Project 2, Store Manager and Order Filler

Due: Please check due date on BlackBoard

Objectives

Use of STL Vectors, Stacks and Queues will all be covered in this project. This second Data Structures project may not use the structures that are best for printing the state of the data, but provide experience using STL and the features particular to each data structure in a way that reflects how they may be used in broader applications.

Introduction

A store manager needs to build an organized store for his inventory and then use that store to fill customer orders. This manager has a unique style of managing his store and taking orders. He/she categorizes the inventory by type and fills orders by the types of items that are requested. If the store runs out of a certain type of item the customer can choose to either accept the part of their order that can be filled or have their order cancelled.

You will be given two data files (examples given) that you will need to use to build your store and your orders and the results of filling the orders using the store will be written to a results file.

Your Driver should include a greeting function named printGreeting(), that will print YOUR NAME AND SECTION.

More details will be described below.

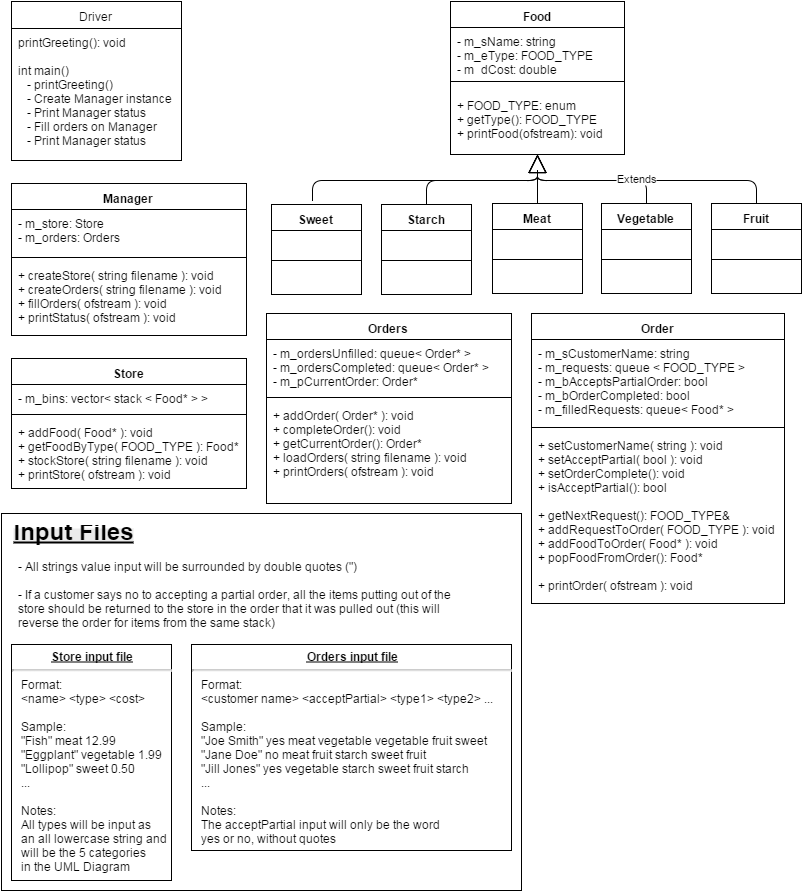
The project should be built to contain several major classes, detailed in the UML Diagram below. The UML shows classes, functions and member variables that must be present in your project but additional functions or member variables may be added that are not present in the UML Diagram.

The store should be built to contain a vector of stacks where each stack represents a FOOD\_TYPE. Please not that FOOD\_TYPE should be defined as an enum within Food that can be used as a variable type throughout the project ([https://youtu.be/RNTFqIwqTPk](https://www.google.com/url?q=https://youtu.be/RNTFqIwqTPk&sa=D&ust=1571577656285000&usg=AOvVaw3b--J6tIk50xDgLAP6q7vp) for enum more info on enums as parameters). Food is a base class from which each of the food types will derive. Each inherited food object should know its own type. Remember that some stack may never get populated if the store doesn’t have any food of that type. The store should know how to build itself from a file containing the store input data.

The orders class should maintain the orders that will be requested from the store. Each order will have a customer name, a notation of if the customer accepts partial orders, and a list of food types. If a customer accepts partial orders then every item on their order that can be filled should be. If a customer does not accept partial orders then the items should be added back into the store in the order that they were removed and filling the order should be stopped as soon as it is discovered that the order will fail to be filled. Returning items to the stack will result in two items from the same stack reversing order when placed back into the stack.

Loading the store and the orders and handling the filling of the orders should happen in a class called Manager. The manager should contain one store and one set of orders and should print the results of the orders to a file.

Project UML Diagram



\* Example input and output files are provided for you. Consider each set of files. The location of the input file will be in the same directory as your Driver.cpp file. Sample files may not address all cases.

Output File (results.txt)

The results of the orders that are filled should be stored in a file called results.txt. This file will be populated entirely by calls to print functions within the classes. Data should always remain in the type of data structure in which they are stored. Below is the format expected out of each print function listed in the UML Diagram.

Store::printStore()

Format for each Food type:

<Food Type>: <count> <item1Name> <item2Name> <item3Name> …

Example:

---------- Store ----------

Meat: 4 Chicken Breast, Chicken Legs, Steak, Fish

Fruit: 3 Nectarine, Kiwi, Orange

Vegtable: 3 Corn, Broccoli, Eggplant

Starch: 4 Mac & Cheese, Rigatoni, Spagetti, Potato

Sweet: 2 Ice Cream, Lollipop

Order::printOrder()

Format:

If order is not complete:

<Customer Name>

Requests: <FoodType> <FoodType> <FoodType> …

-----

If order is completed with all items requested:

<Customer Name>

Items:

<item1Name> $<cost (2 decimal places)

<item2Name> $<cost (2 decimal places)

…

\*\*\*Total: $<sum of all items (2 decimal places)

-----

If order is partially completed:

<Customer Name> (Partial)

Unfilled Requests: <FoodType1> <FoodType2> …

Items:

<item1Name> $<cost (2 decimal places)

<item2Name> $<cost (2 decimal places)

…

\*\*\*Total: $<sum of all items (2 decimal places)

-----

If order was not completed because customer does not accept partial orders:

<Customer Name> - Unable to complete order

Unfilled Requests: <FoodType> <FoodType> <FoodType> …

-----

Example:

If order is not complete:

Joe Smith

Requests: Meat Vegetable Vegetable Fruit Sweet

-----

If order is completed with all items requested:

Joe Smith

Items:

Chicken Breast $2.50

Corn $0.25

Broccoli $1.49

Nectarine $0.75

Ice Cream $3.99

\*\*\*Total: $8.98

-----

If order is partially completed:

Jill Jones (Partial)

Unfilled Requests: Sweet Fruit

Items:

Eggplant $1.99

Rigatoni $2.99

Spaghetti $0.89

\*\*\*Total: $5.87

-----

If order was not completed because customer does not accept partial orders:

Jane Doe - Unable to complete order

Unfilled Requests: Sweet Meat Fruit Starch Sweet Fruit

-----

Orders::printOrders()

Format:

---------- Orders -----------

<Order::printOrder() for each order>

Example:

---------- Orders -----------

Joe Smith

Requests: Meat vegetable vegetable Fruit Sweet

-----

Jane Doe

Requests: Meat Fruit Starch Sweet Fruit Sweet

-----

Manager::PrintStatus()

Pseudo Code:

Store::printStore();

Orders::printOrders();

Running and Compiling Requirements

Please compile and complete your work on your own directory on GL. I have had several students that are compiling on mine (or slupoli/pub/cs341...). This takes up a lot of space and my disk quota. My directory is only to be used a repository for your completed files. If this is abused, your directory will be closed automatically by GL.

What to Submit

Read the [course project submission procedures](http://www.csee.umbc.edu/courses/undergraduate/341/fall13/projects/submission.shtml). *Submission closes by script immediately after 9pm.* Submit well before the 8:59pm deadline, because 9:00:01 might already be late (the script takes only a few seconds to run).

You should copy over all of your code under the src directory. You must also supply a Makefile. Do NOT submit your own test data files. **Any unnecessary files submitted will be considered for a deduction.**

Make sure that your code is in the ~/cs341proj/proj2/src directory and not in any other subdirectory of ~/cs341proj/proj2/. In particular, the following Unix commands should work.

cd ~/cs341proj/proj2/src

make

make run STORE=something? ORDERS=something?

make clean

The command “make run” should simply run the project that compiled successfully.

Don't forget the Project Submission requirements shown online!! One hint, **after you submit**, if you type:

ls ~/cs341proj/proj2/

and you see a bunch of .cpp and .h files, this is WRONG. You should see:

src

instead. The C++ programs must be in directories under the src directory. Your submissions will be compiled by a script. The script will go to your proj1 directory and run your makefile. This is required. You will be severely penalized if you do not follow the submission instructions.