Author: Benjamin R. Olson
Date: April 23, 2014
Course: CS162 – Introduction to Computer Science II
Instructor: Terry Rooker

Project 2 Design: Grocery List

Discovering Requirements and Summary of Program Functionality:

We need a grocery list program where the user can add an item to a given choice of locations within a grocery store. The list of locations will be predetermined by the program. For every item the user wants to add to the grocery list, he/she will choose one of the displayed store locations and then provide the name of the item for that location. The user should also be able to delete items from the grocery list. Therefore, the user must have some way of selecting an item to delete from the grocery list. Regarding this, the grocery list will be displayed to the user with the items listed by their location, so that all the items for a certain location are listed together. The user can then choose, by entering a number, the item to delete based on that item's ID# displayed.

Data to Represent:

A grocery list of items, where each item belongs to a store location and has a name and quantity (how many of that item to buy). The store will have ten aisles: 1 – Fresh Produce, 2 – Dairy/Meat, 3 – Canned Goods, 4 – Cereal/Morning, 5 – Baking, 6 – Pasta/Sauces/Etc., 7 – Oils/Wine, 8 – Bread/Condiments, 9 – Kitchenware, 10 – Paper Goods.

How this data will be represented:

Each store aisle will be a separate vector (10 total vectors). Each grocery list item will be a struct with two members, one for the item's name and one for its quantity. Each struct will belong to one of the 10 vectors. All of this will be defined inside a single “items” class, and the object instantiated from the class will represent the grocery list. Also within the class will be a string array of store aisle names, and this array will act as a bridge to logically connect a location name to its corresponding number in a switch statement which accesses the correct vector. All of this data will be private to the class, with getter and setter methods so that the user can interact with the data.

How the data will be worked with:

The object holding the data will not be responsible for screening user input, but functions outside the object will ensure the object is being used correctly and provide an accurate way to represent the object to the user, add an item name and quantity to the correct struct, and delete an item correctly. Here is the division of functions, some belonging to the item class, and some outside the item class:

Highest level functions in main menu (switch within do-while loop), for user interaction:

displayList: Takes a pointer to an item object (grocery list), displays an ID#, name, and quantity for each item struct in the item object.

addToList: uses inputLocation, inputName, inputQuantity, and object.addItem

deleteFromList: uses inputItemDeletion and object.deleteItem

Functions to get user input (all using getline):

inputItemDeletion: Prompts the user to enter the ID# for the item to delete.

inputLocation: Prompts the user to enter a number that represents an aisle of the store

*Input constraint 1: must be an integer 1 through 10 (one of the 10 aisles).

inputName: Prompts the user to enter the name of the item to be purchased from that aisle.

*Input constraint 2: must be no longer than 20 characters

inputQuantity: Prompts the user to enter the quantity of that item.

*Input constraint 3: must be an integer 1 through 1000 (first check # of digits)

*Input constraint helper functions:

1 – isAisleNumber, 2 – isValidItemName, 3 – isValidQuantity

Items class functions (return data values and set data from setter arguments):

mutator functions:

addItem: Takes store location as an integer, item name as a string, and item quantity as an integer, and creates a struct for the item (with its name and quantity) and adds it to the vector.

deleteItem: Takes store location as an integer, item number as an integer, and deletes the struct at the item number position in the vector that corresponds to the given store location.

accessor functions:

getAisle: Takes an integer 1 through 10, representing the aisle, and returns a pointer to the corresponding vector.

getAisleName: Takes an integer 1 through 10, representing the aisle, and returns a string which is the name of that aisle.

Process of Implementation:

First priority is the data class, "Items." The class will be written as a separate file and then successfully instantiated in a main() function of another file, with input tests hard-coded.

Once the class is tested and complete, input constraint functions will be implemented and tested with manual, user input. Functions: isAisleNumber, isValidItemName, isValidQuantity.

Then the functions to get user input will be implemented and tested.

Finally, the highest level functions – displayList, addToList, and deleteFromList will be implemented and tested., because their definitions will require the use of several other functions that must be already implemented at this point.