Project 5

The Coffee Shop Queue

After creating your coffee shop algorithm, your coffee shop has been going great. So well, in fact, you are considering expanding your staff to have more baristas. To decide how to staff your shop, you want to measure how each change will improve the customer experience. You will write a program that simulates making orders using a queue. You can then use the program to analyze customer wait times on different days of the week.

Learning Objectives covered:

1. I can write code in C++ that uses auto and arrays.
2. I can appropriately allocate memory on the heap using smart pointers.
3. I can work with queues, lists, and other data structures in C++
4. I can write a program that adheres to a style guide.
5. I can write high-quality documentation.

Overview:

Your program will model a coffee shop with one or more baristas. As customers enter the shop, their orders are added to a queue. Each barista will select the front item in the queue, complete the order, and then get the next item. To represent the customers, you will use a file containing a list of orders with arrival times and the time it takes to complete each order. Your final analysis will show the average wait time for each order and the baristas' idle time.

Details:

Program structure:

1. Two **provided** header files
   - `shopQueue.h`: Definition of the `ShopQueue` class
   - `order.h`: Definition of the `Order` class
   - `Barista.h`: Definition of the `Barista` class
2. Two **provided** source code files
   - `shopQueue.cpp`: Implementation of the `ShopQueue` class
   - `order.cpp`: Implementation of the `Order` class
3. Two additional source code files
   - `myShop.cpp`
     - contains `main()`
     - manages the shop
   - `Barista.cpp`: Implementation of the `Barista` class
4. A `makefile`

Before starting: Carefully read the code in `shopQueue.cpp, shopQueue.h, order.cpp and order.h`

Compiling and running:

Your program should compile to the executable: `run_simulation`
Your program will be compiled using your `makefile`.

Your program should be run with the following command line arguments:

```
./run_simulation inputFile numBaristas outputFile totalTime
```

`inputFile`: text file containing order information
`numBaristas`: the number of baristas
`outputFile`: output file name
`totalTime`: number of minutes to run the simulation

The `Barista` Class:

The baristas will be represented as a class with the following member variables:

1. `ID_number`: set first barista to 0, second to 1, etc.
2. `is_busy`: 1 for busy, 0 for not busy
3. `task_start`: the timestep when they started the task
4. `task_duration`: how long the task will take to complete
5. `free_time`: The total time the barista was free during the simulation

Read `Barista.h` then **implement `Barista.cpp`**

The Input Files:

Each input file will contain 3 columns:

1. The order number
2. The arrival time in minutes
3. The "cook" time: How long it takes to complete the order in minutes

The orders:

Store the orders in a queue. You have been provided with the `shopQueue` class. Each order in the queue has the following member variables:

1. `ID`: stores the order number
2. `arrival`: stores the arrival time
3. `cook_time`: stores how long it takes the complete the order
4. `start_time`: the time the item is removed from the queue and assigned to a Barista.
5. `wait_time`: How long the item was waiting in the queue.

To set up the queue, read the input file, adding each item to the queue. Use the values from the input file to set the `ID`, `arrival` and `cook_time` variables. Set the values for `start_time` and `wait_time` to -1. When the item is removed from the queue, set the `start_time` to the current time and calculate the `wait_time` (`start_time – arrival`)

The Baristas:

Keep track of the baristas using a `vector`. At each time step, determine if the barista is busy. If so, see if their order is done. Then, see if they can take a new order.

You will be implementing the `Barista` class on your own.

Set up the simulation:

1. Read the input file of orders into a queue.
2. Initialize a vector of baristas, with `is_busy`, `task_start` `free_time`, and `task_duration` set to zero.
3. Create a vector to store completed orders. This will allow you to print them to the output file at the end of the program.

The simulation:

Use a loop to run your program for desired number of time steps. At each time step (representing 1 minute), check to see if the barista has finished their task by comparing the current time to `task_start + task_duration`. If they are finished, set `is_busy`, `task_start` and `task_duration` to zero.

Assign new orders to baristas from the front of the queue. Check that the current time is greater than arrival time of the front order. (If not, then the customer hasn't "arrived" at the store yet). If so, remove the order from the queue and add it to the completed order vector. Modify `is_busy`, `task_start` and `task_duration` of the barista.

Finally, check if baristas are still free. If so, increase the value of `free_time` for the barista.

Hint: Start writing your program to work with 1 Barista. Then modify it to work with multiple.

Output:

As your program runs it should do the following:

1. Check for the number of command line arguments
   Print the following error if there are not enough arguments.

```
Error: Wrong number of arguments!
Usage: ./run_simulation <inputFile> <numBaristas> <outputFile> <totalTime>
```

2. Print the welcome message (replace X with the number of Baristas)

```
Welcome to Gompei's Coffee Shop!
------ # of Baristas: X ------
```

3. At the end of the simulation:
   a. Print the contents of the completed order vector the output file.
      Example file is on Canvas.

b. Calculate and print the following summary order information:

```
X customers are waiting for their order.
Y customers have been served.
The average wait time was Z.ZZ minutes.
```

Replacing X with the length of the queue at the end of the simulation, Y with the number of completed orders, and Z with the average wait time of all orders in the completed order vector. Print Z with two values after the decimal point.

c. Print the ideal time of each Barista with the following format:

```
Barista X was idle Y.YY% of the time.
```

Replacing X with Barista ID and Y with the Barista `free_time` divided by the total time of the simulation. Print Y with 2 values after the decimal point.

Notes for working with multiple Baristas:

1. The first Barista will end up with a much lower idle time than the other Baristas. This is due to how we are running the simulation.
2. Make sure the order arrival time is >= current time before starting an order.

Grading:

The full grading rubric is explained in the CS 2303 Project Guidelines document. Two additional rubric items are listed below.

Met	Rubric Item
	The program creates and uses a `vector`
	Function prototypes are in the .h file and the .h file included correctly in the .cpp files

The grading breakdown is shown in the following table. To earn a particular grade, you must meet all criteria in the row corresponding to that grade. There are 7 core rubric items and 7 supplemental items (5 in the Rubric document and 2 listed above).

		Rubric Items Met	
	Autograder tests	**Core**	**Supplemental**
Excellent	All Passed	7	6+
Meets Expectations	All Passed	7	5+
Needs Revision	All Passed	<7	N/A
Not Graded	Failed 1 or More test	N/A	N/A