CAIRO UNIVERSITY

FACULTY OF ENGINEERING

COMPUTER ENGINEERING DEPARTMENT

FIRST YEAR

RESTAURANT SYSTEM

Project Report - Team 10

Ahmed Gamal Abdel-Samei Ibrahim

Sec: 1 | BN: 03

Ahmed Hamdy Muhammad Nossir

Sec: 1 | BN: 04

Gaser Ashraf Fayez Atris

Sec: 1 | BN: 23

Hossam Muhammad Saeed Mostafa

Sec: 1 | BN: 26

June 2020

In partial fulfillment of the requirements of CMP1040

Cairo University, Faculty of Engineering
Computer Engineering Department
Data Structures and Algorithms
CMP102/CMPN102

Data Structures and Algorithms
Final Assessment Report

Team Name: Team 10	**Number of members: 4**

Email:ahmaddjamal01@gmail.com

Section1: Ahmed Gamal Abdelsamie Ibrahim, 103

ReturnOutput()

Member of: Class Restaurant
Inputs: _
Returns:
Outputs a file with the required statistics
Called By:
- Restaurant::simulation(PROG_MODE Mode)

Calls:
- Restaurant::AverageServingTime()
- Restaurant::AverageWaitingTime()
- Order::GetAT()
- Order::GetID()
- Order::getFinishTime()
- Order::getServTime()
- Order::getWaitTime()
- Restaurant::sortFinishedOrders(LinkedList<Order*>&FinishOrders)

Function Logic description:
This function produces the finish time, ID, arrival time, waiting time, and serving time of each order in an output file. Then, produces total number of orders, number of each type, total number of cooks, number of each type, number of injuries, average waiting and serving times, and percentage of auto-promoted orders.

AverageWaitingTime()

Member of: Class Restaurant
Inputs:_
Returns: average waiting time of all finished orders

Called By:
- Restaurant::ReturnOutput()

Calls:
- Order::getWaitTime()

Function Logic description:
Traverses the finished orders list, adding up all the waiting times, then dividing their sum by the number of orders

AverageServingTime()

Member of: Class Restaurant
Inputs:_
Returns: average serving time of all finished orders

Called By:
- Restaurant::ReturnOutput()

Calls:
- Order::getServTime()

Function Logic description:
Traverses the finished orders list, adding up all the serving times, then dividing their sum by the number of orders

sortFinishedOrders(LinkedList<Order*>& FinishOrders)

Member of: Class Restaurant
Inputs: Finished orders linked list by reference
Returns: void, sorts the list according to finishing and serving times

Called By:
- Restaurant::ReturnOutput()

Calls:
- Order::getFinishTime()
- Order::getServTime()

Function Logic description:
Traverses the finished orders list, checks if consecutive orders have the same finishing time. If so, sorts them in ascending order according to serving time. Note that finished orders are already entered in ascending order.

Section2: Ahmed Hamdy Muhammad Nossir, 104

CheckIfCooksTakeBreak(int currStep)

Member of: Class Restaurant
Inputs: int currStep
Returns: void

Called By:
- Restaurant::simulation(PROG_MODE Mode)

Calls:
- Cook:: setTimeBackToAvaList(int t)
- Cook:: isInj()
- Cook::getServingOrder()
- Restaurant:: RemoveAndGetCookByIdFromANC(int t)

Function Logic description:
This function traverse each type of cooks and checks if the cook is injured or not he should take his rest or break

CheckIfCooksBackFromBreak (int currStep)

Member of: Class Restaurant
Inputs: int currStep
Returns: void

Called By:
- Restaurant::simulation(PROG_MODE Mode)

Calls:
- Cook:: setTimeBackToAvaList(int t)
- Cook::getTimeBackToAvaList()
- Cook:: isInj()
- Cook::getServingOrder()
- Restaurant:: RemoveAndGetCookByIdFromANC(int t)

Function Logic description:
This function traverse each type of cooks and checks if the cook is injured or not he should take his rest or break

checkAutoPromote (int currStep)

Member of: Class Restaurant
Inputs: int currStep
Returns: void

Called By:
- Restaurant::simulation(PROG_MODE Mode)

Calls:
- Order:: GetID(Order)
- Order:: SetType(ORD_TYPE)
- Restaurant:: DeleteOrderID(int id)

Function Logic description:

This function traverse each normal order and promotes it to VIP if its waiting time reaches AutoPromotion limit.

Section3: Gaser Ashraf Fayez Atris, 123

simulation(PROG_MODE Mode)

Member of: Class Restaurant
Inputs: PROG_MODE Mode
Returns: void

Called By:
- Restaurant:: RunSimulation()

Calls:
- GUI::AddToDrawingList(Cook* pC)
- GUI::AddToDrawingList(Order* pOrd)
- Restaurant::CheckIfCooksBackFromBreak(int currStep)
- Restaurant::CheckIfCooksTakeBreak(int currStep)
- Restaurant::ExecuteEvents(int CurrentTimeStep)
- Restaurant::HandleWithFinshedOrders(int currtime)
- Restaurant::HurtCooks (Node<Cook*>* pVIP, Node<Cook*>* pVGN, Node<Cook*>* pNRM, int CurrentTimeStep)
- Restaurant::checkAutoPromote(int currStep)
- Restaurant::addToServeOrder(Order* o)
- GUI::UpdateInterface()
- GUI::PrintMessage(string* msg, int numOfStrings) const
- GUI::PrintMessage(string msg) const
- GUI::ResetDrawingList()
- GUI::waitForClick() const
- Restaurant::ReturnOutput()

Function Logic description:

This function checks if there are cooks need to take a break or rest then check the cooks finished their break or rest time and needed to be back to the available cooks then traverse on all the normal orders and if the waiting time is greater than the Auto Promotion Level then that order should be promoted to a VIP order, after that it's time to assign the orders to the available cooks first the VIP orders should be assigned first and to a VIP cook and if there are not available vip cooks then to be assigned to an available normal cook and if there isn't then to be assigned to an available vegan cook, then traversing on the Vegan orders which should be assigned only to the available vegan cooks and if there isn't then the order should wait until there are available vegan cooks, after VIP and Vegan now to assign the normal cooks to the available normal and vip cooks, respectively.
In every TS we generate a random probability and if this number is less

than or equal to the Injury Probability then a busy cook should be injured his speed would be the half then take a rest after finished his order
After that we handle the finished orders by moving them from the list of the serving orders to the finished orders and the busy cooks become available or go break or rest
Also in every TS we draw the waiting, in-serving and done orders and as well as the available cooks
The last part controls The Shape of the Interface depending on the Proj_Mode the user chose: Silent Mode calculates everything instantly, Interactive mode lets you see each TS in action and on your own pace, Step by Step mode lets you see each TS with a constant delay between each, finally this function call “ReturnOutput” to create the outfile with the stats.

HandleWithFinshedOrders (int currStep)

Member of: Class Restaurant
Inputs: int currStep
Returns: void

Called By:
- Restaurant::simulation(PROG_MODE Mode)

Calls:
- Cook::getOrderThatWorkedAt()
- Cook:: getTimeFinshOrder()
- Cokk:: getOrderThatWorkedAt()
- Cook:: setTimeBackToAvaList(int t)
- Cook:: isInj()
- Cook:: setTimeFinshOrder(int t)
- Cook:: setOrderThatWorkedAt(Order* o)
- Cook:: setServingOrder(int s)
- Cook::getServingOrder()
- Restaurant:: RemoveAndGetOrderByIdFromServeOrderList(int id)
- Restaurant:: RemoveAndGetCookByIdFromANC(int t)

Function Logic description:
This function traverses on each cook and if it’s found that the assigned order is finished then it will be added to the list of the finished orders each to its type.

Section4: Hossam Muhammad Saeed Mostafa, 126

HurtCooks (Node<Cook*>* pVIP, Node<Cook*>* pVGN, Node<Cook*>* pNRM, int CurrentTimeStep)

Member of: Class Restaurant
Inputs: `Node<Cook*>* pVIP, Node<Cook*>* pVGN, Node<Cook*>* pNRM, int CurrentTimeStep`
Returns: bool

Called By:
- Restaurant::simulation(PROG_MODE Mode)

Calls:
- Order::getAt()
- Order::getsize()
- Order::getfinishtime()
- Order::getservetime()
- Order::getwaittime()
- Order::setfinishtime()
- Cook:: getOrderThatWorkedAt()
- Cook:: setInj()
- Cook:: setTimeFinshOrder(int t)
- Cook:: getSpeed()
- Cook:: getInjSpeed()

Function Logic description:
This function is called if the probability randomized needs to injure some cook. It resets the finishing and serving times of order after comparing the 3 types to find the first busy cook.

RemoveAndGetCookByIdFromANC(int id);

RemoveAndGetCookByIdFromAGC(int id); RemoveAndGetCookByIdFromAVC(int id)

Member of: Class Restaurant
Inputs: int id
Returns: Node<Cook*>*

Called By:

- Restaurant::CheckIfCooksTakeBreak(int currStep)

Calls:

- LinkedList<Cook*>::RemoveCookFromListWithOutDelete(int id)

Function Logic description:

These function takes an id and removes the cook with the passed id from the Available Cooks for each type

RemoveAndGetCookByIdFromBNC(int id); RemoveAndGetCookByIdFromBGC(int id); RemoveAndGetCookByIdFromBVC(int id)

Member of: Class Restaurant
Inputs: int id
Returns: Node<Cook*>*

Called By:

- Restaurant::CheckIfCooksBackFromBreak(int currStep)
- Restaurant::HandleWithFinshedOrders(int currtime)

Calls:

- LinkedList<Cook*>::RemoveCookFromListWithOutDelete(int id)

Function Logic description:

These function takes an id and removes the cook with the passed id from the Busy Cooks for each type

Tests:

The 8 sample tests we generated cover most of the different scenarios. A test does not have random parameters if (minSpeeds == maxSpeeds && minBreaks == maxBreaks && Injury Prob. == 0). The following table summarizes the properties of each sample test :

Test	# Events	Cancel	Prom	Random	Injury Prob.	# Cooks
1.txt	4	Yes	No	Yes	0.25	3
2.txt	15	No	No	Yes	0.5	3
3.txt	32	Yes	Yes	No	0	8
4.txt	12	No	No	No	0	6
5.txt	20	Yes	Yes	Yes	0.4	4
6.txt	10	Yes	Yes	Yes	0.7	10
7.txt	30	Yes	Yes	No	0	5