FDS Project

Name: My Restaurant

Gabriel KUKA 1/12/2018

Purpose

The purpose of this project is to demonstrate how a kitchen in a restaurant manages its orders. The orders include within them food and drinks along with other information like prices, positions, and quantity.

The main idea is to make use of polymorphism, inheritance, abstraction, method overloading, and different algorithms. The program contains two type of algorithms: Boyer-Moore algorithm and insertion sort algorithm.

Overall view

All of the orders are managed by a list that acts like a queue. This list contains methods that perform actions on the orders. These actions include:

* Displaying orders
* Adding orders
* Removing orders
* Searching orders
* Sorting orders

Orders are represented by class “Order”. The class stores important data like price, table number, server name, number of guests, food, and drinks. This class inherits from class “Item”. “Item” is an abstract class with two virtual methods.

Other abstract classes include “Food” and “Drinks”. These are base classes for their subtypes.

Classes that inherit from “Food”:

* Appetizers
* MainCourse
* Desserts

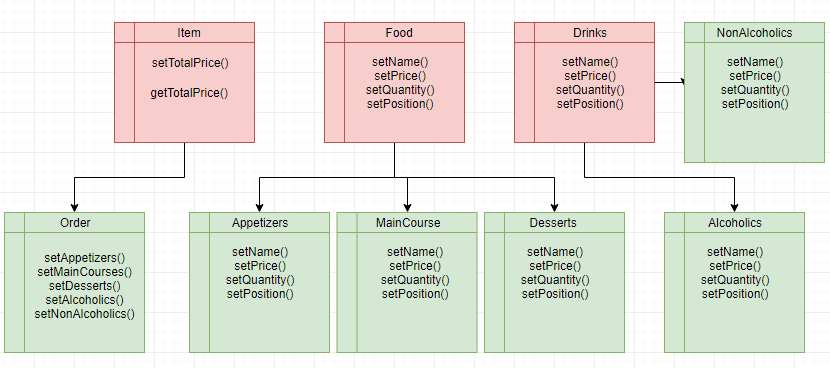
Classes that inherit from “Drinks”:

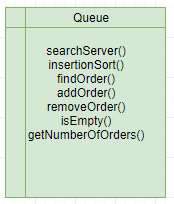
* Alcoholic
* NonAlcoholic

With that said, there are three types of food dishes and two types of drinks. Each of these types is represented by its own class which inherits either “Food” or “Drinks”. These subclasses store data like price, position, discount (if any), quantity, and the name of the dish or drink.

In the main file (where the “main” function is located), there are functions that the server (or the chef) needs to enter different commands. These commands include: adding and order, removing order, finding order, and listing the commands.

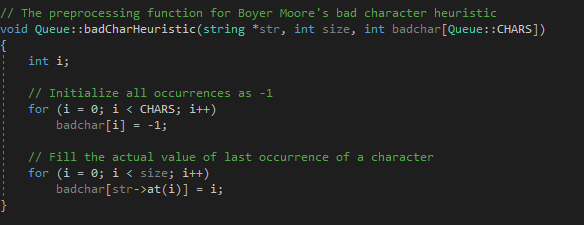
Graphical presentation

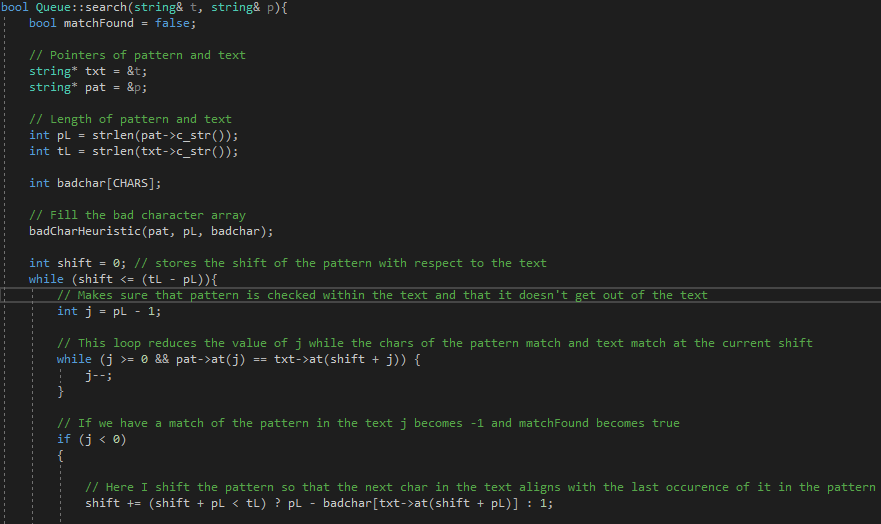


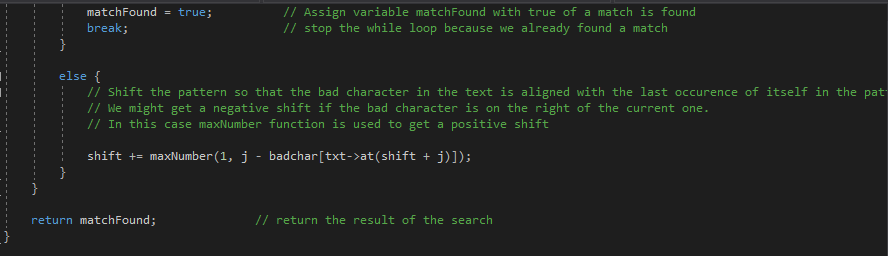


Boyer-Moore Algorithm

The program contains an algorithm that searches if a particular server exists in the list of orders. If so, the algorithm will display all the orders assigned to that server. The logic is composed of two functions: one is the preprocessing and the other is the search.





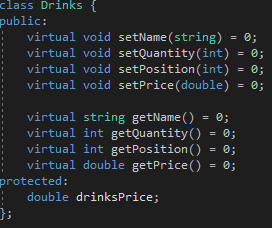
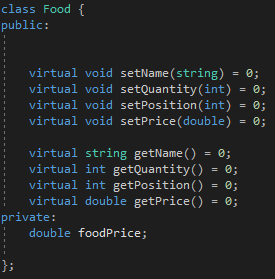


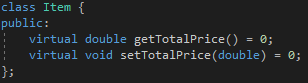
Insertion sort

The insertion sort algorithm will sort the order based on their price. The most expensive order is displayed the first continuing with the less expensive ones. The algorithm only **displays** the orders in descending order. It does not change their orders in the list.



Abstraction





The functions of these classes are initialized with the value 0 and declared as virtual. These functions are also called as pure virtual. The classes are called abstract classes. The implementation of these functions is added in the classes that inherit them.

Method overloading





Some functions have the same purpose but they need different parameters to achieve it. In that sense, in the “Queue” class, the program contains three functions called “displayOrders”, which goal is to display the orders but they have different kind and number of parameters. The first one prints all of the orders in the list. The second one prints orders that are contained in a vector and the third one prints all the orders that are assigned to a specific server (whose value is saved in the string parameter).

The same logic goes for the add order functions. They both have the same goal. The first one creates the order by adding data through parameters, whereas the second one simply adds the order which has the data within it.

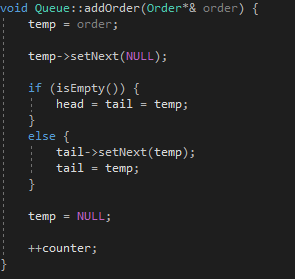
Queue

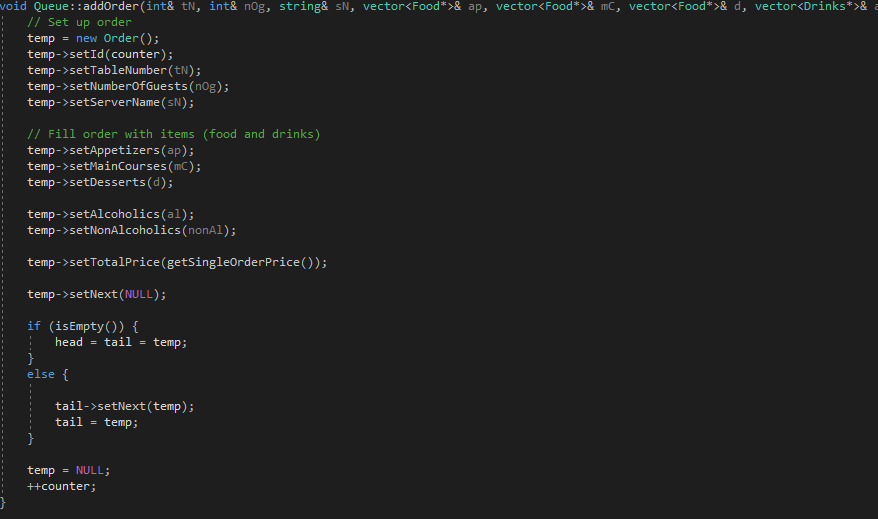
Class “Queue” has 3 pointers which help to location the first, the last, and the next order in the list. These pointers are: head, tail, and next. The main goals of the class are: adding an order, removing an order, sorting orders, and finding an order.

Adding orders

The function has one parameter: the address of the pointer that points to the order. The pointer will be assigned to variable “temp”.

First, we check of the list is empty. If so, head and tail will have the same value (since this is the first and the last order). Otherwise, the program adds a new order after the tail by using the “setNext” function and the new order will be the new tail of the list. Pointer “temp” is then assigned to null.

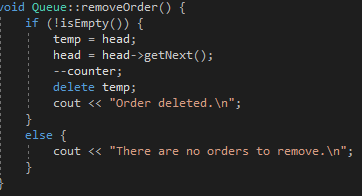




The other functions that adds a new order has the same logic. Different from the first one, here we set the data to the order and then add it to the list.

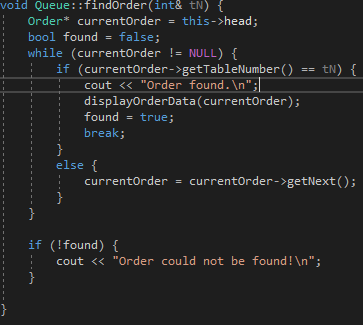
Removing an order

The function first checks whether the list is empty or not. If not, then the oldest order is removed (the one added first) and pointer “head” is assigned to the next order in line. After that we call the destructor of class “Order” to eliminate memory leak.



Finding an order

The function has one parameter: the address of an int variable. The variable stores the table number. This way, we can find a specific order by its table number. The logic is simple. The program uses a while loop to check every order for its table number. If the table number of the order being checked matches with the value of the int variable that means that we have found our order. The program breaks the loop and prints the order on the screen. If the table number is not found, it means that the particular table is empty and has no clients.

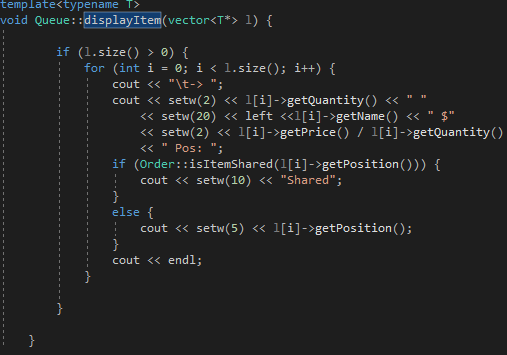


Sorting orders

The algorithm is explained above in the algorithms section.

Templates

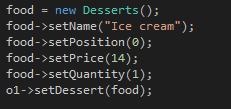
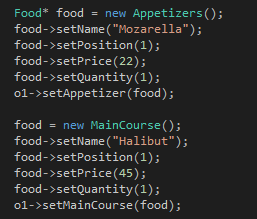
Functions which return type or the datatype of the parameters it has is undefined are called template functions. When, at compile time, you don’t know the datatype or the return type of the function than, you create a template function. The program contains multiple functions of similar type. One of them is “displayItem”.



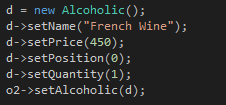
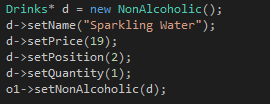
The function prints the data of type of food or drink. For example, if we are printing an appetizer, the program prints its price, quantity, position, and name. The problem is that we don’t know if it is a type of food or drink. In that sense, we declare a function as template so that when we enter the argument it does not matter whether it is a drink or a type of food.

Polymorphism

Polymorphism in programming means using an object that can take many forms. In this case, we have classes like “Food” and “Drinks”. When we want to create an object of type Appetizer or Desserts, we can use the same class as datatype (“Food” in this case) and proceed with the logic. This makes thing very easy because this way we don’t need to define many functions as template. A simple example is shown below:



Here, I create one object (food) of with datatype food and I assign it as either appetizer, main course or dessert.



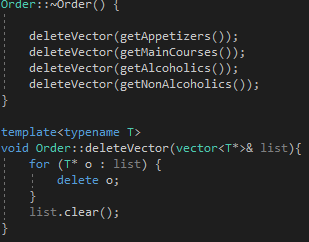
Here, the logic is the same with class “Drink”.

MyRestaurant.cpp

This is the main file in the program (which contains the main function). Here, I have created a function which checks for the user input. Based on that input a specific action is performed (like adding, deleting, sorting orders etc.). If the input doesn’t match with any of the commands, then the program prints unknown command. All of these commands are stored in an array of data. The check is performed with a switch statement. It checks the index of the command. If the input matches with a particular element in this array, then an action is performed. Each of these actions is performed by a function that calls inner functions of class “Queue” or class “Order”.

Memory leak

Since C++ doesn’t have a garbage collector like Java or C#, the programmer must handle deleting unnecessary data. These data can be pointers, which in this project are used everywhere. After the program finishes its execution, the pointers still remain in the memory. Other programs that want to allocate memory for their variables might find it difficult to find empty slots in the memory. To make things more efficient, we delete these pointers from the memory after we are done with the program. To delete the pointers, we need to call the destructors of classes like “Queue” or “Order”.



Here, we have the destructor that clears pointers saved in vectors. We delete lists of appetizers, main courses, and all types of drinks. The reason why I use vectors is that one order might contain multiple kinds of drinks of food. Since the pointers are stored in vectors, I have created a template function that uses a for loop to clear the pointers. After deleting pointers, the program clears the list.

Program in action

