Nathan Leyden

Cocktail App Midterm Writeup

When we started this project, everyone in the group, including myself had high ambitions for this project. We had ideas about creating your own drinks, being able to search for drinks based on what you had in your cabinet, counting the number of visits each drink had, calculating a price of what it would cost to get the remaining items for a drink. However, as we got further into the semester, we all realized we had to scrap many of these ideas, as the end of the semester was coming up fast, and we were just getting the base of the program done.

In the beginning, when we were still in the planning phase, we had decided that the best option was to use a sequence diagram to represent our code. While I personally didn't help too much with this, and since this was the UML diagram that would best represent our code, I worked on splitting up our classes, as we had too many classes that were doing too many things and had more people than we had classes to work on. When we got past the planning phase and down to the coding phase, I was the one who decided to work on an Iterator, to be able to connect multiple types of containers. Within the second sprint, I knew I was in over my head, as even after looking up page after page on what an Iterator was and how to program one, I just kept getting more and more lost, and anytime I thought I had it figured out, something would come up that would send me back into researching. By the end of the sprint, I had decided, after bringing it up with the rest of the group to scrap the idea and move on to the Search class, which will be used for many different user cases.

When deciding on how to write the code for the Search class, I thought primarily what different use cases it could apply to. It could be used for simple searching for a drink based on the name of the drink, a drink could be searched based on the ingredients in the drink, a user could search for drinks that are in their favorites list and based on what ingredients are in their cabinet as well. Many of the user cases were able to be lumped into a simple search by name and search by ingredient. With what I was eventually planning to do, I had to go through and research multiple different templates and ideas. The first of which was Maps, which is a container that let you search for an object by a string rather than a number. My biggest help with that was cplusplus.com which gave me much of the information I needed to get started with maps. This was enough to get my first version of search, which simply returned any drink that exactly matched what was searched for.

The next and most difficult version of Search was attempting to get the map to return any drink that partially matched the name of the drink. After failing to create a method on my own, I turned to stackoverflow.com and found a similar question that was answered by Branko Dimitrijevic. His code ended up using the built in iterator in C++, and while I didn’t want to see that at first after how much time I wasted in the first two sprints, I experimented with it and was able to figure out how the built in iterator works and helped me understand the concept behind them, although I’m still unsure on making one myself. After using his code as a base and making a few adjustments, I got the output I was looking for, although it only returned one value.

From there I developed the final version of Search, which could return any number of values that partially matched what the user was searching for. All that required was changing an if statement to a for loop and creating a secondary map which stored any drinks that matched the name of the drink, which would be emptied when a new search was called. This required learning about pairs, but it was a simple thing to learn, especially with cplusplus.com. As for searching by ingredient, it also required only a few changes, primarily a do-while loop to check over all ingredients in a drink until all drinks have been checked or the ingredient was found.

After I felt confident about the Search class, I began to learn QT, however, I fell behind everyone else, and after realizing that some of the UI elements can be repurposed for different classes, I began to look for other ways to help the group, not wanting to simply wait until the UI was done. Now, as we come upon the final stretch for the project, I am working towards setting up the second half of the PowerPoint presentation for the group, on top of finding any sort of image needed for the drinks and cleaning them up with Photoshop if necessary.

Links;

Maps reference: http://www.cplusplus.com/reference/map/map/

Pair reference: http://www.cplusplus.com/reference/utility/pair/pair/

Partial match basis: https://stackoverflow.com/a/9350066

Original version of the searchDrink function:

```
Drink Search::searchDrinks(string name) //Searches through a map of Drinks returning the
first Drink that matches search, or nothing if it's not found.
{
	if (drinkStorage.count(name) == true)
	{
		return drinkStorage[name];
	}
	else
		;
}
```

Updated version of the searchDrink function:

```
//this function is given a map containing all drinks, a name of a drink to search for and
will return anything that partially or completely matches
//what the user was searching for. This is returned by reference in a map of drinks.
void Search::searchDrink(DrinkMap drinkStorage, const string& search, DrinkMap&
foundDrink)
{
	DrinkMap::iterator i; //creates an iterator to easily access the key and value in
drinkStorage.
	for (i = drinkStorage.begin(); i != drinkStorage.end(); i++) //loop that goes
through the begining of drinkStorage to the end of drinkStorage.
	{
		const string& key = i->first; //gets the key of the current point in
drinkStorage.  used to compare the key with the user input
		if (key.compare(0, search.size(), search, 0, search.size()) == 0)
//compares the user input with the current key. explenation of what is happening below

					    //if the first search.size() values in key are the same as the
first search.size() values in search, then...
		{
			foundDrink.insert(DrinkPair(i->first, i->second)); //...insert the
key and drink of the current point on map drinkStorage into the map foundDrink
		}
	}
}
```