Project 1 2110

Jake Deeks-Jad0420(11200287)

Project Description

The first project assigned for the class Computer Foundations 2 was revolving around the data structure bit vectors. We were assigned to help a struggling restaurant with managing their inventory. The program you created will need to be capable of reading the list of ingredients for each dish, storing the ingredients list for each dish into a bit vector, and compute data analytics on the ingredients list for each dish to help the restaurant determine where it will need to make improvements. As well as reading all the ingredients of each dish, you will have to be able to read the list of ingredients the restaurant has available to use for the dishes and determine if the creation of the dish is even possible with their current inventory. If the dish cannot be created, you will need to inform the restaurant/User that the dish is not able to be created due to whatever ingredient is not available.

Data Structures

The Data Structures used in the project all revolves around vectors. With the actual bit vector representation being an actual vector of Booleans to represent the 0/1 representation required by the instructions. What I used to store all the data needed for all of the ingredients and dish names was a class Menu. The class Menu held the private data types of an integer size that was used to hold the size of the vectors, a vector of strings to hold the restaurant's available inventory, a string which holds the dish name, and a vector of Booleans to hold the 0/1 representation of the ingredients in the dish compared to the inventory. There was really no other data structures that were used within my project.

I used a lot of vectors throughout the whole project to store the data in local variables to manipulate the data easier before storing the data into the correct versions in the class. How I managed the data collected from reading in was to create a vector of my class Menu with each position in the vector holding a dish's entire information, the dish name and ingredient list for the dish, so all the information of a dish would be at the same place and not mistaken for another dish's information. For the comparisons of ingredients used at least once, exactly once, more than once, and not used at all, they were made a class object and were made to store the resulting bit vectors when using set operation functions I created to get the desired outcome for each.

System Functionality

The system Functionality of my program is actually quite simple in how it reads in and manipulates the data. The first thing that is done in my program is welcome the user with a welcome function and a load function, which all the work is done in the load function. Inside the load function, the first thing done is asking the user to input the file which all the data would be read from. This is done by using the operations of the library fstream to read in the file. Next, the first line of the file is grabbed using get line, which if file is correct it is the inventory list, to be used as the restaurants inventory for the rest of the project. You would have to be careful with how you store this because get line will get the whole line when you need each ingredient in the comma delimited list to be a separate variable. This is done by using the function split which turns the string into a stream and then proceeds to go through the stream and cutting the stream at each variable by the delimiter comma with each cut being put into a vector and then it will return the vector's address back to the function

call where you would set the return value to another vector you created to be able to be used continuing on in the project. This method of splitting the comma delimited list when reading in the ingredients is used every time I need to read in the ingredients that has a comma delimited list for the ingredients. All the data read in from the input file is first stored into a local variable before being pushed into a vector of class when all the data for a single dish is collected.

The print out of the data collected is formatted with several different steps to format it correctly. The restaurants inventory is the first thing printed out so I just had a loop that printed out each position of the local vector of strings that held the information. To print the dish's information, I first printed out the dish's name that was stored in the class of that position. How the bit vector is outputted is through a class function that I created called PrintBit which prints the bit vector with the 0/1 representation. You would use the same position of the vector you used for the name and just tag on the PrintBit to print out the right data. You would use the same ideology for printing out if the ingredients are more than one, at least one, exactly one, and not used.

The Major functions ,that are not a part of my class, used within my program are the set operations(Union, intersection, etc.), split function(Separates a comma delimited list), and my load function (which is where most of my code resides). The set operation functions are pretty self-explanatory in that I used the templates of pseudocode provided in the note slides. The split function was explained in the first paragraph on how it works. Lastly, the load function somehow ended up containing my whole code. It wasn't my original intention but ended up for the better due to the fact that I didn't need to keep jumping my data back and forth through functions for computations. The whole purpose for load turned into loading the input file to read and then proceeding to run all the computations and finally printing out the results of said computations.

My Class functions are represented by the mutators and accessors needed to promote encapsulation of the private data types along with the function of insert, remove, and contain. The functions insert, remove, and contain were specifically made for the bit vector as the way to add new items in a bit vector and so on. Each of the three functions work by the same logic of searching through the bit vector and trying to see if the passed string provided matches up with any of the positions in the bit vector and proceeding to do different operations if it does exist within the bit vector. Such as, changing the value from false to true for insert, or the opposite with true to false for the remove function. While the contain function, it doesn't need to replace the value at the position found but just return a value of true if was found and false if it wasn't found back to the user.