|  |
| --- |
|  |
|  |
|  |
|  |
|  |

CMP2801M

Assessment 1

Table of Contents

[1 Application Overview 3](#_Toc61608277)

[2 Implemented functionality 3](#_Toc61608278)

[2.1 Additional Tasks 3](#_Toc61608279)

[3 Test cases 4](#_Toc61608280)

# Application Overview

My application is a console based takeaway system. It is designed to read in a custom menu in a .csv format, show it to the user in a user-friendly manner, and allow them to create, edit, and finally receive a receipt for an order. This system makes use of multiple classes, inheritance, and manual pointers to move data about menu items efficiently without the need for extra code or the copying of memory chunks that can be addressed by reference instead.

# Implemented functionality

Within my Item.h header file, Appetiser, MainCourse, and Beverage are all derived from the Item class. MainCourse has no function or variables for itself, and inherits everything from the Item class.  
Similarly, both Order.cpp and Menu.cpp inherit from the ItemList abstract class, and both respectively use the “items” vector. They both also have their own “toString()” functions that provide functionality to the pure virtual “toString()” function in the ItemList class.

The Item class differs from the brief as it includes all attributes that an Item could have. This means it also has “shareable”, “abv” etc. included in the main parent class, as well as the child classes. I implemented this in order to be able to access these attributes of an item stored in the “Items” vector. Since this list is a list of Items\*, I can only read back what variables are included in the parent class of Item, regardless of which child type is inputted. By having all the possible attributes of an Item in the Item class, I am able to read them from the Items vector.

I also didn’t make use of the “isAlcoholic()” function of Beverage, opting instead to check if the ABV value was greater than 0 each time a beverage is sent to the console/written to a file.

Within Takeaway.cpp, I added some error checking to prevent the user from crashing the system with invalid parameters. The “add” and “remove” commands, which both support multiple items at once, have a check to make sure the item to be added/removed is within the list of available items. If it isn’t, it warns the user and continues. The “checkout” command also takes any other command aside from “y” as meaning that the user wants to make changes to their order. This prevents any errors from arising regarding invalid inputs when asking for user confirmation, as any invalid input will simply continue the program.

## Additional Tasks

I undertook two of the three additional tasks, namely adding the functionality for add and remove to work with multiple items at once, as well as making “Order.calculateTotal()” calculate the “2-4-1” savings based on the cheapest 2-4-1 item bought.

Within Takeaway.cpp, under the add and remove commands, I added a for loop that runs through the “parameters” vector after the first index. I then use each parameter after the initial command (index 0, which would be “add”, for instance) and add that value to the order. The same functionality is used for “remove” as well. This way the command could take a theoretically finite list of items to add or remove at the same time.

The Order class had some extra variables added to allow for easier twoForOnce price calculation and printing. “twoForOnceTotal” takes the total price of the 2-4-1 items, minus any savings that you have. “twoForOneFlag” is simply a bool to let other classes know whether or not any savings from 2-4-1 items have occurred. The function iterates through all the items added to the “items” vector in order (the order list) and checks if they’re a 2-4-1 item. If they aren’t, the price simply gets added to the total. If they are, it increments the “twoForOneItems” integer and adds the price to a “twoForOnePrices” vector. It then checks to see if “twoForOneItems” is greater than 1. If it is, that means there are two 2-4-1 items currently buffered, and it sets the “twoForOnceFlag” true to indicate that savings will be made. It then compares the two prices in the “twoForOncePrices” vector and only adds the smallest of each to the total.

# Test cases

Briefly describe how you tested your program. For each function (if applicable) and the program demonstrate some test cases in a table form and indicate whether the tests were passed or not.

Please note that this is for us to evaluate how you approach software testing. The test cases we evaluate in the evaluation stage will be different and you will receive a mark will be affected by the number of test cases your program passes from within the cases we define. Below is an example to get you started:

Throughout the creation of my program, I have constantly been white-box testing by sending specific inputs I know should return a desired output to functions in my code and evaluating their performance. Whilst I could have used unit testing to accomplish this, I instead opted for what I believe to be a quicker method of creating a temporary main function and printing to console any outputs I test. For instance, in my unused “test.cpp”, I add multiple items to my order and print the result of “order.toString()” to the console. This was testing during the early stages of my code before Takeaway.cpp was fully implemented.

Once my

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case** | **Input Values** | **Expected output** | **Passed?** |
| Exit the program | Exit | The program to terminate gracefully. | Yes |
| Open the menu | menu | The menu, properly formatted with headers, as seen in the brief. | Yes |
| Add an item to the order | add 2 | The order has a single item added to it, corresponding to the number 2 item on the menu. | Yes |
| Add multiple items to the order | add 2 5 2 10 | The order has multiple items added to it, corresponding to the numbers given, including duplicates. | Yes |
| Remove an item from the order | remove 2 | The order has item 2 (corresponding to the second item in the checkout list) removed. | Yes |
| Remove multiple items from the order | Remove 1 2 3 | The order has multiple items removed. | Yes |