Cairo University, Faculty of Engineering

Computer Engineering Department Spring 2020

Data Structures and Algorithms

CMPN102

**Data Structures and Algorithms**

**Final Assessment Report**

**Team Name: 4-7\_T6** | **Number of Members: 3**

Email: [mostafa.hassanien659@gmail.com](mailto:mostafa.hassanien659@gmail.com)

[Hosammotaz700@gmail.com](mailto:Hosammotaz700@gmail.com)

[Abdelrahman.mohamed1772000@gmail.com](mailto:Abdelrahman.mohamed1772000@gmail.com)

**Section1: Mostafa Hassanien Ahmed, 1180581**

1. **Assign\_Orders**

* **Member of:** Class Restaurant
* **Inputs:**
* int CurrentTimeStep: the current time step during the simulation
* **Returns:**
* No Returns (void)
* **Called By:**

- Restaurant::RunSimulation()

* **Calls:**

- Restaurant::Get\_AvailableCook()

- Order:: setStatus()

- Cook:: Assign\_Order()

* **Function Logic description:**

It checks the availability of different cooks’ types, hence, it assigns the different orders’ types according to the agreed assigning criteria in the project document. Moreover, it changes the status of the order to be served and assign it to the chosen available cook.

1. **Get\_AvailableCook**

* **Member of:** Class Restaurant
* **Inputs:**
* List<Cook\*> & Co: cooks’ type list to be searched in for an available cook (Normal\_Cooks, VIP\_Cooks, Vegan\_Cooks)
* **Returns:**
* Returns the index of the available cook in the passed cooks list
* **Called By:**

- Restaurant::Assign\_Orders()

- Restaurant::Urgent\_Orders()

* **Calls:**

- Cook:: get\_Assigned()

* **Function Logic description:**

It check the assigning status of every cook in the passed list and returns the index of the first available cook in the list.

1. **Assign\_Order**

* **Member of:** Class Cook
* **Inputs:**
* Order\* pO: an order pointer that points to the assigned order
* int CurrentStep: the current time step during the simulation
* **Returns:**
* No Returns (void)
* **Called By:**

- Restaurant::Assign\_Orders()

- Restaurant::Urgent\_Orders()

* **Calls:**

- Order:: Set\_ServeTime()

- Order::get\_size()

* **Function Logic description:**

It assigns the passed order pointer to the order pointer in the class cook to know which cook is assigned to which order. Moreover, it sets the required order size to the assigned cook and change cook’s status to be assigned.

1. **Advance\_Order**

* **Member of:** Class Restaurant
* **Inputs:**
* int CurrentTimeStep: the current time step during the simulation
* **Returns:**
* No Returns (void)
* **Called By:**

- Restaurant::RunSimulation()

* **Calls:**

- Cook:: get\_Assigned()

- Cook:: Advance\_AssignedOrd()

* **Function Logic description:**

It advances the assigned order. In other words, every time step it searches for the assigned cooks to advance their assigned orders; it calls Advance\_AssignedOrd() function that checks if the order size is done or not. If the order is done, it is transferred from in-service orders list to finished orders list. On the other hand, if it is not, then the defined speed – whether it is his/her full speed or half one according to the cook status - of assigned cook is subtracted from the required order size.

1. **Advance\_AssignedOrd**

* **Member of:** Class Cook
* **Inputs:**
* int CurrentTimeStep: the current time step during the simulation
* **Returns:**
* Returns an order pointer that points to the assigned finished order. If the return is null, then the order has not finished yet.
* **Called By:**

- Restaurant::Advance\_Order()

* **Calls:**

- Order:: setStatus()

- Order:: Set\_FinishedTime()

* **Function Logic description:**

It advances the assigned order. In other words, every time step it checks the assigned cook’s status if it is assigned, assigned urgently, or injured. Accordingly, it checks if the order is finished or not. If it hasn’t finished yet it decreases the required order size by the speed (if the cook is assigned) or by half the speed (if the cook is assigned urgently or injured). On the other hand, if the order is finished it change its status to done (finished) and change the cooks status, for sure. In addition, it return a pointer for the finished assigned order.

1. **PrintMessage**

* **Member of:** Class GUI
* **Inputs:**
* string msg: the desired output message
* bool Clear\_Status: if it is true, I want to clear the status bar.
* double Num: define the position of the line in the status bar to print

multiple lines

* **Returns:**
* No Returns (void)
* **Called By:**

- Restaurant::RunSimulation()

- Restaurant::Output\_File()

* **Calls:**

- GUI:: ClearStatusBar()

- window:: SetPen()

- window:: DrawString()

- window:: SetFont()

- window:: DrawString()

* **Function Logic description:**

It was modified with two features, the boolean variable and double number, to have a control on the clearance of the status bar and to be able to print multiple lines.

1. **Output\_File**

* **Member of:** Class Restaurant
* **Inputs:**
* No Inputs
* **Returns:**
* No Returns (void)
* **Called By:**

- Restaurant::RunSimulation()

* **Calls:**

- GUI:: PrintMessage()

- Order:: get\_FinishTime()

- Order:: GetID()

- Order:: get\_ArrTime()

- Order:: getServeTime()

- Restaurant::Sorted\_FinishedOrders()

* **Function Logic description:**

This function is responsible for producing the output file with the required format as stated in the project document.

1. **Sorted\_FinishedOrders**

* **Member of:** Class Restaurant
* **Inputs:**
* Queue<Order\*> pO: Queue of the finished orders
* int& count: number of total finished orders
* **Returns:**
* Returns array of pointers that points to the finished orders, but sorted with a given criteria.
* **Called By:**

- Restaurant::Output\_File()

* **Calls:**

- Order:: get\_FinishTime()

- Order:: getServeTime()

* **Function Logic description:**

This function is responsible for sorting finished orders by finishing time in ascending order. If more than one order is finished at the same time step, then they are ordered by serving time.

1. **Break\_Time**

* **Member of:** Class Restaurant
* **Inputs:**
* No Inputs
* **Returns:**
* No Returns (void)
* **Called By:**

- Restaurant::RunSimulation()

* **Calls:**

- Cook:: Get\_Num\_OrdDone()

- Cook:: Get\_NumOrder()

- Cook: Set\_Assigned\_Status()

* **Function Logic description:**

It check if the assigned cook has already done all the required number of orders before taking the break or not. If he/she has already done that, then it changes its status to “In\_Break”.

1. **Return\_Break**

* **Member of:** Class Restaurant
* **Inputs:**
* No Inputs
* **Returns:**
* No Returns (void)
* **Called By:**

- Restaurant::RunSimulation()

* **Calls:**

- Cook:: Get\_Break\_StartTime()

- Cook:: GetBreak()

- Cook: get\_Assigned()

- Cook:: Set\_Assigned\_Status()

- Cook:: Advance\_BreakDuration()

* **Function Logic description:**

It check in the In\_Break cooks if they have already finished their break or not. If a cook has already finished the break duration, then its status is changed to not assigned and initializes the number of order done and break time to zero. If not, then the break duration is increased by one time step for this cook.

1. **Advance\_BreakDuration**

* **Member of:** Class Cook
* **Inputs:**
* No Inputs
* **Returns:**
* No Returns (void)
* **Called By:**

- Restaurant::Return\_Break()

* **Calls:**
* No Calls
* **Function Logic description:**

It simply advances the break time by one time step.

1. **Execute**

* **Member of:** Class ArrivalEvent
* **Inputs:**
* Restaurant\* pRest: a pointer to the restaurant
* **Returns:**
* No Returns (void)
* **Called By:**

- Restaurant::ExecuteEvents()

* **Calls:**

- Restaurant:: AddtoNORMALQueue()

- Restaurant:: AddtoVEGANQueue()

- Restaurant:: AddtoVIPQueue()

* **Function Logic description:**

It check the type of the arrived order and calls the corresponding function to add the order to its list.

1. **AddtoNORMALQueue**

* **Member of:** Class Restaurant
* **Inputs:**
* Order\* pOrd: a pointer to the arrived (new) order
* **Returns:**
* No Returns (void)
* **Called By:**

- ArrivalEvent::Execute()

* **Calls:**
* No Calls
* **Function Logic description:**

It inserts the new normal order to the normal orders list in the restaurant.

1. **AddtoVEGANQueue**

* **Member of:** Class Restaurant
* **Inputs:**
* Order\* pOrd: a pointer to the arrived (new) order
* **Returns:**
* No Returns (void)
* **Called By:**

- ArrivalEvent::Execute()

* **Calls:**
* No Calls
* **Function Logic description:**

It enqueues the new vegan order to the vegan orders queue in the restaurant.

1. **AddtoVIPQueue**

* **Member of:** Class Restaurant
* **Inputs:**
* Order\* pOrd: a pointer to the arrived (new) order
* **Returns:**
* No Returns (void)
* **Called By:**

- ArrivalEvent::Execute()

* **Calls:**
* Restaurant::Calc\_Priority()
* **Function Logic description:**

It insert the new VIP order to the VIP orders list in the restaurant after calculating the priority of the new order.

1. **Calc\_Priority**

* **Member of:** Class Restaurant
* **Inputs:**
* Order\* pOrd: a pointer to the arrived (new) order
* **Returns:**
* Returns a double number to represent the priority of the order
* **Called By:**

- Restaurant::AddtoVIPQueue()

* **Calls:**
* Order:: get\_size()
* Order:: get\_ArrTime()
* Order:: get\_Money()
* **Function Logic description:**

It calculates the priority of the passed order using the following equation:

Priority = (0.3\*size) + (0.3\*arriveTime) + (0.4\*money)

**Note:**

The input files were generated by Mostafa Hassanien Ahmed according to the new input format as stated inside the final project document. Hence; after running each test case of the input files, the program generates their corresponding output files. The test cases are ranged from simple to complex one, in addition, they cover cancel events, auto-promotion orders, and urgent orders. Besides, the average waitings are ranged from very low to very high average waiting. All test cases cover the mentioned properties, except test 1 and 6, they don’t cover urgent orders due to their low average wait.

**Section2: Hosam Motaz Hosam, 1180534**

1. **Auto\_PromoteNormal**

* **Member of:** Class Restaurant
* **Inputs:**
* int CurrentTimeStep: the current time step during the simulation
* **Returns:**
* No Returns (void)
* **Called By:**

- Restaurant::RunSimulation()

* **Calls:**

- Order:: get\_ArrTime()

* **Function Logic description:**

It check in the Normal order list for the orders that have exceeded the waiting time that is defined in the input file (AutoP). Then, if it is found such an order, it simply generates a promotion event and sends all required data, besides, executing this event.

1. **Print\_Info\_1**

* **Member of:** Class Restaurant
* **Inputs:**
* int CurrentTimeStep: the current time step during the simulation
* **Returns:**
* Returns the first line string message of the required information, printed at the bottom of the screen.
* **Called By:**

- Restaurant::RunSimulation()

* **Calls:**
* No Calls
* **Function Logic description:**

It generates a string message that accounts for the first line of the required information, printed at the bottom of the screen.

1. **Print\_Info\_2**

* **Member of:** Class Restaurant
* **Inputs:**
* No Inputs
* **Returns:**
* Returns the second line string message of the required information, printed at the bottom of the screen.
* **Called By:**

- Restaurant::RunSimulation()

* **Calls:**
* No Calls
* **Function Logic description:**

It generates a string message that accounts for the second line of the required information, printed at the bottom of the screen.

1. **Print\_Info\_3**

* **Member of:** Class Restaurant
* **Inputs:**
* No Inputs
* **Returns:**
* Returns the third line string message of the required information, printed at the bottom of the screen.
* **Called By:**

- Restaurant::RunSimulation()

* **Calls:**
* No Calls
* **Function Logic description:**

It generates a string message that accounts for the third line of the required information, printed at the bottom of the screen.

1. **Print\_Info\_4**

* **Member of:** Class Restaurant
* **Inputs:**
* No Inputs
* **Returns:**
* Returns the fourth line message of the required information, printed at the bottom of the screen.
* **Called By:**

- Restaurant::RunSimulation()

* **Calls:**
* Cook:: get\_AssignedOrd\_Info()
* **Function Logic description:**

It generates a string message that accounts for the fourth line of the required information, printed at the bottom of the screen.

1. **get\_AssignedOrd\_Info**

* **Member of:** Class Restaurant
* **Inputs:**
* No Inputs
* **Returns:**
* Returns a string message that carry the needed data, in the required format, in the fourth line at the bottom of the screen at each time step.
* **Called By:**

- Restaurant::Print\_Info\_4()

* **Calls:**
* Order:: get\_size()
* Order:: GetID()
* **Function Logic description:**

It is responsible for generating the message that carries the required data in the fourth line at the bottom of the screen. It sends this data if the order has already assigned to the cook at this time step, otherwise, it sends “No” as indication that no assigned order has done.

1. **Print\_Info\_5**

* **Member of:** Class Restaurant
* **Inputs:**
* No Inputs
* **Returns:**
* Returns the fifth line message of the required information, printed at the bottom of the screen.
* **Called By:**

- Restaurant::RunSimulation()

* **Calls:**
* No Calls
* **Function Logic description:**

It generates a string message that accounts for the fifth line of the required information, printed at the bottom of the screen.

1. **Urgent\_Orders**

* **Member of:** Class Restaurant
* **Inputs:**
* int CurrentTimeStep: the current time step during the simulation
* **Returns:**
* No Returns (void)
* **Called By:**

- Restaurant::RunSimulation()

* **Calls:**

- Restaurant::Get\_AvailableCook()

- Order:: setStatus()

- Cook:: Assign\_Order()

- Restaurant:: Get\_InBreakCook()

- Restaurant::Insert\_UrgentOrders()

- Cook:: Assign\_UrgentOrder()

* **Function Logic description:**

It assigns the urgent orders from the urgent orders list using the criteria defined in the project document. In details, it assigns it as a usual order and if there is no available cook, the function calls Get\_InBreakCook function to take the first In\_Break cook to assign to him/her the order and reducing the cook speed by half its value. Besides, in case that there is no In\_Break cook, it searchs in the In\_Rest cooks list to get the first cook and assigns to him/her the order, in addition, reducing the cook speed by half its value.

1. **Assign\_UrgentOrder**

* **Member of:** Class Cook
* **Inputs:**
* Order\* pO: an order pointer that points to the urgent order
* int CurrentStep: the current time step during the simulation
* **Returns:**
* No Returns (void)
* **Called By:**

- Restaurant::Urgent\_Orders()

* **Calls:**

- Order:: Set\_ServeTime()

- Order:: get\_size()

* **Function Logic description:**

It sets the passed order pointer to the order pointer in the cook class and change the cook’s status to assigned urgently. In addition, it sets the serving time of the order and the required order size.

1. **Insert\_UrgentOrders**

* **Member of:** Class Restaurant
* **Inputs:**
* int CurrentStep: the current time step during the simulation
* **Returns:**
* No Returns (void)
* **Called By:**

- Restaurant::Urgent\_Orders()

* **Calls:**

- Order:: get\_ArrTime()

* **Function Logic description:**

It searches in VIP orders list for the orders that have exceeded the VIP\_WT in order to remove them from VIP orders list and insert them in Urgent orders list to be served first in the following time steps.

1. **Get\_InBreakCook**

* **Member of:** Class Restaurant
* **Inputs:**
* List<Cook\*> & Co: cooks’ type list to be searched in for an In-Break cook
* **Returns:**
* Returns the index of the In-Break cook in the passed cooks list
* **Called By:**

- Restaurant::Urgent\_Orders()

* **Calls:**

- Cook:: get\_Assigned()

* **Function Logic description:**

It check the assigning status of every cook in the passed list and returns the index of the first In-Break cook in the passed list.

1. **Execute**

* **Member of:** Class Promotion\_Event
* **Inputs:**
* Restaurant\* pRest: a pointer to the restaurant
* **Returns:**
* No Returns (void)
* **Called By:**

- Restaurant::ExecuteEvents()

- Restaurant::Auto\_PromoteNormal()

* **Calls:**

- Restaurant:: Promote\_Normal()

* **Function Logic description:**

It calls Promote\_Normal function of the restaurant class to change the normal order to VIP order.

1. **Promote\_Normal**

* **Member of:** Class Restaurant
* **Inputs:**
* int OrderID: order ID
* double Money: the extra paid money
* **Returns:**
* No Returns (void)
* **Called By:**

- Promotion\_Event::Execute()

* **Calls:**
* Order:: SetMoney()
* Order:: setType()
* **Function Logic description:**

It checks firstly on the ID, whether it is Normal order’s ID or not. If it is, the function changes the money of the order to the previous money plus the extra money, besides, changing its type from normal to VIP. Moreover, it picks the promoted order from normal orders list and insert it in VIP orders list.

**Section3: Abdelrahman Mohamed Mahdy, 1180521**

1. **RunSimulation**

* **Member of:** Class Restaurant
* **Inputs:**
* No Inputs
* **Returns:**
* No Returns (void)
* **Called By:**

- main( )

* **Calls:**

- GUI::PrintMessage() // modified PrintMessage() for phase 2.

- Restaurant::ExecuteEvents()

- Restaurant:: Load\_InfoFile()

- Restaurant::Urgent\_Orders()

- Restaurant::Auto\_PromoteNormal()

- Restaurant::Return\_Break()

- Restaurant::Return\_Rest()

- Restaurant::Assign\_Orders()

- Restaurant::FillDrawingList()

- Restaurant::Print\_Info\_1()

- Restaurant::Print\_Info\_2()

- Restaurant::Print\_Info\_3()

- Restaurant::Print\_Info\_4()

- Restaurant::Print\_Info\_5()

- Restaurant::Advance\_Order()

- Restaurant::Break\_Time()

- Restaurant::Injured\_BusyCook()

- Restaurant::All\_AvailCook()

- Restaurant::Output\_File()

* **Function Logic description:**

It is regarded as the controller of the project that takes the desired mode from the user and runs the simulation according it. It calls the function responsible for loading the input file and calls the major functions that regulate the simulation process. Then, produce the required output file by calling the function responsible for this task.

1. **Load\_InfoFile**

* **Member of:** Class Restaurant
* **Inputs:**
* ifstream& Infile: the input file
* **Returns:**
* No Returns (void)
* **Called By:**

- Restaurant::RunSimulation()

* **Calls:**

- Restaurant:: Read\_CookInfo()

- Restaurant:: Read\_Events()

* **Function Logic description:**

For the sake of the organization, this function calls Read\_Events and Read\_CookInfo functions and loads the other required data from the input file.

1. **Read\_Events()**

* **Member of:** Class Restaurant
* **Inputs:**
* ifstream& Infile: the input file
* **Returns:**
* No Returns (void)
* **Called By:**

- Restaurant::Load\_InfoFile()

* **Calls:**
* No Calls
* **Function Logic description:**

This function is responsible for loading all the required data related to the restaurant’s events. Then, it generate the event, whether it is arrival, cancellation, or promotion event, and load its data.

1. **Read\_CookInfo()**

* **Member of:** Class Restaurant
* **Inputs:**
* ifstream& Infile: the input file
* **Returns:**
* No Returns (void)
* **Called By:**

- Restaurant::Load\_InfoFile()

* **Calls:**

- Cook:: setID()

- Cook:: setType()

- Cook:: setSpeed()

- Cook:: set\_NumOrder()

- Cook:: setBreak()

- Cook:: set\_RstPrd()

* **Function Logic description:**

This function is responsible for loading all the required information related to the restaurant’s cooks. Then, it assigns this data to the generated cook, each type on his own.

1. **All\_AvailCook**

* **Member of:** Class Restaurant
* **Inputs:**
* No Inputs
* **Returns:**
* No Returns (void)
* **Called By:**

- Restaurant::RunSimulation()

* **Calls:**

- Cook:: Set\_Assigned\_Status()

* **Function Logic description:**

It makes all cooks available after the simulation has already finished, in other words, all orders have been served so that all cooks are drawn on the screen.

1. **FillDrawingList**

* **Member of:** Class Restaurant
* **Inputs:**
* No Inputs
* **Returns:**
* No Returns (void)
* **Called By:**

- Restaurant::RunSimulation()

* **Calls:**

- Cook:: get\_Assigned()

- Cook:: GetType()

- GUI:: AddToDrawingList()

- Restaurant:: Sort\_Cook()

* **Function Logic description:**

It fills the screen with the required drawn object, such as available cooks’ IDs, waiting orders’ IDs, serving orders’ IDs, and finished orders’ IDs. Each item is being drawn with the sorting technique defined in the project document.

1. **Sort\_Cook**

* **Member of:** Class Restaurant
* **Inputs:**
* int & Count: the total number of cooks in the restaurant
* **Returns:**
* Returns the total number of cooks, besides, an array of cook pointers that is sorted with the availability of the cook.
* **Called By:**

- Restaurant:: FillDrawingList()

* **Calls:**

- Cook:: Avail\_Time()

* **Function Logic description:**

It sorts all cooks according to the availability of each cook in order to be drawn on the screen as stated in the project document.

1. **Avail\_Time**

* **Member of:** Class Cook
* **Inputs:**
* No inputs
* **Returns:**
* Returns an integer that represents the availability of each cook
* **Called By:**

- Restaurant:: Sort\_Cook()

* **Calls:**

- No Calls

* **Function Logic description:**

It simply calculates the availability of each cook by subtracting the total number of orders required before taking a break from the number of orders has already done by the cook.

1. **Return\_Rest**

* **Member of:** Class Restaurant
* **Inputs:**
* No Inputs
* **Returns:**
* No Returns (void)
* **Called By:**

- Restaurant::RunSimulation()

* **Calls:**

- Cook:: get\_RstPrd()

- Cook:: Get\_RstPrd\_Duration()

- Cook:: Set\_Assigned\_Status()

- Cook:: Advance\_RstPrd\_Duration()

* **Function Logic description:**

It check in the In\_Rest cooks list if each cook has already finished his/her rest or not. If a cook has already finished the rest period duration, then its status is changed to not assigned and initializes the number of order done and rest period duration to zero. If not, then the rest period duration is increased by one time step for this cook.

1. **Advance\_RstPrd\_Duration**

* **Member of:** Class Cook
* **Inputs:**
* No Inputs
* **Returns:**
* No Returns (void)
* **Called By:**

- Restaurant:: Return\_Rest()

* **Calls:**
* No Calls
* **Function Logic description:**

It simply advances the rest period duration by one time step.

1. **Injured\_BusyCook**

* **Member of:** Class Restaurant
* **Inputs:**
* No Inputs
* **Returns:**
* No Returns (void)
* **Called By:**

- Restaurant::RunSimulation()

* **Calls:**

- Restaurant:: Get\_FirstBusyCook()

- Cook:: Set\_Assigned\_Status()

* **Function Logic description:**

At first, it generates a random number R between 0 and 1, then check if

R ≤ InjProp, if so, then it gets the first busy cook (assigned cook) and change its status to injured in order to reduce its speed to its half value.

1. **Get\_FirstBusyCook**

* **Member of:** Class Restaurant
* **Inputs:**
* List<Cook\*> & Co: cooks’ type list to be searched in for an assigned (busy) cook (Normal\_Cooks, VIP\_Cooks, Vegan\_Cooks)
* **Returns:**
* Returns the index of the assigned cook in the passed cooks list
* **Called By:**

- Restaurant::Injured\_BusyCook()

* **Calls:**

- Cook:: get\_Assigned()

* **Function Logic description:**

It check the assigning status of every cook in the passed list and returns the index of the first busy (assigned) cook in the list.

1. **Execute**

* **Member of:** Class Cancellation\_Event
* **Inputs:**
* Restaurant\* pRest: a pointer to the restaurant
* **Returns:**
* No Returns (void)
* **Called By:**

- Restaurant::ExecuteEvents()

* **Calls:**

- Restaurant:: Cancel\_Normal()

* **Function Logic description:**

It calls Cancel\_Normal function of the restaurant class to delete the normal order from normal orders list.

1. **Cancel\_Normal**

* **Member of:** Class Restaurant
* **Inputs:**
* int OrderID: order ID
* **Returns:**
* No Returns (void)
* **Called By:**

- Cancellation\_Event::Execute()

* **Calls:**
* No Calls
* **Function Logic description:**

It checks firstly on the ID, whether it is Normal order’s ID or not. If it is, then it deletes the normal order from the normal orders list.