Design 3 Template: Coffee Shop

Part 1: Understanding the Problem:
(1 pt) [Rephrase the problem statement here]

This program is a coffee shop management platform. It allows a user to view, edit, and create the information for a coffee shop, including its menu, drink options, and prices. It also contains the processes necessary to place orders, search for items by name and price, and clone a shop and all of its contents.

To ensure that you understand the problem correctly, answer the following guiding questions:

1. **(1 pt) What are the available options to the user (the manager)?**

The available options to the user are: view shop info, release a new product, remove a product, search coffee by name, search coffee by price, place an order, clone the shop, and quit the program.

2. **(2 pts) In which text file(s) is the shop address, phone number, and coffee menu stored? How would your program load/populate the information?**

The text files in which the shop address, phone number, and coffee menu are stored in are shop_info.txt for the address and number, and menu.txt for the coffee menu. They will be loaded and populated through the use of fstream objects and user input.

3. **(2 pts) What are the required classes for this assignment? List every "has-a" relationship that you've observed between the required classes.**

The required classes for this assignment are Shop, Menu, Coffee, and Order. Every Shop has a Menu and an Order, and every Menu has an array of Coffee.

4. **(1 pt) What is the starting condition of the program? (i.e., what is the starting revenue and number of orders?)**

The starting condition of the program is that revenue and number of orders are set equal to zero.

5. **(1 pt) Do you need to load order information into the program, why or why not?**

You do not need to load order information into the program as the information is gathered through input prompts to the user.

(2 pts) [List assumptions that you are planning to make for this program]
(Hints:
What if the user enters "asdf" when selecting an option?
What if the user provides a file that does not exist?
What if the user enters a large price smaller than a small price?
What if the user enters a coffee name that already exists in the menu?
Etc.)

Assumptions I am making about this program are that the user will only enter characters, strings, and integers when prompted for those values, that the program will only take input files with specific, pre-defined names, and will output to the proper file. Other assumptions I am making are that the program will re-prompt the user if a menu item already exists, but that it will not re-prompt the user if a price is smaller than a small price. The program should follow a similar convention for other inputs with files or names that already exist, or other values that can be changed.

Part 2: Program Design:
To help you break the problem down into smaller subtasks, answer the following guiding questions:

1. **(1 pt) Of all the available options, in what order are you going to implement them and why?**

   Of all the available options, I am first going to implement the 'View Shop Info' option. From this option, I can work with a baseline to progressively add more content and features as needed, making for a smoother development process. After implementing the 'View Shop Info' option, I will likely add the ability to release and remove products to the menu, then in order, the ability to search for products by name & price, place an order, and clone the shop, and finally quit the program.

2. **(2 pts) The "Big three" (Copy constructor, assignment operator overload, and destructor) is likely needed for a class that contains dynamic memory. In plain English, explain the difference between a copy constructor vs. an assignment operator overload function.**

   A copy constructor allows for the complete cloning of a class, including all of its members and dynamic memory by creating a new object and assigning it the values of the object to be cloned. An assignment operator overload function allows you to assign an existing object to that of another already existing object, moving all the members to the first object.

3. **(1 pt) Of all required classes, which one(s) would need the Big three implementation and why?**

Of all the required classes, that primary class that would need the Big Three implementation is the Shop class. This is because when cloning a shop, we need to be able to access and clone all of its members, including the dynamic memory present in any arrays. This requires the use of the Big Three implementation.

4. **(2 pts) Using pseudocode, what is the general algorithm of adding an element into a dynamic array that is already full?**

The general algorithm to add an element to a dynamic array that is already full is as follows:

new array length = original dynamic array length plus *number of new elements*

new dynamic array of *new array length = original dynamic array elements* and *new elements*

5. **(2 pts) Using pseudocode, what is the general algorithm of removing an element from a dynamic array?**

new array length = original dynamic array length minus *number of former elements*

new dynamic array of *new array length = original dynamic array elements* minus *former elements*

Recall the goals of OOP

In OOP, "Information hiding" and "encapsulation" respectively refer to hiding the implementation details of an object from its user (e.g., private members), and exposing a restricted interface through which to interact with the object (e.g., public member functions). For example, the Coffee class implementation is hidden from the Shop class. If a function needs to access the internal details of a Coffee object, it should generally be implemented as a member function of the Coffee class. If the Shop class (or any other class / global function) needs to interact with the Coffee class in any way, it should do so through the Coffee class's public interface (e.g., public member functions).

Consider an alternative version of the assignment where the user only needs to be able to print the <u>first</u> coffee object in the menu using a Shop object s.

A common error is to use a chain of accessors/getters, which subverts the intention of encapsulation!

Wrong✕
```
void Shop::print_first_coffee() {
   cout << “Name: ” << this->m.get_coffee_array()[0].get_name() << endl;
   cout << “Small cost: ” << this->m.get_coffee_arr()[0].get_small();
   …
}

Shop s;
s.print_first_coffee();
```

Instead, we should first create a member function in the Coffee class to print a coffee object, such as:
```
void Coffee::print_coffee() {
   cout << “Name: ” << this->name << endl;
   cout << “Small price: ” << this->small_cost << endl;
   …
}
```

Then, to print the first coffee object in the menu, we can create a function in the Menu class that can directly access the array of Coffee objects, `coffee_arr`**:**
```
void Menu::print_a_coffee(int index) {
   //note how print_coffee() is called here
   this->coffee_arr[index].print_coffee();
}
```

Finally, we can call `print_a_coffee()` **from the Shop class:**
```
void Shop::print_first_coffee() {
   // Shop class has a Menu object m
   // pass index 0 to print the first coffee object
   this->m.print_a_coffee(0);
}
```

The way of calling it from main() stays the same:
```
Shop s;
s.print_first_coffee();
```

In general, you should choose your public member functions carefully—your goal is to only expose public member functions that accomplish the necessary objectives.

In the actual assignment, two of your objectives are to print the coffee with a given name, and to print all coffees under a certain price.

6. **Once you understand the goals of information hiding and encapsulation, use pseudocode to create designs of all options. For each option, list <u>what function(s) in which class(es)</u> you are planning to create, and the pseudocode for those functions.**

6.1. (1 pt) View shop info

Functions:

```
display shop address, phone number, and menu()
    read the address and phone number from the respective text file
    read the menu from the respective text file
    print out the address and phone number
    print out the menu

display total revenue()
    print out the total revenue for the shop from the member variable

display order information
    print out the order information for the shop from the member variable
```

6.2. (2 pts) Add an item to menu

```
ask user for the coffee name
ask user for the coffee prices
add new element to the array of coffee with the user-selected name and prices
```

6.3. (2 pts) Remove an item from menu

```
ask user for the coffee name to be removed
remove old element from the array of coffee with the user-selected name
```

6.4. (3 pts) Search by coffee name

```
ask user for the coffee name to be searched for
iterate through the coffee array until the coffee is found
if coffee isn’t found, report that it wasn’t found
otherwise, print out the coffee that was found
```

6.5. (3 pts) Search by price

```
ask user for the coffee price to be searched for
iterate through the coffee array until all coffee at or under the price is found
if coffee isn’t found, report that it wasn’t found
otherwise, print out the coffee that was found
```

6.6. (2 pts) Place an order

ask user for the coffee name and size to be ordered
add a transaction to the orders array
increase the number of transactions in num_orders
add the price paid for the order to total revenue

6.7. (3 pts) Clone a Shop
(Be specific, explain how you would test your big three/deep copy implementation)

using the assignment operator overload implementation, create a new shop
clone the current shop information to the new shop, ensuring all dynamic memory is copied as well
display the information for the old and new shop to confirm that the information has been successfully copied

7. (1 pt) How would you synchronize the .txt files with your program?

To synchronize the .txt files with my program, I will make sure that the fewest amount of fstreams are open as possible, that all parts of the code that need it are given updated access to the files through loops, and that there are no confusing fstream or function names that could make it hard to determine what is being changed or worked with in the file stream.

8. (1 pt) After executing once (i.e., complete one option), how would your program display all available options again, until the user chooses to quit?

After executing once the program will display all available options again by using a while loop, until the user chooses to exit

```
while user doesn’t wish to quit:
    display the menu
```

(4 pts) [Use your answer above, create pseudocode (or flowchart) for the remaining functions that you plan to create]

Very importantly, think about how you would connect all pieces together!

Tip: in order to modularize your program, let each function handle one thing/task. When decomposing a program into functions, try listing the verbs/tasks that are performed to solve the problem.

```
menu function: // printed to screen
        view the shop info
        add a product
        remove a product
        search coffee by name
        search coffee by price
        place an order
        clone a shop
        quit the program

        // not printed to screen
        take the user input
        based on user input, select one of the functions corresponding to the
desired outcome

quit the program:
        print a goodbye message
        break the while loop, which will return zero to main
```

Part 3: Program Testing
To help you consider the possible test cases, answer the following guiding questions:

1. **(1 pt) How would you test if the `menu.txt` and `shop_info.txt` are correctly loaded into your program?**

I will test if the menu and shop_info text files are correctly loaded into my program by printing out the information in each file & comparing it to what it should be in the arrays/variables, and by testing for any bad characters or data where it shouldn't exist.

2. **(1 pt) For your answers in Part 2 Q6, how would you test if the option is working as expected?**

To make sure each option in question 6 is working as expected, I would enter different lengths and options for inputs, printing and reviewing the results to make sure each function responds as it should for the given input and output types.

3. **(1 pt) What inputs need to be error handled?**

The inputs that must be error handled include the menu function, the search product by name and price features, the file I/O features, the name and price to add a product and the name to remove a product.

4. **(1 pt) How will you ensure that your program does not have any memory leaks? How will you locate the memory leaks if there are any?**

I will ensure that my program doesn't have any memory leaks by properly deleting all dynamic memory before the end of the program. During different stages of development, I will use Valgrind and gdb to discover and troubleshoot any memory leaks that are present.

(6 pts) [Create a testing table that has representative good, bad, and edge cases for each input, and their expected outputs]

Conditions	**Good**	**Bad**	**Edge**
Menu	User enters an integer within the range of available menu options; program continues through menu like normal	User enters an integer exceeding the range of available menu options; program re-prompts user until acceptable input is entered	User enters a non-integer value; program re-prompts user until acceptable input is entered
Search by Name	User enters a string that either matches or doesn't match a known coffee type; program continues response accordingly	User enters a string that contains spaces; program re-prompts until a single continuous string is entered	User only enters non-string values; program re-prompts until a single continuous string is entered
Search by Price	User enters an integer or float that either matches or doesn't match any known price range; program continues response accordingly	User enters an integer or float that is negative; program re-prompts until a valid integer or float is entered	User only enters non-integer or float values; program re-prompts until a valid integer or float is entered
File I/O	User inputs a file or path that can/does exist and contains no bad characters or data; data is appended to file	User inputs a path that does not exist; error message printed & user asked to enter a valid file/path	User inputs a file that is inaccessible by the program due to lack of file/access privileges; error message printed & user asked to enter a valid file/path
Add a Product	User enters the proper string and	User enters a string with spaces and/or	User enters non-string and/or non-

	float values for the product name and prices; program continues like normal	negative integer/float values for the name and prices; program re-prompts until valid values are entered	float types for the name and prices; program re-prompts until valid values are entered
Delete a Product	User enters a string that either matches or doesn't match a known coffee type; program continues response accordingly	User enters a string that contains spaces; program re-prompts until a single continuous string is entered	User only enters non-string values; program re-prompts until a single continuous string is entered