

Data Structures and Algorithms

Final Assessment Report

Submitted to:

Dr/Magda Fayek

Submitted by:

Hazem Kasim

Ahmed Mohamed Ibrahim

Peter Michael

Ahmed Mahmoud Mohamed

Spring, 2020

**Section: 1 Name: Hazem Kasim BN: 24**

**Member Of:** Class PromoteEvent

**Function:** PromoteEvent (the constructor)

**Inputs:**

* eTime: the timestep when this event takes place.
* oID: order of the event.
* m: the money paid for the order to be promoted and in case if it equals zero, this means that this order is auto promoted.

**Called by:**

* This is the initialization function of the class, so basically it gets called when we create an instance of the class or in the restaurant terminology, when an order needs to promoted and this happens inside the Resturant::Add\_Events() function.

**Function logic description:**

This class is inherited from class Event, which serves as the base class for all coming events, each class inherits from Events needs to handle a certain type of events and for this one, it’s the events to be promoted due to extra-money paid or that the order of type normal had waited long enough, so it gets an auto promotion

**Section: 1 Name: Hazem Kasim BN: 24**

**Member Of:** PromoteEvent

**Function:** Execute

**Inputs:**

* pRest: a pointer to object of type Restaurant.

**Return Type:** void

**Called by:**

* Resturant::ExecuteEvents()

**Calls:**

* pRest->Promote\_Order()

**Function logic description:**

This function is a polymorphic function and for this class, it takes the order information which are the order arrival time, the order ID and the extra money and then promotes that order to a VIP order

**Section: 1 Name: Hazem Kasim BN: 24**

**Member Of:** Class CancelEvent

**Function:** Execute

**Inputs:**

* pRest: a pointer to object of type Restaurant.

**Return type:** void

**Called by:**

* Resturant::ExecuteEvents()

**Calls:**

* pRest->Delete\_Order()

**Function logic description:**

This function is a polymorphic function and for this class it takes the order of the ID then it deletes it from the orders list if it exists.

**Section:1 Name: Ahmed Mohamed Ibrahim BN:8**

**Member Of:** Class Restaurant

**Function:** ServeVIPOrder

**Inputs:**

* pOrder: a pointer to object of type Order.
* t: the timestep at which this VIP order arrives.

**Return type:** Boolean

**Called by:**

* Resturant::ChooseCook()

**Calls:**

* Cook::getInjured()
* Cook::getBreak()
* Cook::getBusy()
* Order::getStatus()
* Cook::serveNewOrder()
* Restaurant::Assign()

**Function logic description:**

This function is about finding the suitable cook specialized in VIP orders for that coming VIP order. It performs some checks to pick the cook that is not busy, not injured, not in break, so he will be able to take the waiting order. If no cook is available then return false.

**Section:1 Name: Ahmed Mohamed Ibrahim BN:8**

**Member Of:** Class Restaurant

**Function:** ServeVeganOrder

**Inputs:**

* pOrder: a pointer to object of type Order.
* t: the timestep at which this VIP order arrives.

**Return type:** Boolean

**Called by:**

* Resturant::ChooseCook()

**Calls:**

* Cook::getInjured()
* Cook::getBreak()
* Cook::getBusy()
* Order::getStatus()
* Cook::serveNewOrder()
* Restaurant::Assign()

**Function logic description:**

This function is about finding the suitable cook specialized in Vegan orders for that coming Vegan order. It performs some checks to pick the cook that is not busy, not injured, not in break, so he will be able to take the waiting order. If no cook is available then return false.

**Section:1 Name: Ahmed Mohamed Ibrahim BN:8**

**Member Of:** Class Restaurant

**Function:** ServeNormalOrder

**Inputs:**

* pOrder: a pointer to object of type Order.
* t: the timestep at which this VIP order arrives.

**Return type:** Boolean

**Called by:**

* Resturant::ChooseCook()

**Calls:**

* Cook::getInjured()
* Cook::getBreak()
* Cook::getBusy()
* Order::getStatus()
* Cook::serveNewOrder()
* Restaurant::Assign()

**Function logic description:**

This function is about finding the suitable cook specialized in Normal orders for that coming Normal order. It performs some checks to pick the cook that is not busy, not injured, not in break, so he will be able to take the waiting order. If no cook is available then return false.

**Section:1 Name: Ahmed Mohamed Ibrahim BN:8**

**Member Of:** Class Restaurant

**Function:** ChooseCook

**Inputs:**

* pOrder: a pointer to object of type Order.
* t: the timestep at which this VIP order arrives.

**Return type:** Boolean

**Called by:**

* Resturant::CheckOrder()
* Resturant::CheckUrgentOrders()

**Calls:**

* Order::getStatus()
* Order::getArrival()
* Order::GetType()
* Restaurant::ServerVIPOrde()
* Restaurant::ServerVeganOrde()
* Restaurant::ServerNormalOrde()

**Function logic description:**

This function is like the manager who gives each waiting order to the cook that’s available. To achieve that, this function check first if the orders are waiting orders or not. If they are still waiting then it checks the order type to give it to the suitable available cook.

**Section: 1 Name: Hazem Kasim BN: 24**

**Member Of:** Class Restaurant

**Function :** FillDrawingList

**Inputs:** Doesn’t take any arguments

**Return type:** void

**Called by:**

* Restaurant::RunSimulation()

**Calls:**

* FillDrawingList()

**Function logic description:**

Send a pointer for every cook & order to GUI::AddToDrawingList(ptr) where it is drawn on the gui with the proper position & color.

**Section:1 Name: Ahmed Mahmoud Mohamed BN:12**

**Member Of:** Class Cook

**Function:** Take Break

**Inputs:** Doesn’t take any arguments

**Return type:** Void

**Called by:**

* void Cook::serveCurenntOrder()
* void Restaurant::CooksInBreak()

**Calls:**

* void Cook::setRest(bool b)

**Function logic description:**

This function is responsible to handle breaks of all types and statuses of all cooks. When called by a cook it checks if he is injured or not and upon that it starts to act. If he is injured then it checks if he is busy, at rest, or finished his rest, and if he is not injured then it checks if he needs a break, already in break, or finished break and handles each case.

**Section: 1 Name: Hazem Kasim BN: 24**

**Member Of:** Class GUI

**Function:**  PrintMessage

**Inputs:**

* msg : the string needed to be typed
* num : fixed number to print fixed outputs as number of cooks etc. In their right positions.
* N : changeable number to print orders assigned to cooks depending on the number of orders being assigned in that time step.

**Return type:** void

**Called by:**

* Restaurant::RunSimulation()
* Restaurant::Num\_WT\_Orders()
* Restaurant::Num\_Cooks\_Able()
* Restaurant::Assign(Cook\* pC, Order\* pO, int n)
* Restaurant::Served\_num()

**Calls:**

* pGUI->PrintMessage(string,int,int)

**Function logic description:**

The function send a string and 2 numbers this 2 numbers enter in two switch cases each case has a specific dimension on the status bar ,so after determining the dimension, the string that was sent is written there.

**Section: 1 Name: Peter Michael BN: 21**

**Member Of:** Class Restaurant

**Function:** EventsOrdersFound

**Inputs:** Doesn’t take any arguments

**Return type:** void

**Called by:**

* Restaurant::RunSimulation()

**Calls:**

* EventsOrdersFound()

**Function logic description:**

Check if there are any orders left in serve or arrival status before exiting the simulation.

**Section: 1 Name: Hazem Kasim BN: 24**

**Member Of:** Class Restaurant

**Function :** Delete\_Order

**Inputs:**

* ID : The ID of order that will be deleted

**Return type:** void

**Called by:**

* CancelEvent::Execute(Restaurant\* pRest)

**Calls:**

* pRest->Delete\_Order(this->getOrderID());

**Function logic description:**

Calls Delete function in the **Linkedlist** class and send to it the ID of the order to be deleted and removed from the order list.

**Section:1 Name: Ahmed Mohamed Ibrahim BN:8**

**Member Of:** Class Restaurant

**Function:** Serving\_Transfer

**Inputs:** Doesn’t take any arguments

**Return type:** void

**Called by:**

* Restaurant::RunSimulation()

**Calls:**

* Serving\_Transfer()

**Function logic description:**

Check after each time step for all types of orders if they are in service now it remove them from waiting/arrival queue and add them to the InServiceOrders queue.

**Section:1 Name: Ahmed Mohamed Ibrahim BN:8**

**Member Of:** Class Restaurant

**Function:** Finished\_Transfer

**Inputs:** Doesn’t take any arguments

**Return type:** void

**Called by:**

* Restaurant::RunSimulation()

**Calls:**

* Finished\_Transfer()

**Function logic description:**

Check after each time step for all orders in service if they are finished, it removes them from InServiceOrders queue and add them to the FinishedOrders queue.

**Section:1 Name: Ahmed Mahmoud Mohamed BN:12**

**Member Of:** Class Restaurant

**Function :** CooksInBreak

**Inputs:** Doesn’t take any arguments

**Return type:** void

**Called by:**

* Restaurant::RunSimulation()

**Calls:**

* CooksInBreak()

**Function logic description:**

Check after each time step for all cooks if they are in break it calls **pCook->TakeBreak()** to increase the time taken by them in break by 1.

**Section:1 Name: Ahmed Mahmoud Mohamed BN:12**

**Member Of:** Class Restaurant

**Function:** Cook\_Dute

**Inputs:** Doesn’t take any arguments

**Return type:** void

**Called by:**

* Restaurant::RunSimulation()

**Calls:**

* Cook\_Dute()

**Function logic description:**

Check after each time step for all cooks if they are assigned to an order it calls **pCook->serveCurenntOrder()** which make the cook prepare a number of dishes in the order assigned to him depending on it’s speed.

**Section:1 Name: Ahmed Mohamed Ibrahim BN:8**

**Member Of:** Class Restaurant

**Function :** Increase\_WT

**Inputs:** Doesn’t take any arguments

**Return type:** void

**Called by:**

* Restaurant::RunSimulation()

**Calls:**

* Increase\_WT()

**Function logic description:**

For each time step it increases the waiting time of all orders in the waiting lists by 1.

**Section: 1 Name: Hazem Kasim BN: 24**

**Member Of:** Class Restaurant

**Function:** AutoP\_Check

**Inputs:** Doesn’t take any arguments

**Return type:** void

**Called by:**

* Restaurant::RunSimulation()

**Calls:**

* AutoP\_Check()

**Function logic description:**

Checks after each time step if any normal order waiting time exceeded the auto promotion time it promotes that normal order to a vip order by calling **Promote\_Order(pOrder->GetID(), pOrder->getArrival(), 0)** .”extra\_money is 0 because it’s auto promoted”

**Section: 1 Name: Peter Michael BN: 21**

**Member Of:** Class Restaurant

**Function :** Num\_WT\_Orders

**Inputs:** Doesn’t take any arguments

**Return type:** void

**Called by:**

* Restaurant::RunSimulation()

**Calls:**

* Num\_WT\_Orders()

**Function logic description:**

This function print the number of waiting orders of each type at each time step.

**Section: 1 Name: Peter Michael BN: 21**

**Member Of:** Class Restaurant

**Function :** Num\_Cooks\_Able

**Inputs:** Doesn’t take any arguments

**Return type:** void

**Called by:**

* Restaurant::RunSimulation()

**Calls:**

* Num\_Cooks\_Able()

**Function logic description:**

This function print the number of available cooks of each type at each time step.

**Section: 1 Name: Hazem Kasim BN: 24**

**Member Of:** Class Restaurant

**Function :** Assign

**Inputs:**

* pC: it is a pointer to the cook assigned to the order.
* pO: it is a pointer to the order being assigned to the cook.
* n: an integer refers to the dimension where the message of assigning will be printed on the status bar.

**Return type:** void

**Called by:**

* Restaurant::ServeVIPOrder(Order\*& pOrder, const int& t)
* Restaurant::ServeVeganOrder(Order\* pOrder, const int& t)
* Restaurant::ServeNormalOrder(Order\* pOrder, const int& t)

**Calls:**

* Assign(Cook\*, Order\*, int)

**Function logic description:**

This function print a message on the status bar if there is any order assigned to a cook in that time step and the message depends on the **int n** ”message position on status bar” & on order/cook type , to print a proper message giving information about the ID & type of order/cook being assigned.

**Section: 1 Name: Peter Michael BN: 21**

**Member Of:** Class Restaurant

**Function :** Promote\_Order

**Inputs:**

* ID : The ID of order that will be promoted
* arrival\_time : the time when the promote request arrived
* extra\_money : the extra money paid to promote the order to a vip order

**Return type:** void

**Called by:**

* PromoteEvent::Execute(Restaurant\* pRest)

**Calls:**

* pRest->Promote\_Order(this->getOrderID(), this->getEventTime(), Extra\_Money);

**Function logic description:**

Change the order type to vip and add the extra money to the money of order and change it’s arrival time to the new arrival time then add it to vip queue and delete it from normal list orders.

**Section: 1 Name: Peter Michael BN: 21**

**Member Of:** Class Restaurant

**Function:** Served\_num

**Inputs:** Doesn’t take any arguments

**Return type:** void

**Called by:**

* Restaurant::RunSimulation()

**Calls:**

* Served\_num()

**Function logic description:**

This function prints an up to date message of the total number of finished normal/vegan/vip orders.

**Section:1 Name: Ahmed Mohamed Ibrahim BN:8**

**Member Of:** Class Restaurant

**Function:** Output\_File

**Inputs:**

* name: The string that output txt document will be named with.

**Return type:** void

**Called by:**

* Restaurant::RunSimulation()

**Calls:**

* Output\_File(string)

**Function logic description:**

This function is called at the end of each simulation where it creates a txt document and writes in it the Finish\_Time , ID , Arrival\_Time , Waiting\_Time ,and Serving\_Time of each order , then it writes the total number of normal/vegan/vip orders done , then it writes the total number of normal/vegan/vip/injured cooks , then it writes the average time of waiting & serving time , then it writes the total number of Urgent orders & the orders which was auto promoted.

**Section: 1 Name: Peter Michael BN: 21**

**Member Of:** Class Cook

**Function:** serveNewOrder

**Inputs:**

* Order\*& ptr: pointer to the sent order
* const int& t: current time step

**Return type:** Void

**Called by:**

* bool Restaurant::ServeVIPOrder(Order\*& pOrder, const int& t)
* bool Restaurant::ServeVeganOrder(Order\* pOrder, const int& t)
* bool Restaurant::ServeNormalOrder(Order\* pOrder, const int& t)

**Calls:**

* void Cook::setBusy(bool b)
* void Order::setStatus(ORD\_STATUS s)
* void Cook::serveCurenntOrder()

**Function logic description:**

This function is called when a cook is assigned a new order. First it makes the cook bust then it sets the status of the assigned order to serving and finally it calls function serveCurenntOrder().

**Section: 1 Name: Peter Michael BN: 21**

**Member Of:** Class Cook

**Function:** serveCurenntOrder

**Inputs:** Doesn’t take any arguments

**Return type:** Void

**Called by:**

* void Cook::serveNewOrder(Order\*& ptr, const int& t
* void Restaurant::Cook\_Dute()

**Calls:**

* int Order::getSize()
* void Order::setSize(int s)
* void Order::setFinish(int a)
* int Order::getArrival()
* int Order::getWT()
* void Order::setStatus(ORD\_STATUS s)
* void Order::setServ(int a)
* void Cook::setBusy(bool b)
* void Cook::TakeBreak()

**Function logic description:**

This function is responsible for serving all the different kind of orders. It checks if the cook has finished the order he is assigned to if so, it starts to decrease the size of the order where the decreased size is the speed of the cook. And if the cook has already finished the order it makes the cook free and calls the functions mentioned above to handle what happens when an order is finished.

**Section:1 Name: Ahmed Mahmoud Mohamed BN:12**

**Member Of:** Class Restaurant

**Function:** cutbreaks

**Inputs:** Doesn’t take any arguments

**Return type:** Bool

**Called by:**

* void Restaurant::CheckUrgentOrders(int t)

**Calls:**

* bool Cook::getBreak()
* void Cook::cutbreak()

**Function logic description:**

this function is responsible for searching for cooks in break to cut their breaks and if it doesn’t found cooks in break, it looks for cooks who are in rest and tries to do the same thing if it doesn’t found any of the previous it returns false meaning all cooks are busy in the current time step.

**Section:1 Name: Ahmed Mahmoud Mohamed BN:12**

**Member Of:** Class Cook

**Function:** cutbreak

**Inputs:** Doesn’t take any arguments

**Return type:** Void

**Called by:**

* bool Restaurant::cutbreaks()

**Calls:**

* void Cook::setRest(bool b)

**Function logic description:**

This functions cuts the break or rest of cooks in break and cooks in rest respectively.

**Section: 1 Name: Peter Michael BN: 21**

**Member Of:** Class Restaurant

**Function:** checkHealth

**Inputs:** Doesn’t take any arguments

**Return type:** Void

**Called by:**

* void Restaurant::RunSimulation()

**Calls:**

* bool Cook::getBusy()
* void Cook::setinjuredbefore()
* void Cook::setInjured(bool b)
* void Cook::setRestTime(int i)

**Function logic description:**

This function is responsible for getting busy cooks injured first it generates a random number to check if any cook will get injured at the current time step then it traverses the cooks’ queues to find the first bust cook and make them get injured. It calls different functions that handles what happens when a cook gets injured like decreasing their speed to its half and so on.

**Section:1 Name: Ahmed Mahmoud Mohamed BN:12**

**Member Of:** Class Restaurant

**Function:** CheckUrgentOrders

**Inputs:**

* int t: current time step

**Return type:** Void

**Called by:**

* void Restaurant::RunSimulation()

**Calls:**

* double Order::getMoney()
* int Order::getSize()
* int Order::getArrival()
* bool Restaurant::ChooseCook(Order\* pOrder, const int& t)
* bool Restaurant::cutbreaks()

**Function logic description:**

This function handles urgent orders. It is responsible for two actions first: checks VIP orders that waited longer than **VIP\_WT** and hence became urgent orders and if it founds any it adds them to the urgent orders queue. Second: if it founds any urgent orders in the urgent orders queue it starts to try to serve them first it looks for free cooks if not found it tries to cut the breaks of the cooks who are either in break the cooks in rest. If all the cooks are busy and no one in beak or rest it waits for the next time step to repeat the previous steps.

**Section:1 Name: Ahmed Mahmoud Mohamed BN:12**

**Member Of:** Class Restaurant

**Function:** Add\_Cooks

**Inputs:**

* int n, int v, int g: the number of cooks for each type
* int SN\_min, int SN\_max: minimum and maximum speeds for normal cooks
* int SG\_min, int SG\_max: minimum and maximum speeds for vegan cooks
* int SV\_min, int SV\_max: minimum and maximum speeds for VIP cooks
* int BN\_min, int BN\_max: minimum and maximum break time for normal cooks
* int BG\_min, int BG\_max: minimum and maximum break time for vegan cooks
* int BV\_min, int BV\_max: minimum and maximum break time for VIP cooks

**Return type:** Void

**Called by:**

* void Restaurant::Output\_File(stri

**Calls:**

* doesn’t call any function related to our logic only (enqueue) and (dequeue) functions and rand function which is already implemented in c++.

**Function logic description:**

This function is responsible for adding the cooks in their queues. It creates number of cooks of each kind according to the numbers sent to it and for each cooks it generates a random speed ranging from the minimum and maximum speeds sent to it also it creates a random break time similarly ranging from the minimum and maximum break times sent to it and finally adds them to their queues.

**Section:1 Name: Ahmed Mahmoud Mohamed BN:12**

**Member Of:** Class Restaurant

**Function:** calcinjuredcooks

**Inputs:** Doesn’t take any arguments

**Return type:** Void

**Called by:**

* void Restaurant::Output\_File(string name)

**Calls:**

* bool Cook::gotinjuredbefore()

**Function logic description:**

This function is responsible for counting the number of injured cooks by traversing cooks’ queues and see who got injured before. We made this function to avoid counting cooks who got injured more than once.