# Programming Assignment #1: Grocery Store

This assignment attempts to serve as a refresher for concepts that I hope you learned in CS 112. In this assignment, you will be building a simple storefront for a small grocery store called Jack's Mart. The store maintains its inventory in a text file. Not being very tech savvy, the owner of Jack's Mart needs you to build a program to read from and maintain the list of currently stocked items. The program will also simulate a simple shopping cart in which customers may add items.

## File Format

Your program should read a file called "grocery\_list.txt" (see included file for an example file). Each line of the file contains one grocery item. Each item contains three pieces of information: the name of the item, the item's quantity, and the item's cost. Each piece of information is separated by a colon (:). For example, "cheese:5:6" means that the item's name is "cheese", that there are 5 pieces of cheese in stock, and that each piece of cheese costs $6.

## Sample Output

Below is the desired output of your program. **Note that the number of items that I test will be different from the number of items listed in the sample output**. As such, you will either need to use the STL Vector class or a really large array to complete your assignment.

|  |
| --- |
| Welcome to Jack's Mart!  How much money do you have?: a  Invalid input. Please try again  How much money do you have?: 4  Here's what we have in stock today:  \*\*\* apples $1 (qty: 5)  \*\*\* milk $2 (qty: 3)  \*\*\* bread $1 (qty: 3)  \*\*\* candy $1 (qty: 10)  \*\*\* cheese $6 (qty: 5)  \*\*\* oranges $2 (qty: 4)  \*\*\* cherries $2 (qty: 3)  You have $4  What would you like to put into your cart: oranges  Would you like to continue shopping? (y/n): y  Here's what we have in stock today:  \*\*\* apples $1 (qty: 5)  \*\*\* milk $2 (qty: 3)  \*\*\* bread $1 (qty: 3)  \*\*\* candy $1 (qty: 10)  \*\*\* cheese $6 (qty: 5)  \*\*\* oranges $2 (qty: 3)  \*\*\* cherries $2 (qty: 3)  You have $2  What would you like to put into your cart: apples  Would you like to continue shopping? (y/n): y  Here's what we have in stock today:  \*\*\* apples $1 (qty: 4)  \*\*\* milk $2 (qty: 3)  \*\*\* bread $1 (qty: 3)  \*\*\* candy $1 (qty: 10)  \*\*\* cheese $6 (qty: 5)  \*\*\* oranges $2 (qty: 3)  \*\*\* cherries $2 (qty: 3)  You have $1  What would you like to put into your cart: milk  Sorry, you don't have enough money!  Would you like to continue shopping? (y/n): y  Here's what we have in stock today:  \*\*\* apples $1 (qty: 4)  \*\*\* milk $2 (qty: 3)  \*\*\* bread $1 (qty: 3)  \*\*\* candy $1 (qty: 10)  \*\*\* cheese $6 (qty: 5)  \*\*\* oranges $2 (qty: 3)  \*\*\* cherries $2 (qty: 3)  You have $1  What would you like to put into your cart: butter  Sorry, we're all out of butter today.  Would you like to continue shopping? (y/n): y  Here's what we have in stock today:  \*\*\* apples $1 (qty: 4)  \*\*\* milk $2 (qty: 3)  \*\*\* bread $1 (qty: 3)  \*\*\* candy $1 (qty: 10)  \*\*\* cheese $6 (qty: 5)  \*\*\* oranges $2 (qty: 3)  \*\*\* cherries $2 (qty: 3)  You have $1  What would you like to put into your cart: candy  Would you like to continue shopping? (y/n): n  Here's what you've added to your shopping cart:  \* oranges  \* apples  \* candy  You spent $4 and have $0 left over.  Thank you, come again! |

## Program Flow

Your program should operate in the following manner:

* Open the file "grocery\_list.txt".
* Read the names and quantities of items from the text file.
* Prompt the user for the amount of money that they currently have.
* Until the shopper decides to exit, do the following:
  + Display a menu with the currently stocked items, their cost, and their quantities.
  + Ask the shopper what they'd like to add to their cart.
  + If the item exists and there is sufficient quantity, add the item to their cart.
  + If the item does not exist, let the user that the item is not in stock.
  + If the item costs more than what the users has, let the user know that they have insufficient funds to make the purchase.
  + Update the list of items currently in stock to reflect the new quantities.
* After the user has decided to exit, display the current contents of the shopping cart along with how much was spent and how much money the user has left over.
* Note that the program should also write back to "grocery\_list.txt" with the updated quantity of items.

## Header Comment, and Formatting

1. Be sure to modify the file header comment at the top of your script to indicate your name, student ID, completion time, and the names of any individuals that you collaborated with on the assignment.
2. Remember to follow the basic coding style guide. For a list of basic rules, [see my website](http://adamcarter.com/teaching/cpts121/style) or examine my example files from previous assignments and labs.

# Reflection Essay

In addition to the programming tasks listed above, your submission must include an essay that reflects on your experiences with this homework. This essay must be at least 350 words long. Note that the focus of this paper should be on your reflection, ***not*** on structure (e.g. introductory paragraph, conclusion, etc.). The essay is graded on content (i.e. it shows deep though) rather than syntax (e.g. spelling) and structure. Below are some prompts that can be used to get you thinking. Feel free to use these or to make up your own.

* Describe a particular struggle that you overcame when working on this programming assignment.
* Conversely, describe an issue with your assignment that you were unable to resolve.
* Provide advice to a future student on how he or she might succeed on this assignment.
* Describe the most fun aspect of the assignment.
* Describe the most challenging aspect of the assignment.
* Describe the most difficult aspect of the assignment to understand.
* Provide any suggestions for improving the assignment in the future.

## Deliverables

You must upload your program and reflection as a ZIP file through Canvas no later than midnight on Friday, September 16, 2016. Remember that your submission must either contain a CodeBlocks or Visual Studio project file!

## Grading Criteria

Your assignment will be judged by the following criteria:

### Error Free Compile (weight: 10%)

* [0] Your program contains compiler errors.
* [10] Your program compiles without issue.

### Error Free Runtime (weight: 10%)

* [0] Your program throws a runtime exception.
* [10] Your program does not encounter any runtime exceptions.

### Money Management (weight: 10%)

* [0] Your program does not track the user's money.
* [1] Your program prompts for and keeps track of the user's money. However, it does not check for insufficient funds.
* [2] Your program prompts for and keeps track of the user's money. If the user does not have enough money to buy the desired item, output a message informing the user that they have insufficient funds.

### Error Handling (weight: 10%)

* [0] Your program does not handle novel user input
* [1] Your program handles some, but not much novel user input. Novel input is defined as unexpected user input. For example, inputting a string when an integer is expected. In cases of novel input, your program should output an error message and re-prompt the user for correct input.
* [2] Your program handles most novel input as defined in [1].
* [3] The program gracefully handles all novel user input as defined in [1].

### Accurate Inventory (weight: 20%)

* [0] The program's inventory menu does not accurately reflect the stock as listed in "grocery\_list.txt"
* [1] The program's starting inventory matches what is listed in "grocery\_list.txt"
* [2] In addition to [1], the program writes the current stock back to "grocery\_list.txt" but has a few issues.
* [3] In addition to [1], the program writes a correct, updated inventory to "grocery\_list.txt" at the end of the program.

### User Interface (weight: 20%)

* [0] Your program does not match the user interface guidelines as specified in the sample output
* [1] Your program almost matches the specified user interface guidelines
* [2] Your program completely matches the specified user interface guidelines

### Style (weight: 10%)

* [0] Your code contains more than 7 distinct stylistic errors
* [1] Your code contains between 2 and 3 distinct stylistic errors
* [2] Your code contains 0, 1, or 2 distinct stylistic errors

### Reflection (10%)

* Your reflection meets the minimum requirements as specified earlier in this document.