Pr



Cairo University Computer Engineering Department

Faculty of Engineering First Year



**Data structures & Algorithms Final assessment report**

**Team 7**

|  |  |  |
| --- | --- | --- |
| Name | Section | B.N. |
| Ghieath Omar Saleh Ajam | 2 | 5 |
| Mostafa Ahmed Sobhy Ahmed | 2 | 22 |
| Mostafa Mohamed Ahmed El-Gendy | 2 | 25 |
| Youssef Atef Tawfik Hafez | 2 | 39 |

**Spring 2020**

**Name: Ghieath Omar Saleh Ajam**

**ID: 205**

**Function Name:** RemoveFromBreakList

**Member of:** class Restaurant

**Inputs:** int currentTimeStep

**Returns:** void

**Called by:**

**Calls:**

* PriorityQueue<T>::peekFront(T& item)
* PriorityQueue<T>::Dequeue(T& item)
* Cook::EndBreak(Restaurant\* pRest)

**Function logic description:**

It checks all the cooks’ break lists (Priority Queue). If there is a cook who finished his break, then it deletes it from the list and instantiates and uses the end break function to reset his parameters.

**Function Name:** Run

**Member of:** class Restaurant

**Inputs:** PROG\_MODE mode

**Returns:** void

**Called by:** Restaurant::RunSimulation()

**Calls:**

- isEmpty() (for orders and events queues)

- Restaurant::ExecuteEvents(int ts)

- Restaurant::RemoveFromBreakList(int ts)

- Restaurant::RemoveFromRestList(int ts)

- Restaurant::CompleteOrders(int ts)

- Restaurant::ManageOrders(int ts)

- Restaurant::AssignOrders(int ts)

- Restaurant::CheckInjuries(int ts)

- Restaurant::FillDrawingList(int ts)

- GUI::waitForClick()

- std::Sleep(int ms)

- GUI::PrintMessage(str msg)

**Function logic description:**

It handles the flow of the program by executing events, handling breaks, rests and injuries of cooks, handles orders then passes instructions to GUI depending on program mode.

**Function Name:** Serve

**Member of:** class OrderService

**Inputs:** Restaurant\* pRest

**Returns:** void

**Called by:** Restaurant::AssignOrders(int currentTimeStep)

**Calls:**

* Order::SetStatus(ORD\_TYPE s)
* Order::SetST(int time)
* Order::SetFT(int time)
* Restaurant::AddToInserviceList(OrderService\* pServe)

**Function logic description:**

It sets the service time and the finish time of the assigned order and changes its status to SRV. Then, it adds a pointer to OrderService to the InserviceList.

**Function Name:** VipOrder

**Member of:** class VipOrder

**Inputs:** NormalOrder\* ord, int time

**Returns:** void

**Called by:** Restaurant::PromoteOrder(int ID, int time)

**Calls:**

* VipOrder::CalculatePriorityFactor()

**Function logic description:**

It is a constructor which converts a normal order to a vip order which is need in promotion event and auto promotion of normal orders to vip orders and calculates its priority factor. It takes time as a parameter to set the promotion time of the order in order to calculate the time at which the order becomes urgent.

**Function Name:** RunSimulation

**Member of:** class Restaurant

**Inputs:** void

**Returns:** void

**Called by:** Main function

**Calls:**

**-** GUI::getGUIMode()

- Restaurant::ReadFile()

- Restaurant::Run(PROG\_MODE mode)

- Restaurant::PrintFile()

**Function logic description:**

Organizes overall flow of the program, i.e. Creates a GUI, gets mode and passes it, handles input, starts simulation and prints final output.

**Function Name:** GetNormalOrderFromID

**Member of:** Restaurant

**Inputs:** int ID

**Returns:** NormalOrder\* no

**Called by:**

**-** Restaurant::CancelOrder

- Restaurant::PromoteOrder

- PromotionEvent::Execute

**Calls:**

- LinkedList<NormalOrder\*, int>::GetEntry

**Function logic description:**

It searches for a normal order by its ID then returns a pointer to it if found.

**Function Name:** operator >

**Member of:** class VipOrder

**Inputs:** VipOrder& ord

**Returns:** bool

**Called by:** LinkedSortedList<T, K>::Insert(const T& item)

**Calls:** void

**Function logic description:**

It is an overloaded greater than operator to compare two vip orders by using their priority factors to insert them sorted where the order of the highest priority is inserted first.

**Name: Mostafa Ahmed Sobhy Ahmed**

**ID: 222**

**Function Name:** FinishOrder

**Member of:** class OrderService

**Inputs:** Restaurant\* pRest, int time

**Returns:** void

**Called by:** Restaurant::CompleteOrders

**Calls:**

* Cook::IncrementServicedOrders()
* Order::SetStatus(ORD\_TYPE s)
* VipOrder::IsUrgent()
* Cook::GetServicedOrders()
* Cook::GetMaxNumberOrders()
* Cook::ResetServicedOrders()
* Cook::SetStartBreakTime(int time)
* Restaurant::AddToBreakList(Cook\* pCook)
* Restaurant::AddToRestList(Cook\* pCook)
* Restaurant::AddToCookList(Cook\* pCook)
* Restaurant::AddToFinishedList(Order\* pOrd)

**Function logic description:**

It increments the number of the serviced orders of the cook and sets the status of the order to DONE then adds it to the finished orders’ list. It checks if the order is a vip urgent order then checks if the cook was in break to reset his number of serviced orders then it adds the cook to the cook list and if he was injured and serving an urgent vip order then he is added to the cook list. If the order was of any type rather than the urgent vip type, then it checks if the cook was injured, if he was injured it checks if his number of served orders equals the maximum number of orders (the cook must take a break also) and resets it so that he takes only the rest. Then, it checks if the cook must take a break then it adds him to the break list and if not then he is added to the available cook list.

**Function Name:** CancelOrder

**Member of:** Restaurant

**Inputs:** int ID

**Returns:** void

**Called by:**

- CancellationEvent::Execute

**Calls:**

- LinkedList<NormalOrder \*, int>::Delete(T& item)

**Function logic description:**

It cancels an order of given ID by deleting it from the normal orders’ list.

**Function Name:** PromoteOrder

**Member of:** Restaurant

**Inputs:** int ID, int time

**Returns:** void

**Called by:**

- Restaurant::ManageOrders

- PromotionEvent::Execute

**Calls:**

- Restaurant::GetNormalOrderFromID

- LinkedList<NormalOrder \*, int>::Delete (T& item)

- LinkedSortedList<VipOrder \*, int>::Insert (T& item)

**Function logic description:**

Gets a pointer to the normal order of given ID then transforms it into a VIP order if found. It also sets the promotion time of order to be able to calculate the time at which the order will become urgent.

**Function Name:** AddToNormalList

**Member of:** Restaurant

**Inputs:** NormalOrder\* po

**Returns:** void

**Called by:**

- ArrivalEvent::Execute

**Calls:**

- LinkedList<NormalOrder \*, int>::InsertEnd

**Function logic description:**

Takes a pointer to a normal order and adds it to the end of the list of normal orders then increments number of normal orders by 1.

**Same for:** AddToVeganList, AddToVipList

**Function Name:** FindCook

**Member of:** Restaurant

**Inputs:** Order\*

**Returns:** Cook\*

**Called by:**

- Restaurant::FindCookForUrgentOrder

- Restaurant::AssignOrders

**Calls:**

- Queue<Cook \*>::dequeue(T& item)

- also uses Dynamic cast to check the type of the order

**Function logic description:**

Searches through available cooks then returns a valid one depending on the order type.

**Function Name:** AssignOrders

**Member of:** Restaurant

**Inputs:** int current TimeStep

**Returns:** void

**Called by:**

- Restaurant::Run

**Calls:**

- Order::GetAT

- Restaurant::FindCook

- OrderService::Serve

**Function logic description:**

Loops through lists of orders then searches for cooks to assign with found orders.

**Function Name:** operator int

**Member of:** class OrderService

**Inputs:** void

**Returns:** int

**Called by:** LinkedList<T,K>::GetEntry(T& item, K key)

**Calls:** void

**Function logic description:**

It converts the order service to int to be able to search for it in the list when it is done. The int is the FT of the order.

**Function Name:** AddToInserviceList

**Member of:** class Restaurant

**Inputs:** OrderService\* pServe

**Returns:** void

**Called by:**

* void OrderService::Serve(Restaurant\* pRest)

**Calls:**

* void BinarySearchTree<T, K>::Insert(T& item)

**Function logic description:**

This function will add the order to in-service list.

**Name: Mostafa Mohamed Ahmed El-Gendy**

**ID: 225**

**Function Name:** PrintFile

**Member of:** class Restaurant

**Inputs:** void

**Returns:** void

**Called by:** Restaurant::RunSimulation()

**Calls:**

* GUI::PrintMessage(string msg)
* GUI::GetString()
* LinkedList<T, K>::isEmpty()
* Order::GetID()
* Order::GetFT()
* Order::GetST()
* Order::GetAT()

**Function logic description:**

It takes the name of the output file from the user. Then, it calculates the total service time and the total wait time of the finished orders to calculate the average service time and the average wait time. Then, it prints the output file in the specified format.

**Function Name:** operator int

**Member of:** class VipOrder

**Inputs:** void

**Returns:** int

**Called by:** LinkedSortedList<T,K>::GetEntry(T& item, K key)

**Calls:** void

**Function logic description:**

It converts the vip order to int to be able to search for it in the list. The returned integer is (Vip\_WT + PromotionTime). It is used to get the urgent vip orders. If the order was originally a vip order (not promoted normal order) then the promotion time will be the arrival time.

**Function Name:** EndBreak

**Member of:** class Cook

**Inputs:** Restaurant\* pRest

**Returns:** void

**Called by:** Restaurant::RemoveFromBreakList(int currentTimeStep)

**Calls:**

* Restaurant::AddToCookList(Cook\* pCook)

**Function logic description:**

It resets the serviced orders of the cook and sets the value of the start break to -1 (default value). Then, it adds the cook to cook’s list.

**Function Name:** CompleteOrders

**Member of:** Restaurant

**Inputs:** int currentTimeStep

**Returns:** void

**Called by:**

- Restaurant::Run(PROG\_MODE mode)

**Calls:**

- LinkedList<T, K>::Delete(T& item)

- OrderService::FinishOrder(Restaurant\* pRest, int time)

**Function logic description:**

It checks for the finished orders at the current time step then deletes it from the in – service list. Then, it calls FinishOrder function to move any finished order to finished orders list. It checks if the order was a vip urgent order and the cook was in break or rest. If the cook was in break then it resets his break and adds him to the cook’s list, if he was in rest then his speed is not doubled and he is also added to the cook’s list. This function also adds the cook’s to their appropriate list (rest or break or free).

**Function Name:** ManageOrders

**Member of:** Restaurant

**Inputs:** int currentTimeStep

**Returns:** void

**Called by:**

- Restaurant::Run(PROG\_MODE mode)

**Calls:**

- Order::GetAT()

- NormalOrder::GetAutoPromote()

- Restaurant::PromoteOrder(int ID, int time)

- Order::GetID()

- LinkedSortedList<T,K>::GetEntry(T& item, K key)

- Restaurant::FindCookForUrgentOrder(VipOrder\* pOrd)

- VipOrder::Urgent()

- OrderService::Serve(Restaurant\* pRest)

- LinkedSortedList<T, K>::Delete(T& item)

- Queue<T>::peekFront(T& item)

- Queue<T>::dequeue(T& item)

**Function logic description:**

This function handles special events including promoting normal orders and it checks all the orders to be urgent and enqueues them in the urgent orders list. Then, it assigns cooks to urgent VIP orders.

**Function Name:** AddToRestList

**Member of:** class Restaurant

**Inputs:** Cook\* pCook

**Returns:** void

**Called by:**

* void OrderService::FinishOrder(Restaurant\* pRest, int time)

**Calls:**

* Cook::GetType()
* Queue<T>::enqueue(const T& newEntry)

**Function logic description:**

This function checks the cook type and put it in its suitable list in restaurant so if the cook type is VIP, the function will enqueue it in the VIP queue and so on to the rest of cook types.

**Function Name:** InjureCook

**Member of:** class OrderService

**Inputs:** int currentTimeStep

**Returns:** void

**Called by:** Restaurant::CheckInjuries()

**Calls:**

* Cook::SetInjury(bool flag)
* Cook::GetSpeed()
* Cook::HalfSpeed()
* Order::GetSize()
* Order::SetST(int time)
* Order::SetFT(int time)

**Function logic description:**

It calculates the remained dishes of the order before the injury of the cook, then it estimates the new service time and finish time then it sets them to the order and it decreases the speed of the cook to its half.

**Function Name:** EndRest

**Member of:** class Cook

**Inputs:** Restaurant\* pRest

**Returns:** void

**Called by:** Restaurant::RemoveFromRestList(int currentTimeStep)

**Calls:**

* Restaurant::AddToCookList(Restaurant\* pRest)

**Function logic description:**

It sets the injury to false, doubles the cook’s speed, and sets the start break time to -1. Then, it adds the cook to the cook’s list.

**Name: Youssef Atef Tawfik Hafez**

**ID: 239**

**Function Name:** CheckInjuries

**Member of:** class Restaurant

**Inputs:** int currentTimeStep

**Returns:** void

**Called by:**

* Restaurant::Run(PROG\_MODE mode)

**Calls:**

* LinkedList<T, K>::GetEntry(T& item)
* OrderService::InjureCook(int currentTimeStep)

**Function logic description:**

This function generates a random float number (between 0 and 1) then it chooses the first busy cook and injures him.

**Function Name:** AddToCookList

**Member of:** class Restaurant

**Inputs:** Cook\* pCook

**Returns:** void

**Called by:**

* Cook::EndBreak(Restaurant\* pRest)
* Cook::EndRest(Restaurant\* pRest)
* OrderService::FinishOrder(Restaurant\* pRest, int time)

**Calls:**

* ORD\_TYPE Cook::GetType()
* Queue<T>::enqueue(const T& newEntry)

**Function logic description:**

This function sets the input cook in its suitable cook list so it first checks its type then add it to the list.

**Function Name:** AddToFinishedList

**Member of:** class Restaurant

**Inputs:** Order\* ord

**Returns:** void

**Called by:**

* void OrderService::FinishOrder(Restaurant\* pRest, int time)

**Calls:**

* LinkedList<T, K>::InsertEnd(const T& data)

**Function logic description:**

This function inserts the finished orders in the finished orders list.

**Function Name:** AddToBreakList

**Member of:** class Restaurant

**Inputs:** Cook\* pCook

**Returns:** void

**Called by:**

* OrderService::FinishOrder(Restaurant\* pRest, int time)

**Calls:**

* ORD\_TYPE Cook::GetType()
* PriorityQueue<T>::enqueue(const T& newEntry)

**Function logic description:**

This function takes a cook that must have a break and add it to break queue.

**Function Name:** FillDrawingList

**Member of:** Restaurant

**Inputs:** int currentTimeStep

**Returns:** void

**Called by:**

- Restaurant::Run

**Calls:**

- GUI::ResetDrawingList

- Order::GetType

- OrderService::GetOrder

- OrderService::GetCook

- GUI::AddToDrawingList

- OrderService::GetStartTime

- Cook::GetType

- Cook::GetID

- Order::GetID

- GUI::PrintMessage

- GUI::UpdateInterface

**Function logic description:**

Resets drawing list then fills it again from scratch by looping through every list of orders/cooks while counting numbers of each type of orders and their service status.

**Function Name:** operator >

**Member of:** class Cook

**Inputs:** Cook& cook

**Returns:** bool

**Called by:** PriorityQueue<T >::enqueue(T& item)

**Calls:** void

**Function logic description:**

It compares between two cooks in terms of the break duration so that the cook with the lowest break duration has higher priority ti be enqueued in the break list.

**Function Name:** FindCookForUrgentOrder

**Member of:** Restaurant

**Inputs:** VipOrder\*

**Returns:** Cook\*

**Called by:**

- Restaurant::ManageOrders

**Calls:**

- Restaurant::FindCook

- Queue<T>::dequeue(T& item)

**Function logic description:**

First searches through available cooks by calling FindCook function. If no cook available it searches in cooks break lists then cooks rest lists.