10 points.
Due: Monday, November 15, 2021 at 11:59 pm

Section 1 Program Summary

In this program, we will revisit the Assignment 2 Dairy Bar program, adding a few improvements based on the topics we've covered in the past couple weeks. In particular, we'll use file input/output to store a menu and receipts that live on separately after the program closes, and add some custom operators to make the core logic easier to read and write.

Note: You are welcome to use the code you submitted for assignment 2 as a starting point for assignment 4. In fact, we encourage you to do so to gain experience maintaining and improving your own codebase - returning to old code you wrote can help you see what parts of your personal coding style can be refined. That said, we will provide starter code for this assignment, please see details near the end of this document.

Section 2 Existing Classes/Structs

In this section, we'll briefly review the pieces of the original program.

(a)

Drink Struct

This is a data structure to represent a drink. Each drink has a name, a base price, and a STYLE. There are four styles of drink:

1. NEAT: Serve the drink as-is. The total price is just the base price.
2. ROCKS: Throw in some ice for an extra-cold drink. Add 0.25 to the base price.
3. DOUBLE: Double the amount of delicious dairy goodness in the drink. Multiply the base price by 2.0.
4. TALL: If the Customer can't handle a full-strength drink, cut it with premium ~~water~~ skim milk. Add 1.00 to the base price.

The public functions for the `Drink` struct are:

- `Drink(string _name, float _price, STYLE, _style = NEAT)` (parameterized constructor)
- `Drink(const Drink& copy_from)` (copy constructor)
- `void Prepare(const STYLE serving_style)`
- `float TotalPrice()`
- `void Print()`

(b)

Customer Class

This is a data structure to represent a Customer. Each Customer should has an array of drinks and a count of how many drinks are in the array. In assignment 2, the array had size 10, and we ignored any new orders beyond that limit.

The public functions for `Customer` are listed below:

- `Customer()` (default constructor)
- `void Serve(const Drink& drink, const STYLE style)`
- `float TotalTab(const float tip)`
- `void Print(const float tip)`

(c)

Main Program

The menus and menu items for the main program are:

1. In the main menu, the options are to order a drink or close your tab.
2. If the user chooses "order a drink," they will get a new menu asking which drink they would like to order.
3. When the user picks their drink, they will receive an additional sub-menu asking how they want the drink served, and then be returned to the main menu.
4. If the user chooses "close your tab" in the main menu, they will receive an additional prompt asking how much they would like to tip, and the program will then print a receipt.
   Finally, the program will reset the Customer object, so they start with an empty list.

The "main" and "drink" menus each include an "exit" option as well. For the main menu, "exit" will end the program. In the drink menu, exit will return the user to the main menu.

Menu Formatting/Inputs
Each menu item has a number indicating the value for the user to enter to pick that menu item (see example menus below). If the user enters a number that does **not** correspond to a menu entry, the program re-prompts the user and reads in a new number. If the user enters an invalid type, putting `cin` into the fail state, the program clears the error and ignores the remainder of the input line, and re-prompts untils the user provides a valid input.

Input for tipping
The program interprets the input as a percentage; that is, an input of 5 implies a 5% tip. As with the menus, the program checks that the input was the correct type, and is non-negative, re-prompting until the user has given a valid input.

Section 3 New Features

We will now cover the new features to be implemented for this assignment. These include several operators to override, as well as file I/O features to read in a menu and save receipts to file.

(a)

New features for Drink

- `operator>>`
  In this program, we'll be reading the drink menu and existing tabs from a file. To do this, it will be helpful if we can easily read Drink objects with a stream extraction operator.
  So, you should create an `operator>>` using the conventions for return type/parameters discussed in lecture. The function should read in the drink name, followed by price, and lastly the style.
  The style will be an all-caps string representing the style; you'll need to read the string and use a switch or if-else block to get the correct `STYLE` variable. The data will always be separated by a space in the input file, on a single line.
  For example, in the input file, one line may look like:
  `Skim 1.00 ROCKS`
  Note that each drink will have a single-word name this time to avoid complicated string parsing.

- `operator<<`
  This is essentially the opposite of the extraction operator above. Again, use the conventions discussed in class for the return type and parameter(s).
  Output the name, base price, and an all-caps string representing the STYLE, with each separated by a space. Include a newline (either `'\n'` or `endl`) at the end of the output, so the Drink is printed to its own line.

(b)

New features for Customer

In addition to displaying receipts to the user, we would like the ability to store receipts for later review. To do this, we'll need to add a couple features to the Customer class:

- `string name`
  The Customer class should get a new member variable to store their name.
- `Customer(string _name)` (param'd constructor)
  You should add a parameterized constructor, which accepts a name as input, so that a new customer can have a name.
- `string GetName() const`
  Finally, we need a way to check a Customer's name. This function should just return the name variable.
- `operator>>`
  Like the Drink class, we'll read in data for a Customer from a file in some cases. Again, use the conventions from class for the return type and parameters. When reading a customer, the first item to read is the name, followed by the drink count (note the limit is still 10). As with Drink, we will only use single-name customers. After this, you can run a loop, reading in one Drink at a time into the Customer's drink array (with the » operator), until you reach count.
- `operator<<`
  Write a stream insertion operator for Customer, continuing to follow the conventions outlined in lecture. As with Drink, this is the reverse of the input operator:
  First, output the name and the drink count, separated by a space. Add a newline, then write out the Drinks from the array, one at a time.

(c)

New features for main menu

First, when the program starts, before you display the main menu, you should ask the user to enter their name. Use this name for the initial Customer object. From this point on, whenever the user pays or saves (see below) their tab, you should prompt for a new name. Then, as with assignment 2, call the constructor to replace the Customer with a fresh instance, using the new name.

In addition, when the program starts, you should open a filestream to read a file with the name "menu.txt". The first line will contain an integer indicating the number of drinks in the menu. The remaining lines will contain drinks in the format discussed previously (all drinks in the menu.txt have a NEAT style).

You should dynamically allocate an array of Drinks, using the size given in the first line. Then, you should run a loop to read in each Drink in the menu.
Note that, unlike Assignment 3, you only need a Drink*, not Drink**, since the Drinks don't need to be accessed as pointers.
We will add two items to the main menu:

- Save Tab
  If the user selects this option, your program should open a filestream to output to a file named `"{customername}.txt"`, where `{customername}` is replaced by the Customer's name. Then, output the customer to this file with the << operator, close the file, and prompt for a new customer name (as described above).
- Load Tab
  If the user selects this option, your program should open a filestream to read a file with the same naming convention as the "Save Tab" case (`"{customername}.txt"`). Then, use the »|| operator to read in the customer data from this file. We will never attempt to load a tab for a customer who does not have a .txt file.

Note that our changes to allow saving and loading of Customer data does not change the way that paying a tab works. You can continue to use the existing Customer functions for paying the tab.

Section 4 Menu Examples

Below, we've given updated examples of each menu, which show what items the menus should have, and in what order they should go. Your program is **not** required to print exact copies of the examples here (though you're welcome to copy the example formats if you like), but you **must** keep the menu items in the same order.
Note that the details of the "Drink" menu will vary depending on what was in the `menu.txt`.
Here's an example of the main menu:

```
Welcome to the Downtown Dairy Bar!
[1] Order a Drink
[2] Pay Your Tab
[3] Save Tab for Later
[4] Load Previous Tab
[5] Exit program
Please enter a menu item:
```

And an example of the "Drink" menu:

```
Which Drink Would You Like to Order?
[1] Skim
[2] 2%
[3] Doogh
[4] Milkshake
[5] Smoothie
[6] Return to main menu
Please enter a menu item:
```

And an example of the "how to serve" menu:

```
How Would You Like Your Drink Served?
[1] Neat
[2] On the Rocks
[3] Double
[4] Tall
Please enter a menu item:
```

Finally, an example of a "receipt":

```
Skim     (neat):       $1.00
Doogh    (tall):       $4.00
Smoothie (neat):       $4.50
  ***
Tax:                    5.0%
Tip:                   15.0%
Total:                $11.40
Enter any number to return to menu.
```

Section 5 File Examples

Below, we'll also show examples of what the menu.txt and customername.txt files should look like. Note, we'll aslo provide an actual sample menu.txt.
First, the sample `menu.txt`:

```
5
Skim 1.00 NEAT
2% 2.00 NEAT
Doogh 2.00 NEAT
Milkshake 5.00 NEAT
Smoothie 4.50 NEAT
```

Second, the sample `customername.txt`:

```
customername 3
Skim 1.00 NEAT
Doogh 2.00 TALL
Smoothie 4.50 NEAT
```

Section 6 Getting Starter Code

We will make "starter" code available for assignment 4. This will come in the form of an official solution to assignment 2, which you can download from Canvas (we will provide a link in the assignment description). You may start assignment 4 using the starter code, or start with your own assignment 2 solution. If you prefer your own code but have some bugs, you can pick and choose parts of the official starter code to help fix those existing issues.

Section 7 Deliverables

You should submit a single .zip file, named `[NetID]_asg4.zip`, where [NetID] is replaced by your UW Net ID (i.e. the user name you use to log into Canvas). Inside the zip file, there should be the following files:

`main.cpp, Drink.h, Drink.cpp, Customer.h, Customer.cpp`

Note that you do not need to submit any of your .txt data files (menus, saved tabs, etc).

In each cpp file, you should include a comment at the top of the file with your name and Net ID.
When main.cpp is built and run, it should print out a menu similar to the examples, and allow the user to interact with the menus as described above.

Section 8 Hints

While not required, you are welcome to add private functions to `Drink` and `Customer` as desired. Just don't change/remove any of the functions that were specified above.

To build your code on the CSL machines, run `g++ main.cpp Drink.cpp Customer.cpp -o main`.

Section 9 Rubric

Item	Points
Program builds and runs without compile errors	3
Drink has string insertion and extraction operators (<< and >>)	1
Drink operators are implemented correctly	1
Customer has a name and related functions, as well as insertion and extraction operators	1.5
Customer operators are implemented correctly	1
Main menu has options to save and load files with a Customer's tab	1
Program correctly prompts for a new Customer's name	0.5
Program reads a menu from file, and correctly shows all items from file	1
Total	**10**