Aleisha Smith

Julia Abbott

**Description/Requirements:**

Our program will simulate a restaurant for the time in hours provided by the user. The program will simulate the profits and expenses as a varied number of customers enter the premises and purchase at least a dish from the main course. The likelihood that they will purchase drinks, appetizers, and dessert in addition to the main course decreases in the order that they are listed. The probability that they will order a main course is 100%, a drink 70%, an appetizer 40%, and a dessert 20% (we operate in a healthy town). The balance starts out at zero each time the simulation is run, and increases with each item a customer purchases. This rate randomly fluctuates, as both the items a customer will purchase as well as the number of items varies each time a customer orders and the simulation is run. The expenses are the pay rate for the staff, and is taken out of the balance at the end of the simulation after the profits are put in. This is a set rate, and increases linearly with the total amount of time the simulation is run. The graph will be a bar graph depicting the number of customers, money coming in, money going out, total balance, and run time of the simulation. We will break these up into two graphs – one showing only one customer per hour with the total amount of time the simulation is running increasing, and the other showing the simulation running for only one hour, with the number of customers per hour increasing. The other variables(min and max amounts of time for the servers and the cooks) will stay constant with the minimum time being one, and the maximum time being two.

**USE CASES:**

1.) The owner of a store wants to calculate their projected income based on the hours they are open and how many customers are expected to arrive

2.)The store is not open any hours, so the net balance remains in equilibrium.

3.)The store is only open one hour, but more than a hundred customers arrive to buy food for a holiday.

4.)The store is open a week, but the servers and cooks are overworked and their minimum time to complete tasks increases drastically

5.)Researchers wish to test various scenarios to find evidence supporting their new hypothesis as to how to increase revenue.

6.)Testing how many staff are optimal for the best efficiency for realistic minimum and maximum wait times.

7.)Space chinchillas and alien robots are coming to conquer earth, and need to understand how to run a successful business so that they can raise money for their conquest.\*Note: This is a joke

main:

1) Set srand(time(0)).

2) Set again to true.

3) While again is true, do steps 4-5. (L)

4) Call complete set\_a\_cycle().

5) Set again to the the return value from go\_again.

\*\*\*\*\*\*\*\*STEP 4 OF MAIN\*\*\*\*\*\*\*\*\*

Class Cycle

MEMBERS:

-int max\_c

-int min\_c

-int max\_s

-int min\_s

-int arrival\_rate

-Fincances fin

-double prof

-double exp

METHODS:

Name: Constructor

Desc: Default constructor.

Params: None

Return: None

Name: complete\_a\_cycle()

Desc: Completes a cycle by running the simulation.

Params: None

Return: void

1) Create total\_time.

2) Ask user for the total simulation time (in hours), read in response.

3) Create a Restaurant named r.

4) Create an int called clock.

5) Call set\_cycle().

6) Set prof to 0.

7) Set clock to 0.

8) While clock is less than total\_time do steps 9-14. (L)

9) Create a cook called co using min\_c and mx\_c as parameters.

10) Call cooks do\_work method.

11) Create a server named s using min\_s and max\_s as parameters.

13) Set prof equal to Restaurant's run\_sim method, sending in clock and arrival rate.

14) Call setProfits sending in prof.

15) Call making\_a\_profit.

16) Print out the current balance.

17) Calculate the total to be paid to the workers.

18) Call paying\_expenses sending in the total pay for the workers.

19) Print out the balance.

20) Set the balance to 0.

Name: set\_cycle()

Desc: Sets the max and min values for cooks and servers, and the arrival rate.

Params: None

Return: void

1) Asks the user for the arrival rate of the customers, read in response

2) Ask the user for the min cooking time, read in response.

3) Ask the user for the max cooking time, read in response.

4) While min > max repeat steps 2 and 3. (L)

5) Ask the user for the min serving time, read in response.

6) Ask the user for the max serving time, read in response.

7) While min > max repeat steps 5 and 6.

class Restaurant

MEMBERS:

-queue<Customers \*> cust

-double cust\_pay

METHODS:

Name: Constructor

Desc: Default constructor

Params: None

Return: None

Name: run\_sim()

Desc: Runs the simulation.

Params: int clock, int arrival\_rate

Return: double

1) return the value from do\_stuff.

Name: do\_stuff

Desc: Uses the customer queue to get the orders and the price paid by the customer.

Params: int clock, int arrival\_rate

Return: double

1) If( (rand() / RAND\_MAX) < arrival\_rate), push a new customer into the queue.

2) If the queue is not empty, do steps 3-9.

3) Create a customer name custer.

4) Set cust\_pay to 0.

5) Set i to 0.

6) While i < arrival\_rate do step 7.

7) Add the results of custer's make\_order method to cust\_pay.

8) Pop the customer from the queue.

9) Return cust\_pay.

10) Return 0.

class Customers

MEMBERS:

-Main\_Course\_Menu m

-Appetizer\_Menu a

-Drink\_Menu d

-Dessert\_Menu ds

METHODS:

Name: Constructor

Desc: Default Constructor.

Params: None

Return: None

Name: Constructor

Desc:

Params: int arrival\_rate

Return: None

1) Calls make\_order.

Name: make\_order()

Desc: Places an order.

Params: None

Return: double

1) Generate a random number from 0 to 9.

2) Create amt and set it to 0.

3) If the number is between 0 and 10, call set menu, and set amt to the return from order from the main course.

4) If the number is between 3 and 10, repeat step 3 for drink.

5) If the number is between 6 and 10, repeat step 3 for appetizer.

6) If the number is between 8 and 10, repeat step 3 for dessert.

7) Return amt.

class Menu

METHODS:

Name: Constructor

Desc: Default constructor

Params: None

Return: None

Name: order()

Desc: Virtual function for polymorphism and inheritance.

Params: None

return: double

Name: set\_menu()

Desc: Virtual function for polymorphism and inheritance.

Params: None

return: void

class [Main\_Course\_Menu, Drink\_Menu, Dessert\_Menu, Appetizer\_Menu] : Menu

MEMBERS:

-map<string, double> main\_menu\_ma;

-map<string,double>::iterator it\_m

METHODS:

Name: Constructor

Desc: Default constructor.

Params: None

return: None

Name: order()

Desc: Makes an order based on random number generation and maps.

Params: None

return: double

1) Generate a random number between 0 and 4.

2) If the number == any of the number find the order in the map and return the price.

3) Return 0.

Name: set\_menu()

Desc: Sets up the menu in the map.

Params: None

return: void

1) Sets up the map by making pairs of food and price.

class Staff

METHODS:

Name: Constructor

Desc: default constructor.

Params: None

Return: None

Name: do\_work()

Desc: Virtual function for polymorphism and inheritance.

Params: int clock

Return: void

class [Cook, Servers] : Staff

MEMBERS:

-int num\_staff

-int max\_time

-int min\_time

METHODS:

Name: Constructor

Desc: Default constructor

Params: Non

Return: None

Name: Constructor

Desc: Constructor

Params: unsigned short int max, unsigned short int min

Return: None

1) Sets max and min time to the parameters.

2) Set number of staff to 1.

Name: do\_work()

Desc: Controls the time in function.

Params: int clock

Return: void

1) If the clock - min\_tim > max\_time, do steps 2-3.

2) Set the clock to the min\_time.

3) While clock is less than or equal to max\_time, increase clock.

class Finances

MEMBERS:

-double profits

-double expenses

-double balance = 0;

METHODS:

Name: Constructor

Desc: Default constructor

Params: None

Return: None

Name: getBalance()

Desc: Returns the balance.

Params: None

Return: double balance

1) Return balance.

Name: setBalance()

Desc: Sets the balance.

Params: double balance

Return: void

1) Sets balance to the parameter.

Name: getProfits()

Desc: Returns the profits.

Params: None

Return: double profits

1) Return profits.

Name: setProfits()

Desc: Sets the profits.

Params: double profit

Return: void

1) Sets profits to the parameter.

Name: getExpenses()

Desc: Returns the expenses

Params: None

Return: double expenses

1) Return expenses.

Name: setExpenses()

Desc: Sets the expenses.

Params: double expense

Return: void

1) Sets expenses to the parameter.

Name: making\_a\_profit()

Desc: Adds to the balance.

Params: double days\_profits

Return: void

1) Adds days\_profits to the balance.

Name: paying\_expenses()

Desc: Subtracts from the balance.

Params: double days\_expenses

Return: void

1) Subtracts days expenses from the balance.

util.cpp

Name: go\_again()

Desc: Asks if the user wants to go again.

Params: None

Return: bool

1) Ask the user if they want to go again, read int the response.

2) Return strcmp(resp, "yes") == 0.