

*Prepared by:*

Ahmed Ihab Yousry Abdel-Fattah

**Sec**: 01 **BN**: 02

Donia Abdel-Fattah Abdel-Karim Meselhi

**Sec**: 01 **BN**: 29

Raghad Khaled Abdel-Hai Abo-Khadra

**Sec**: 01 **BN**: 31

Abeer Hussein Mohamed

**Sec**: 01 **BN**: 40

**June**

2020

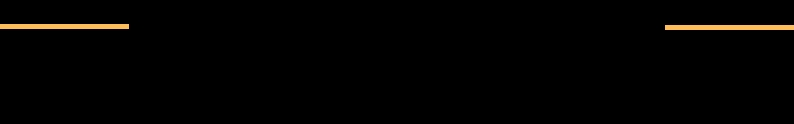
*Supervised by:*

Dr. Magda Fayek

Eng. Ali Elsedeek

Eng. Eman Hossam

**Restaurant Management**



**CONTENTS**

**P.No**



2 Section 1

**CheckUrgentOrders………………2**

**GetCooksFor\_Urgent\_VIP………3**

**Serve\_Urgent\_VIP………………**..**4**

**` Restaurant\_Modes……………....5**

5 Section 2:

Serve**…………………..…..………...7**

**Promote……………………………...8**

**Health\_Emergency…………….….9**

**getfromRestCookQ.……………….10**

9 Section 3

**Executepromotion………….………11**

**serve\_VIP\_order…………….…….12**

**serve\_Vegan\_order……….………13**

**serve\_Normal\_order……….….…14**

**getfrombusyCook………………….15**

**getfromBreakCookQ………………16**

**getfromInServingQ………………..17**

**promoteorder………………………..17**

**Cancellorder………………………….18**

19 Section 4

**Output File Function………….……19**

**Test Cases……….…………….……..20**

21 Appendix

**Section1: Ahmed Ihab Yousry ,102**

**CheckUrgentOrders**

**Member of:** Class Restaurant

**Inputs:** CurrentTimeStep: The time step which the program execute

**Returns:** This is Void function does not return anything

**Called by:**

* Restaurant::RunSimulation()
* Restaurant::Restaurant\_Modes(int Mode)
* Restaurant::Serve\_Urgent\_VIP(int CurrrentTimeStep)

**Calls:** No thing

**Function Logic Description:**

This function turns the VIP orders Queue into an array by using the function toArray that is implemented in PriorityQueue , then it loops on the array and check every VIP order whether its waiting time exceeded VIP\_WT or not, if it exceeded it will become an **Urgent Order** and it will be pushed to a new Queue that is Pointer-to-Pointer that called **QUrgentOrders** and increment the number of Urgent orders to be able to print them later.

**GetCooksFor\_Urgent\_VIP**

**Member of:** Class Restaurant

**Inputs:** CurrentTimeStep: The time step which the program execute

**Returns:** Returns a Boolean variable called found that indicates that if the function found a cook that can serve the urgent order or not.

**Called by:**

* Restaurant::RunSimulation()
* Restaurant::Restaurant\_Modes(int Mode)
* Restaurant::Serve\_Urgent\_VIP(int CurrrentTimeStep)

**Calls:** No thing

**Function Logic Description:**

This Function Job is to get a cook to serve the urgent order , first it checks if there is urgent orders by make a peak on QUregentOrders then if it found an urgent order it checks first if all cooks queues are empty, if all of them empty it checks on the cooks that are in break if it found one it removes the cook from break queue and return it to suitable queue depends on its type, but if there is no cooks in break it checks for the cooks that are in Rest and if it found one it do the same and return the cook to its suitable list and make bool variable found equal true , but if from first there are cooks in queues it only make found variable equal true.

**Serve\_Urgent\_VIP**

**Member of:** Class Restaurant

**Inputs:** CurrentTimeStep: The time step which the program execute

**Returns:** This is Void function does not return anything

**Called by:**

* Restaurant::RunSimulation()
* Restaurant::Restaurant\_Modes(int Mode)

**Calls:**

* Restaurant::CheckUrgentOrders(CurrentTimeStep);
* Restaurant::GetCooksFor\_Urgent\_VIP(CurrentTimeStep);

**Function Logic Description**

This function job is serving the urgent orders if found, first it calls two functions as it mentioned above and it stores the returned value from GetCooksFor\_Urgent\_VIP into a bool variable called flag if this flag equals true then there is an available cook that can serve the urgent order, then it checks if the available cook is VIP or Normal or Vegan and serve the order to the cook as the serving logic mentioned in serving functions, but if the flag equals false it means that there is no cook can serve the order so it do nothing.

**Restaurant\_Modes**

**Member of:** Class Restaurant

**Inputs:** it takes an integer input that’s indicate which mode will run.

**Returns:** This is Void function does not return anything.

**Called by:**

* Restaurant::RunSimulation()

**Calls:**

* Restaurant::fileLoading();
* Restaurant::ExecuteEvents(CurrentTimeStep);
* Restaurant::getfromBreakCookQ(CurrentTimeStep);
* Restaurant::getfrombusyCookQ(CurrentTimeStep);
* Restaurant::getfromInServingQ(CurrentTimeStep);
* Restaurant::getfromRestCookQ(CurrentTimeStep);
* Restaurant::Serve\_Urgent\_VIP(CurrentTimeStep);
* Restaurant::Executepromotion(CurrentTimeStep);
* Restaurant::serve\_VIP\_orders(CurrentTimeStep);
* Restaurant::serve\_Vegan\_orders(CurrentTimeStep);
* Restaurant::serve\_Normal\_orders(CurrentTimeStep);
* Restaurant::FillDrawingList();
* Restaurant::outputFileLoading();

and **GUI** Functions:

* GUI::PrintMessage(“ ”);
* GUI::waitForClick();
* GUI::ClearStatusBar();
* GUI::UpdateInterface();
* GUI::ResetDrawingList();

**Function Logic Description**

This function is responsible to simulate the program depending on the chosen mode by the user, first it checks which mode that user entered 1 or 2 or 3 by if else statements, then it calls file loading function to read inputs from text file and calls execute function to execute the entered events and put every event to its suitable list then it checks if there is cooks finished their break or rest duration to get them back to available list and also checks if there is a cook finished serving an order to make him available again, after that it starts to serve the orders by the following sequence Urgent->VIP->Vegan->Normal but before serving VIP orders it checks if there is normal order should be promoted or not to serve it with the VIP orders, then it prints all required information to the GUI window as the document mentioned and it waits for a click or for 1 second depend on the chosen mode and repeats this operation until all queue are empty. After all orders serving finished if there is cooks in break or rest it wait until they become available again.

# Section2: Donia Abdel-Fattah Abdel Kariem, 129

**Serve**

**Member** **of**: Class Order

**Inputs**:

crrTS: current time step

**Returns**:

None

**Called** **By**:

* Restaurant::RunSimulation()
* Restaurant::serve\_Normal\_orders(int CurrentTimeStep)
* Restaurant::serve\_VIP\_orders(int CurrentTimeStep)
* Restaurant::serve\_Vegan\_orders(int CurrentTimeStep)

**Calls**:

* Order:: setServTime(crrTS)
* Order:: setWaitTime()
* Order:: setFinishTime()
* Order:: setStatus(SRV)

**Function** **Logic** **description**:

It takes the Current time step as an argument and sets the serving time with it, after that it sets the waiting time which is equal to (serving time step – arrival time step), it sets the finish time depending on serving interval, which is calculated in serve\_orders functions, and finally it changes the status of the order to be SRVG instead of WAITING.

**Promote**

**Member** **of**: Class Order

**Inputs**:

* added money: excess money paid by normal order to be promoted to VIP.

**Returns**:

None

**Called** **By**:

* Restaurant::RunSimulation( )
* Restaurant:: Executepromotion(int CurrentTimeStep)
* Restaurant::promoteorder(int Id, double exmoney)

**Calls**:

* Order:: setPriority()

**Function** **Logic** **description**:

* Changes the type of the order from Normal to VIP
* Add the extra money to the total money of the order and update it
* Set the priority of the order as it became VIP and needs to be in priority queue

# Health\_Emergency

**Member** **of**: Class Restaurant

**Inputs**:

* curr\_ts: current time step

**Returns**:

Bool: to indicate if the first cook in busy queue is already injured or not.

**Called** **By**:

* Restaurant::RunSimulation( )
* Restaurant:: Restaurant\_Modes(int Mode)

**Calls**:

* Cook:: Is\_injured()
* Order:: getOrderSize()
* Order::getServTime()
* Cook:: getSpeed()
* Order::setServInt(ST)
* Order:: setFinishTime();
* Order:: getFinishTime();
* Cook:: set\_RstTime(int Finishtime)
* Cook:: injure(true)

**Function** **Logic** **description**:

First of all, it checks if the first busy cook can be injured or not (is already injured or not), if yes, it calculates the following for the first busy cook:

* No. dishes left
* Decreases the speed of the cook to its half
* Change the finish time for both cook and order depending on the new speed
* Dequeue the order and the cook and enqueue them again using the new priority
* Set the rest time step that the cook will finish his rest at
* Make the cook injured by setting the status to (injured)

**getfromRestCookQ**

**Member** **of**: Class Restaurant

**Inputs**:

* CurrentTimeStep

**Returns**:

None

**Called** **By**:

* Restaurant::RunSimulation( )
* Restaurant:: Restaurant\_Modes(int Mode)

**Calls**:

* Cook:: get\_rstTime()
* Order:: Has\_Urg()

**Function** **Logic** **description**:

It Peeks the( cooks\_in\_rest ) queueand Checks if the rest time step is equal to current time step, and if so, it removes the cook from the rest queue, return him to the available cooks again, makes him un-injured, and increases his speed to its normal value.

**Section3: Raghad Khaled Abd El-hay, 131**

**Executepromotion**

**Member** **of**: Class Restaurant

**Inputs**:

* curr\_ts: current time step.

**Returns:**

None.

**Called by:**

* Restaurant::RunSimulation( )
* Restaurant:: Restaurant\_Modes(int Mode)

**Calls:**

* Promote (double & addemoney)

**Function** **Logic** **description**:

It is check if the waiting time of the order in the normal linked list excess Autop time step first delete it from the normal than promote the order finally enqueue in VIP queue.

**serve\_VIP\_orders**

**Member** **of**: Class Restaurant

**Inputs**:

* curr\_ts: current time step.

**Returns:**

None.

**Called by:**

* Restaurant::RunSimulation( )
* Restaurant:: Restaurant\_Modes(int Mode)

**Calls:**

* Order::Serve(CurrentTimeStep)
* Cook::setnumoforderdServed(int num)
* Cook ::assign\_Order(int order)
* Cook::getnumoforderdServed(int num)
* Order::getOrderSize()
* Cook::getSpeed()

**Function** **Logic** **description**:

Try to serve all orders in the VIP queue each order try to assign it to VIP cook (increase number of served orders by this cook by one, calculate the serving interval ST then set order ST, use serve function, dequeue order from QVIP then enqueue it in inserving Queue, enqueue the cook in busy cook queue)

If there is no VIP cook try to assign it to Normal cook if there is No try to assign it to Vegan Cook.

**serve\_Vegan\_orders**

**Member** **of**: Class Restaurant

**Inputs**:

* curr\_ts: current time step.

**Returns:**

None.

**Called by:**

* Restaurant::RunSimulation( )
* Restaurant:: Restaurant\_Modes(int Mode)

**Calls:**

* Order::Serve(CurrentTimeStep)
* Cook::setnumoforderdServed(int num)
* Cook ::assign\_Order(int order)
* Cook::getnumoforderdServed(int num)
* Order::getOrderSize()
* Cook::getSpeed()

**Function** **Logic** **description**:

Try to serve vegan orders by assign it to vegan cooks only(increase number of served orders by this cook by one, calculate the serving interval ST then set order ST, use serve function, dequeue order from QVegan then enqueue it in inserving Queue, enqueue the cook in busy cook queue)

**serve\_Normal\_orders**

**Member** **of**: Class Restaurant

**Inputs**:

* curr\_ts: current time step.

**Returns:**

None.

**Called by:**

* Restaurant::RunSimulation( )
* Restaurant:: Restaurant\_Modes(int Mode)

**Calls:**

* Order::Serve(CurrentTimeStep)
* Cook::setnumoforderdServed(int num)
* Cook ::assign\_Order(int order)
* Cook::getnumoforderdServed(int num)
* Order::getOrderSize()
* Cook::getSpeed()

**Function** **Logic** **description**:

Try to serve normal orders by assign it to normal cook if there is normal cooks we try to assign it to VIP cooks. (increase number of served orders by this cook by one, calculate the serving interval ST then set order ST, use serve function, dequeue order from QVegan then enqueue it in inserving Queue, enqueue the cook in busy cook queue)

**getfrombusyCookQ**

**Member** **of**: Class Restaurant

**Inputs**:

* curr\_ts: current time step.

**Returns:**

None.

**Called by:**

* Restaurant::RunSimulation( )
* Restaurant:: Restaurant\_Modes(int Mode)

**Calls:**

* Order:: injure(bool hurt)
* Cook::setnumoforderdServed(int num)
* Cook :: Give\_Urg(bool urg)
* Cook::getnumoforderdServed(int num)

**Function** **Logic** **description**:

Peek in busy cook queue to check if the cook finished his order if the finished time equal time the current time step and if the cook was injured it dequeue from busy cook then enqueue in the rest one (CooksInRest). If the cook was injured and also Has\_Urg and it served number of orders that should be serve before get its break. frist double its speed and enqueue it from busy queue then enqueue it in break queue.if it has not Urg it enqueue from busy queue then enqueue it in break queue only .Else it should dequeue the cook then enqueue it in its available queue according to its type.

**getfromBreakCookQ**

**Inputs**:

* curr\_ts: current time step.

**Returns:**

None.

**Called by:**

* Restaurant::RunSimulation( )
* Restaurant:: Restaurant\_Modes(int Mode)

**Calls:**

No thing

**Function** **Logic** **description**:

This function job is to dequeue cook that is finished its break then enqueue it in suitable available queue according to its type.

**getfromInServingQ**

**Inputs**:

* curr\_ts: current time step.

**Returns:**

None.

**Called by:**

* Restaurant::RunSimulation( )
* Restaurant:: Restaurant\_Modes(int Mode)

**Calls:**

No thing

**Function** **Logic** **description**:

This function job is to dequeue order that finished from serving and change its state to DONE and enqueue it in finished Queue.

**promoteorder**(int Id, double exmoney)

**Inputs**:

* Id: order ID.
* Exmoney: extra money.

**Returns:**

None.

**Called by:**

* PromotionEvent::Execute(Restaurant\* pRest)

**Calls:**

* Order::Promote(double& addedmoney)

**Function** **Logic** **description**:

The function search for the order in Normal list that has ID (Id that passed to function) then delete the order from the List and call promote function to the order then enqueue it in VIP queue.

**Cancellorder**

**Inputs**:

* Id: order ID.

**Returns:**

None.

**Called by:**

* CancellationEvent::Execute(Restaurant\* pRest)

**Calls:**

* No thing

**Function** **Logic** **description**:

The function search for the order in Normal list that has ID (Id that passed to function) then delete the order from the List

**Section4: Abeer Hussein Mohamed, 141**

**Output File function**

**Member of:** Class Restaurant.

**Inputs:** No inputs.

**Returns:** Void function doesn’t return anything.

**Called by:**

* Restaurant::Restaurant\_Modes()

**Calls:**

GUI functions:

* PrintMessage(“ “)
* GetString()

**Function Logic Description:**

The function first asks the user for the output file name and assign the name to the data member string “OPfilename” , that by which creates the “OutFile” object, an object of the class ostream, to display the output in the file. The function then checks if the output file is opened, the function then prints the output file format as follows: gets all the counters needed which are: “totalwaittime”, “totalServtime”, “ordsCount”, “cooksCount”,“Nserved”, “V served”, “Gserved”, “numNcooks”,”numGcooks”,“numVcooks”,“originalNormOrdCount”, “numAutoPromOrders”. Then the function takes the Finished Orders List and sort it ascendingly based on Finish Time, and if the orders have same finish time, then it sorts them based on Serving Interval. Then the function prints the required data and calculate the required statistics and prints them.

**Test Cases**

1st 3 test cases are simple with low and moderate avg wait, and the last 3 are complex with high avg wait, and that is done by decreasing cooks speed and increasing their breaks time and other numbers manipulation.

**APPENDIX**

***Assumptions made by team members:***

* In Health Emergency, if a cook has an original speed is equal to (1) , then his speed would be equal to zero in integer devision , to prevent that; I assumed that if the cook speed parameter is less than 1, the setter would set it to one. Donya A.Alfattah
* In Health Emergency, a cook cannot be injured again if he is already injured , he can injured again only if he has finished his rest period or took his break in case of urgent orders. . Donya A.Alfattah
* In output file**,** Normal Orders that has been promoted/ auto promoted isn’t counted with normal orders but VIP, also, Normal orders that have been cancelled isn’t counted. Abeer Hussein
* In output file, VIP orders that has become Urgent isn’t counted with VIP orders but in Urgent orders section only. Abeer Hussein
* In output file, No. Injured presents the total number of injuries not the total number of injured cooks.

Abeer Hussein