**Report**

|  |  |  |  |
| --- | --- | --- | --- |
| |  |  | | --- | --- | | Cairo University, Faculty of Engineering  Computer Engineering Department  Data Structures and Algorithms  CMP102/CMPN102 | Spring 2020 | | |
| **Data Structures and Algorithms**  **Final Assessment Report** | |
| **Team Name: Team 8** | **Number of members: 4** |
| **Email:** [**reem.atalah1@gmail.com**](mailto:reem.atalah1@gmail.com)  [**basantabdo234@gmail.com**](mailto:basantabdo234@gmail.com)  **esraaahmed1762000@gmail.com**  [**15126@stemegypt.edu.eg**](mailto:15126@stemegypt.edu.eg) |  |
| **Section1: Reem Emad, 135**   1. **AutopromotionOfOrders**   **Member** **of**: Class Restaurant  **Inputs**:  No inputs  **Returns**:  Void function  **Called** **By**:   * Restaurant::RunSimulation()   **Calls**:   * Restaurant::AutopromotionOfOrders()   **Function** **Logic** **description**:  This function traverses in the normal orders list to check if this order waited a time which equals auto promotion limit time, if so then it put the order in the VIP priority queue waiting list.   1. **OutputFileOptimization**   **Member** **of**: Class Restaurant  **Inputs**:  outputFile: output file of type ofstream  **Returns**:  Void function  **Called** **By**:   * Restaurant::RunSimulation()   **Calls**:   * Restaurant::AutopromotionOfOrders(ofstream& outputFile)   **Function** **Logic** **description**:  This function writes in the given output file some statistics about orders. First it shows the data of all orders according to their finish time, ID, arrival time, waiting time and serving time. Finished order are put in descending order according to FT and in ascending order according to ST. It also shows the total number of orders and total number of each order type in addition to total number of cooks and total number of each cook type, average waiting time, average serving time, number of urgent orders and percentage of auto promoted orders with respect to total number orders.   1. **MovingOrdersFromServingToDone**   **Member** **of**: Class Restaurant  **Inputs**:  No inputs  **Returns**:  Void function  **Called** **By**:   * Restaurant::RunSimulation()   **Calls**:   * Restaurant:: MovingOrdersFromServingToDone()   **Function** **Logic** **description**:  This function moves orders from serving list to finished queue -in descending order according to FT and in ascending order according to ST- if all its dishes are done, increases the number of done dishes and number of done orders to the cook and make him available again.   1. **operator <**   **Member** **of**: Class Cook  **Inputs**:  pCook: Cook object  **Returns**:  Boolean  **Called** **By**:   * PriorityQueue<Cook\*>:: sort(char c)   **Calls**:   * Cook:: operator <(Cook pCook)   **Function** **Logic** **description**:  This function overloads the less than sign operator to compare between cooks’ speed and returns true or false.   1. **operator <**   **Member** **of**: Class Order  **Inputs**:  pOrder: Order object  **Returns**:  Boolean  **Called** **By**:   * PriorityQueue<<Order\*>`:: sort(char c)   **Calls**:   * Order:: operator <(Order pOrder)   **Function** **Logic** **description**:  This function overloads the less than sign operator to compare between VIP orders’ priority equation and returns true or false.   1. **CalcPriortyVIPEqn**   **Member** **of**: Class Order  **Inputs**:  No inputs  **Returns**:  Integer  **Called** **By**:   * Order:: operator <(Order pOrder)   **Calls**:   * Order:: CalcPriorityVIPEqn()   **Function** **Logic** **description**:  This function calculates the priority equation result for VIP. | |
| **Section1: Bassant AbdelAzim, 120**   1. **GetProOrder**   **Member** **of**: Class Restaurant  **Inputs**:  id: integer number represents the order id  **Returns**:  Void function  **Called** **By**:   * PromotionEvent::Execute(Restaurant\* pRest)   **Calls**:   * pRest->GetProOrder(OrderID);   **Function** **Logic** **description**:  This function removes a normal order in normal orders list into VIP waiting priority queue if it was not served or finished.   1. **ReduceMedicalRest**   **Member** **of**: Class Restaurant  **Inputs**:  No inputs  **Returns**:  Void function  **Called** **By**:   * Restaurant::RunSimulation()   **Calls**:   * Restaurant::ReduceMedicalRest();   **Function** **Logic** **description**:  This function decreases medical period for the injured cook who does not have any order to do till it reaches zero then he is not injured anymore and be available once again with his normal speed.   1. **AssigningOrdersToCooks**   **Member** **of**: Class Restaurant  **Inputs**:  No inputs  **Returns**:  Void function  **Called** **By**:   * Restaurant::RunSimulation()   **Calls**:   * Restaurant:: AssigningOrdersToCooks()   **Function** **Logic** **description**:  This function assigns order to the available cook according to criteria of assigning orders and change their status from waiting to serving and make the chosen cook unavailable till he finishes the order.   1. **DrawAssignedCooksInLastTimestep**   **Member** **of**: Class Restaurant  **Inputs**:  No inputs  **Returns**:  Void function  **Called** **By**:   * Restaurant::RunSimulation()   **Calls**:   * Restaurant:: AssigningOrdersToCooks()   **Function** **Logic** **description**:  This function prints in the status bar all orders which was assigned in the last time step to whom the cook type should be written first then his id then in brackets the order type and it's id.   1. **PrintMessage**   **Member** **of**: Class GUI  **Inputs**:  msg: string object which has the data wanted to be printed.  nVLine: integer represents the number of the line.  writeHorizontal: character indicates to write in right or left corner.  **Returns**:  Void function  **Called** **By**:   * Restaurant:: PrintingTimestep() * Restaurant:: DrawingAllAssignedCooksInTheLastTimeTimestep() * Restaurant:: print\_number\_of\_avalible\_cook() * Restaurant:: Print\_Injured\_Cooks() * Restaurant:: Print\_Cooks\_In\_Break()   **Calls**:   * GUI:: AssigningOrdersToCooks()   **Function** **Logic** **description**:  This function prints in the status bar any data.   1. **Print\_Injured\_Cooks**   **Member** **of**: Class Restaurant  **Inputs**:  No inputs  **Returns**:  Void function  **Called** **By**:   * Restaurant::RunSimulation()   **Calls**:   * Restaurant:: Print\_Injured\_Cooks ()   **Function** **Logic** **description**:  This function prints in the status bar ID of injured cooks. | |
| **Section1: Esraa Gamal, 115**   1. **Reduce\_avaliblity\_cook**   **Member** **of**: Class Restaurant  **Inputs**:  No inputs  **Returns**:  Void function  **Called** **By**:   * Restaurant::RunSimulation()   **Calls**:   * Restaurant::ReduceMedicalRest();   **Function** **Logic** **description**:  This function checks break time for cooks in break who does not have any order to do if they finished their break time to make them available again.   1. **serve\_waiting\_vip**   **Member** **of**: Class Restaurant  **Inputs**:  No inputs  **Returns**:  Void function  **Called** **By**:   * Restaurant::RunSimulation()   **Calls**:   * Restaurant::serve\_waiting\_vip ();   **Function** **Logic** **description**:  This function serves the waiting VIP orders which wait till reaches the VIP wait limit (urgent orders), if no available cooks then take from break then from injured.   1. **Print\_Cooks\_In\_break**   **Member** **of**: Class Restaurant  **Inputs**:  No inputs  **Returns**:  Void function  **Called** **By**:   * Restaurant::RunSimulation()   **Calls**:   * Restaurant:: Print\_Cooks\_In\_Break();   **Function** **Logic** **description**:  This function prints ID of cooks in break.   1. **Print\_number\_of\_avalible\_cook**   **Member** **of**: Class Restaurant  **Inputs**:  No inputs  **Returns**:  Void function  **Called** **By**:   * Restaurant::RunSimulation()   **Calls**:   * Restaurant:: Print\_number\_of\_avalible\_cook ();   **Function** **Logic** **description**:  This function prints in status bar the number of available cooks in each type.   1. **FinshingOfDrawingAllDoneOrders**   **Member** **of**: Class Restaurant  **Inputs**:  No inputs  **Returns**:  Void function  **Called** **By**:   * Restaurant::RunSimulation()   **Calls**:   * Restaurant:: FinshingOfDrawingAllDoneOrders();   **Function** **Logic** **description**:  This function prints in status bar the number of done orders and draw them in the GUI.   1. **DrawAllCooks**   **Member** **of**: Class Restaurant  **Inputs**:  No inputs  **Returns**:  Void function  **Called** **By**:   * Restaurant::RunSimulation()   **Calls**:   * Restaurant::DrawAllCooks();   **Function** **Logic** **description**:  This function draws in GUI all available cooks according to their speed. | |
| **Section1: Taher Muhammed, 139**   1. **RunSimulation**   **Member** **of**: Class Restaurant  **Inputs**:  No inputs  **Returns**:  Void function  **Called** **By**:   * The main function   **Calls**:   * Restaurant:: RunSimulation ();   **Function** **Logic** **description**:  This function creates the GUI, asks about which mode will create and take names of input and output files. Calls all the function needed for the logic of the all modes, write in the output file and loops till all orders are done.   1. **Generate\_R**   **Member** **of**: Class Restaurant  **Inputs**:  No inputs  **Returns**:  Void function  **Called** **By**:   * Restaurant:: RunSimulation ();   **Calls**:   * Restaurant:: Generate\_R();   **Function** **Logic** **description**:  This function generates a random number and if this number is less than injuring probability read from the input file then choose the first order in serving list and injure its cook.   1. **heapify**   **Member** **of**: Class Restaurant  **Inputs**:  arr: Order pointer  n: number of array elements  root: number of root position  **Returns**:  Void function  **Called** **By**:   * Restaurant:: heapSortForWaitOrders(Order\* arr[], int n);   **Calls**:   * Restaurant:: heapify(arr,i,0); * Restaurant:: heapify(arr,n,i);   **Function** **Logic** **description**:  This function turns data into heap to facilitate its sort, it reheaps the data up till make the maximum number in the root position.   1. **heapSortForWaitOrders**   **Member** **of**: Class Restaurant  **Inputs**:  arr: Order pointer  n: number of array elements  **Returns**:  Void function  **Called** **By**:   * Restaurant:: DrawingAllWaitingOrders (int mode);   **Calls**:   * Restaurant:: heapSortForWaitOrders(arr, n);   **Function** **Logic** **description**:  This function uses heap sort algorithm to sort waiting orders in order to print each type of orders sorted with arrival time.   1. **Dynamic\_Deallocation**   **Member** **of**: Class Restaurant  **Inputs**:  No inputs  **Returns**:  Void function  **Called** **By**:   * Restaurant:: RunSimulation ();   **Calls**:   * Restaurant:: Dynamic\_Deallocation();   **Function** **Logic** **description**:  This function deallocates all cooks and orders at the end of the simulation.   1. **DrawingAllWaitingOrders**   **Member** **of**: Class Restaurant  **Inputs**:  mode: integer represents the number of the mode  **Returns**:  Void function  **Called** **By**:   * Restaurant::RunSimulation()   **Calls**:   * Restaurant:: DrawingAllWaitingOrders (mode);   **Function** **Logic** **description**:  This function draws waiting orders ID in the GUI, each type sorted with arrival time. Also prints the number of each type of waiting orders in the GUI status bar. | |