fender
123456789013 16.99 Blue UK Baseball Cap
123456789013 16.99 Blue UK Baseball Cap
123456789012 14.99 Tan UK Baseball Cap
123456789014 13.99 White UK Baseball Cap
100047 1 16.99 Sally Smith
123456789013 16.99 Blue UK Baseball Cap
100048 3 44.97 Bob Wildcat
123456789012 14.99 Tan UK Baseball Cap
123456789012 14.99 Tan UK Baseball Cap
123456789012 14.99 Tan UK Baseball Cap
```

```
There are orders for 3 people
Data for order 0: note 4 items
Data for item 0
Data for item 1
Data for item 2
Data for item 3
Data for order 1: note 1 items
Data for item 0
Data for order 2: note 3 items
Data for item 0
Data for item 1
Data for item 2
```

Hint: use **cout <<** instead of **f <<** until you are sure everything is OK; then change cout's to f's.

SUBMISSION:

Only submit your .cpp file in Canvas.
There is no need to submit your inventory.txt or orders.txt.