* the problem you are solving
  + I was aiming to make a restaurant simulation program, which would simulate the day of a restaurant. It would use customers and employees as objects which would interact with each other through tasks. This would hopefully create a program which somewhat simulated what it is like to have a restaurant where employees are continuously taking orders, making food, and delivering food to customers.
* the approach taken
  + I took the approach of using an overarching Restaurant class which contains all of the logic required for the restaurant, keeping main clean and simple. The aspect which took the longest to figure out was how to model each step of time and how to manage the tasks. I settled on using a TaskManager which would contain a vector of Task objects, each containing information on the employee, customer, order, and duration involved. The TaskManager was responsible for updating the vector and returning new tasks to be assigned once old tasks were complete.
* the algorithm(s) used
  + One algorithm I used was a searching algorithm. My getFreeEmployee() function searches through a vector of employees, and returns a pointer to the first free employee it finds.
  + I also used queue management to keep track of Tasks in my TaskManager’s queue. My TaskManager’s updateQueue() method goes through each task in the queue, updates its time, and assigns the next task in the sequence if the task is complete. For example, if a take order task is complete, then it would create a new task with that customer and return it so that the Restaurant could assign a new employee to that task. This made it so that every task was completed in order as they were assigned.
  + I used lots of pointers in this program, because employees and customers were used in every Task. So, to avoid duplicating objects I simply passed them to Tasks as pointers, so that their data could be modified while still staying in their vector.
* the data structure(s) used
  + I mostly used vectors for this program, as they were the most adaptable and convenient for my purposes. If I had more time I would use std::queue for the queues in the program, but the vectors work just fine. I use vectors to store the employees, customers, and menu items, as well as for the queue of tasks. I also ran into a problem where I couldn’t assign new employees to a task from the Task.cpp file because it would create a circular dependency, so I settled for creating a vector of new tasks which would have employees assigned from the Restaurant side of things.
* inputs and results
  + The program takes input for:
    - Restaurant name
    - Menu items, including:
      * item name
      * item price
      * time to make item
    - Number of cooks to hire
    - Number of waiters to hire
  + The program outputs:
    - A budget at the beginning of the day
    - A simulated day in the restaurant, with every event that takes place
    - An ending budget which shows how much money the restaurant made
* sample input and output files

Welcome to the Restaurant Simulator!

This program will allow you to live out your dreams of owning a restaurant - without the hard parts!

---Setup---

1. Choose a name for your restaurant: Jack's Donuts

Your restaurant is called "Jack's Donuts"

---Menu Creation---

Now, create 1 - 5 menu items for your menu.

How many items would you like to create? 3

Enter a name for menu item number 1: Chocolate Donut

Enter a price for this item between 5 and 20 dollars: 5

Enter how many minutes (between 10 and 30) it takes to make this item: 10

Enter a name for menu item number 2: Vanilla Donut

Enter a price for this item between 5 and 20 dollars: 5

Enter how many minutes (between 10 and 30) it takes to make this item: 10

Enter a name for menu item number 3: Strawberry Donut

Enter a price for this item between 5 and 20 dollars: 5

Enter how many minutes (between 10 and 30) it takes to make this item: 10

Here is your new menu:

1. Chocolate Donut

- Cost: 5

- Ingredient Cost: 1.5

- Time to make: 10 minutes

2. Vanilla Donut

- Cost: 5

- Ingredient Cost: 1.5

- Time to make: 10 minutes

3. Strawberry Donut

- Cost: 5

- Ingredient Cost: 1.5

- Time to make: 10 minutes

---Hiring Employees---

How many cooks would you like to hire? (Maximum 4): 2

How many waiters would you like to hire? (Maximum 3): 3

---Generating Employees---

Cooks: 2

Waiters: 3

---Budget---

Starting Capital: 500.00

Revenue: 0.00

Costs: 50.00

Profits: -50.00

Ending Capital: 450.00

---Beginning Simulation---

Time: 00:00

Time: 00:05

Time: 00:10

Time: 00:15

- Taking Orders:

- [ACTION] Benjamin (Waiter) has started taking Dylan's order of Chocolate Donut

- [ACTION] Isabella (Waiter) has started taking Logan's order of Strawberry Donut

Time: 00:20

- Taking Orders:

- [COMPLETE] Benjamin (Waiter) has finished taking Dylan's order of Chocolate Donut

- [COMPLETE] Isabella (Waiter) has finished taking Logan's order of Strawberry Donut

- Making Orders:

- [ACTION] Lily (Cook) has started making Dylan's order of Chocolate Donut

- [ACTION] James (Cook) has started making Logan's order of Strawberry Donut

Time: 00:25

- Making Orders:

- [COMPLETE] Lily (Cook) has finished making Dylan's order of Chocolate Donut

- [COMPLETE] James (Cook) has finished making Logan's order of Strawberry Donut

- Delivering Orders:

- [ACTION] Benjamin (Waiter) has started delivering Dylan's order of Chocolate Donut

- [ACTION] Isabella (Waiter) has started delivering Logan's order of Strawberry Donut

Time: 00:30

- Delivering Orders:

- [COMPLETE] Benjamin (Waiter) has finished delivering Dylan's order of Chocolate Donut

- [COMPLETE] Isabella (Waiter) has finished delivering Logan's order of Strawberry Donut

Time: 00:35

Time: 00:40

Time: 00:45

Time: 00:50

Time: 00:55

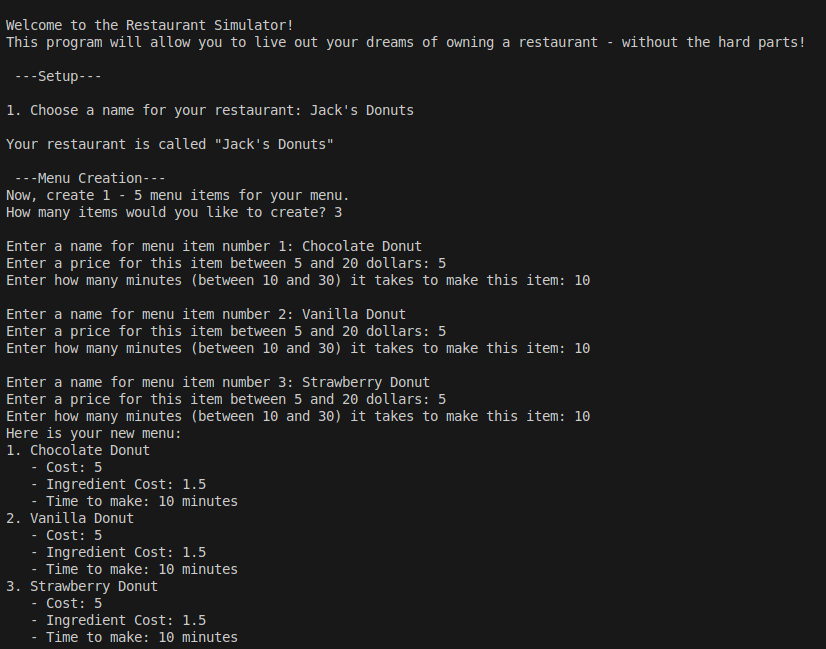
Time: 01:00

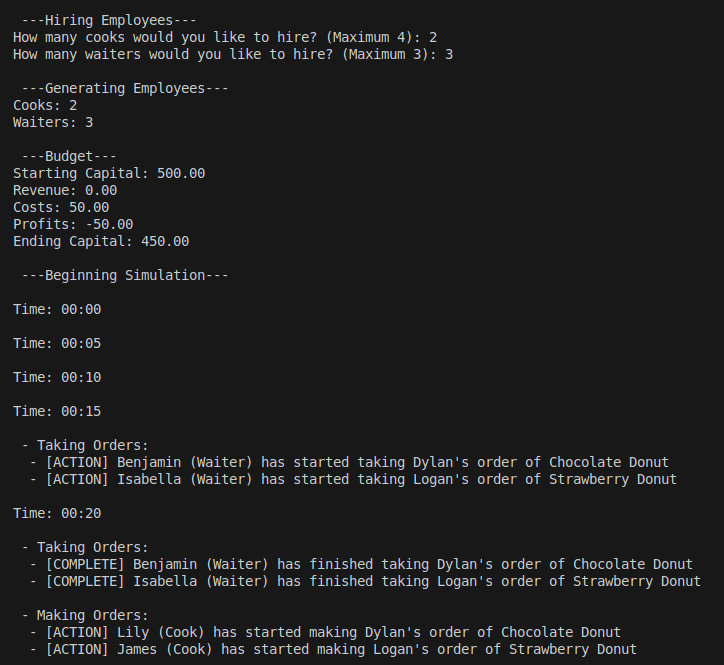
Time: 01:05

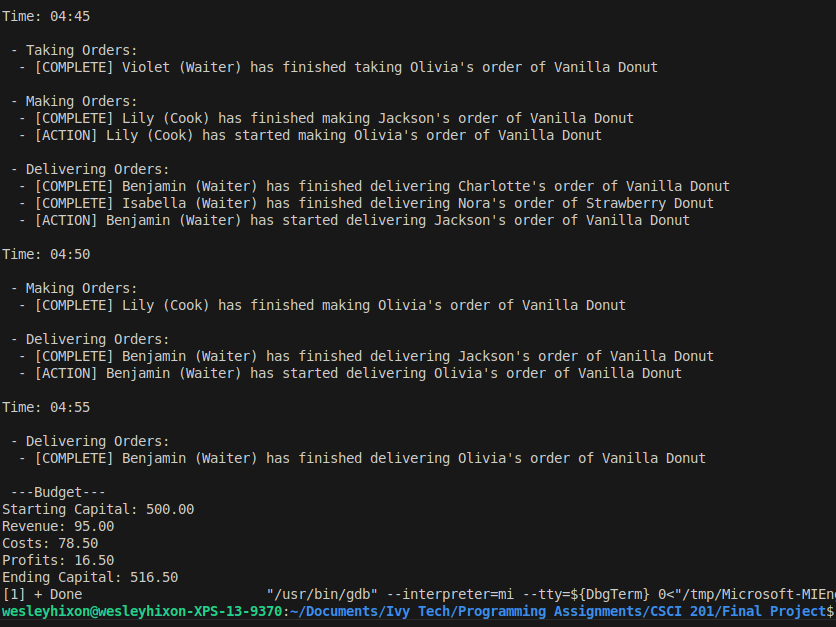
- Taking Orders:

- [ACTION] Benjamin (Waiter) has started taking Henry's order of Strawberry Donut

* screenshots demonstrating the program runs







* lessons learned
  + This program was really challenging, and I’ve learned a lot about object-oriented programming from it. It was the first program I’ve ever had to split into separate header and implementation files, because it was going to be unmaintainable if I kept it in one file.
  + The hardest part was definitely implementing the task system, and figuring out how to update them every step of time. In the end I learned a lot about how to manage a queue, and why using objects makes coding a lot easier sometimes! I can’t imagine how difficult this would’ve been if I only used functions and variables.
  + I also learned a lot about using pointers, to the point where using pointers and iterators became second nature by the end of it. I definitely see the utility in it now, not having to create a new object for every reference. Although, some of my code would’ve probably been simpler if I just passed by reference instead of using a pointer. Ah well.
  + Another aspect that was challenging was figuring out how to trickle customers to the restaurant steadily without overwhelming it. I ended up setting the chance of a customer walking up to 25% every tick, and between 1 and 2 customers every time. When they walk up, they’re added to a vector of pointers, and then the waiters go through that vector and create a new “take\_order” task for every customer there. That way, customers can wait in line.
  + If I had more time, I would’ve implemented more statistics, like customer satisfaction, how long they’ve been waiting, and had customers be more likely to purchase cheaper food. That way, there’s an incentive to lower prices in game. Currently, you can jack up the prices and make more money every time. Also, I would’ve added more days, added a save system, and gameified it more to make it more fun to play. Overall, I’m happy with the result though.