10 points.
Due: Friday, October 15, 2021 at 11:59 pm

Section 1 Program Summary

As a UW-Madison student and die-hard milk-chugging teen, you've taken it upon yourself to open a new dairy bar in downtown Madison. To help out, you've decided to program an automated bartending assistant. For this assignment, you'll create a program that allows a customer to order drinks and pay their tab.
You'll also get some practice creating and using classes in C++. We will again provide header files with a basic code skeleton for each class. However, you will need to fill in the data and function headers based on the instructions in the sections below, and write function bodies in corresponding .cpp files. You will also add code to check for invalid input types in your menus.

Section 2 Classes/Structs

In this section, we'll give details on how to implement the program outlined above.

(a) Drink Struct

This will be a data structure to represent a drink. Because its functions are generally simple (most can be written in one line of code), creating this as a `struct` is appropriate.

Each drink should have a name, a base price, and a STYLE. We will include a STYLE enum in the starter code, with elements NEAT, ROCKS, DOUBLE, and TALL.
More details on what each STYLE means will be given in the description for the `TotalPrice` function.
The public functions for the `Drink` struct are listed below:

- `Drink(string _name, float _price, STYLE _style = NEAT)` (parameterized constructor)
  This is a standard param'ed constructor used to create new drinks. The style should default to `NEAT`.
- `Drink(const Drink& copy_from)` (copy constructor)
  Create a copy constructor for the class as well, which will copy all three attributes from the given Drink. You should use this when copying Drinks from the menu into a Customer's list of purchased Drinks.
- `void Prepare(const STYLE serving_style)`
  This is the function used to "prepare" a drink, which in our case just means setting the Drink's style. In other words, this function is just a "setter" function in disguise. Again, you'll use this when a Customer orders a Drink.
- `float TotalPrice()`
  This function should calculate and return an adjusted price for the Drink, depending on the base price and the 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.

  Return the calculated price of the drink.
- `void Print()`
  Print the name and total price (**not** the base price) on a single line with `cout`. The price should be limited to two decimal places.
  You should add a newline character to the end of the output as well.

(b) Customer Class

This will be a data structure to represent a Customer. The Customer will have functions which are somewhat more complex and "active", such as tallying a price from a list of items, and calculating taxes/tips. Thus, it makes sense to use a Class in this case.

Each Customer should have an array of drinks and a count of how many drinks are in the array. The array should have size 10. If the array becomes full, simply ignore any new orders (the bartender should really cut them off at this point).
Note: You might want to define a constant for the drink limit.

The public functions for `Customer` are listed below:

- `Customer()` (default constructor)
  This is a standard default constructor used to create a customer. The drink count should be set to 0.

- `void Serve(const Drink& drink, const STYLE style)`
  If the Customer has not reached their drink limit (i.e. drink count $<$ drink limit), you should copy the given drink into the customer's drinks array, and call the copy's Prepare function to set the STYLE. You should also increase the drink counter.
  If the Customer *has* reached their limit, return without doing anything.

- `float TotalTab(const float tip)`
  This function should calculate and return the total amount on the Customer's tab.
  This is the sum of all Drinks in the Customers drink list, plus tax, plus tip. You may hard-code the tax as 5.0% (which you may wish to store as 0.05 for ease of multiplication). The tip should be calculated separately from tax; that is, the tip price should not affect the tax, and the tax should not affect the tip.
  For example: suppose a customer ordered two skim milks, and paid a 10% tip. Then it is $2.00 for the drinks, so they owe 0.10 in tax. They separately pay the tip on the original $2.00 cost, so the tip is 0.20. Then the grand total is $2.30.

- `void Print(const float tip)`
  First, Print each Drink from the Customer's drink list.
  Then, print one line with three star (*) characters as a separator.
  Next, print a line with the tax percentage, and another line with the tip percentage.
  Finally, `cout` the total cost after tipping and taxing, followed by another newline.

Section 3 Main Program

(a) Menus

The menus and menu items for the main program will be as follows:

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 (you can just use the constructor for this, since we're not dealing with pointers yet).

The "main" and "drink" menus should 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.

(b) Menu Formatting/Inputs

Each menu item must have a number indicating the value for the user to enter to pick that menu item (see example menus below).

As with assignment 1, if the user enters a number that does **not** correspond to a menu entry, your program should re-prompt the user and read in a new number. Again, this "re-prompt" can either re-print the menu, or print a new prompt. The choice is yours, so long as your program does not wait *silently* for valid input. Continue to prompt the user until the number they enter is a valid menu item.

Unlike the previous assignment, there is no guarantee the inputs will be of the right type. We are, of course, dealing with milk-chuggers instead of geometers, and can not expect the same kind of precise data entry as we did before. Thus, whenever your program reads in a number for any menu/prompt, it should always check whether `cin` has entered the fail state. If it finds `cin` in the fail state, it should clear the error and ignore the remainder of the input line, as we covered in class. Finally, it should re-prompt until the user provides a valid input.

Below, we've given 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. While exact text of each menu item is up to you, the items should have reasonable labels.

Here's an example of the main menu:

```
Welcome to the Downtown Dairy Bar!
[1] Order a Drink
[2] Pay Your Tab
[3] Exit program
Please enter a menu item:
```

And an example of the "Drink" menu:

```
Which Drink Would You Like to Order?
[1] Whole Milk
[2] 2% Milk
[3] Heavy Cream
[4] Milkshake
[5] Malt
[6] Skim Milk (Water)
[7] 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] Straight
[2] On the Rocks
[3] Double
[4] Tall
Please enter a menu item:
```

Finally, an example of a "receipt":

```
Whole Milk (straight): $2.50
Heavy Cream (tall):    $4.50
Skim Milk (straight):  $1.00
  ***
Tax:                    5.0%
Tip:                   15.0%
Total:                 $9.60
Enter any number to return to menu.
```

(c) Drinks Menu

When you create the main program, it will probably be helpful to hardcode the Drinks for your menu. These can then be used to make copies when a Customer places an order. Below are the drink names and base prices you should use:

- Whole Milk, $2.50
- 2% Milk, $2.00
- Heavy Cream, $3.50
- Milkshake, $5.00
- Malt, $6.00
- Skim Milk, $1.00

(d) Input for tipping

As with the menus, when you prompt for a tip percentage you must check that the input was the correct type. Also, as with the inputs for shapes in assignment 1, you'll again need to ensure the user does not enter a negative number. You should re-prompt until the user has given a valid input.

You should interpret the input as a percentage; that is, an input of 5 would imply a 5% tip. For the sake of easier calculations, you may find it helpful to divide the input by 100.0, so that you can then multiply a cost by the value to get the tip amount. In any case, do not assume the user is giving an input already divided by 100.

Section 4 Submission

Submit a zip file, which contains three .cpp files called `main.cpp`, `Drink.cpp`, and `Customer.cpp`, as well as completed copies of the `Drink.h` and `Drink.cpp` files.

Your zip file should be named *NetID*`_asg2.zip`, where *NetID* is your Net ID (the username you use on Canvas, *not* the 10-digit number on your wiscard).

In each cpp file, you must include a comment at the top of the file with your name and Wisc 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 5 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.

Section 6 Rubric

Item	Points
Program builds and runs without compile errors	2
Program has a main menu, with all 3 options specified	1
Menu for ordering a drink has the correct list, and requests a serving style	1
The “pay tab” option correctly prints the receipt	1
Drink class has all required functions and data	1
Customer class has all required functions and data	1
Calculation of Drink prices are implemented correctly	1
Calculation of the Customer’s total tab price is correct	1
After closing a tab, program correctly resets customer to an empty tab	0.5
Program re-prompts for valid input after receiving an invalid type	0.5
Total	**10**