DATA STRUCTURES AND ALGORITHMS

GROUP PROJECT:

: RESTAURANT MANGEMENT SYSTEM:

SUBMITTED BY:

- FAIZAN HAIDER SP22-BCT-030.
- RANA MUHAMMAD ABDULLAH SP22-BCT-039.
- RANA MUHAMMAD MUDASSIR SP22-BCT-040.

SUBMITTED TO:

- Mr. MUHAMMAD MUSTAFA KHATTAK.

TABLE OF CONTENTS.

DECLARATION	ABSTRACT	INTRODUCTION	ALGORITHM	PRE AND POST CONDTIONS	PUESDO CODE
*WE SHOW WHAT OUR PROJECT NAME AND DECLARE IT.	*WE SHOW HOW OUR PROJECT WORKS AND THE PROJECT IS.	*BACKGROUND *MOTIVATION *PROJECT OVERVIEW	*STEPS OF OUR PROJECT. *HOW OUR WORK IS DIVIDED	*We explain what things are required for function and what the output is.	*We write the pseudo code like short code of program that what we are doing.

- **STEPS BY STEP SEQUENCE OF PROJECT:**

1. **DECLARTION.**
2. **ABTSRACT.**
3. **INTRODUCTION.**
4. **ALGORITHM.**
5. **Pseudo Code.**
6. **Pre conditions and Post conditions.**
7. **Complete code of program.**

- **DECLARTION OF PROJECT:**

We declare the Project as "**RESTAURANT MANGEMENT SYSTEM**" which we made under the provisions of our Professor Dr. Muhammad Mustafa Khattak in COMSATS University Islamabad.

- **ABSTRACT:**

In our Project "Restaurant Management System" we made a restaurant in which we a person can manage a restaurant by various functions. We made many functions in our system like:

-A manager can manage menu.

-A manager can manage orders given by customers.

-A manager manages the customers and keep record of the customers at restaurant.

-We can Delete or Add item in our menu.

-A manager can add or remove tables from the restaurant, or add or remove items from the menu and can analize orders of the day.

Furthermore, functions are also included in our Project.

- **Data structures and their purpose:**

1) SINGLY LINKLIST MENU MANAGEMENT
2) QUEUES ORDER MANAGEMENT
3) AVL TREES TABLE MANAGEMENT
4) MAX HEAP TABLE RESERVATION
5) GRAPHS DELIVERY MANAGEMENT
6) STACKS STORING OVERALL ORDERS.

- **REASONS TO USE THESE DATA STRUCTURES:**

Singly Linked List:

We use singly Linked List in menu management because singly Linked List are easily traversed.

Due to which we can easily insert and delete value.

Queues:

We use queues in order management because queues follow the mechanism FIFO (First in First Out) that' why the customer who will order first will be first served.

AVL TREES:

We use AVL trees in table management because the tables will be stored according to their capacity so the table with the largest capacity will be shown first to the customer.

MAX HEAP:

We use Max heap in table reservation because max heap will store the reservation of person with highest age first and then so on.

GRAPHS:

We use Graphs in delivery management system and implement it through adjacency matrix because already defined our locations of delivery and to find the shortest path using DIJSKTRA'S algorithm.

STACKS:

We use stack in storing overall orders of the day because to check the recent order as it is on top of the stack and so on, we can all the orders of the day one by one.

- # INRODUCTION:

BACKGROUND

The Project Restaurant Management System is about a restaurant in which a person can manage different things whether it is order management or staff management. This is program is very easy to use and a person which has little knowledge about C++ can easily use this program. This shows how user-friendly this program is.

MOTIVATION

The idea of the Project Restaurant Management system struck in our mind in a way that we to manage things and in university we also manage many events and restaurant is a place where management is always required and in future we shall open a restaurant and thus we decided to make a **Restaurant management system** as our project so it can help us in future.

- Professor Masoom Allam is Cyber Security Analyst and a teacher in COMSATS University Islamabad

Covered the FIFA WORLDCUP 2022 Cyber Security and they told us to make such projects that will help you in future, so that thing inspired us to make this Project.

- **PROJECT-OVERVIEW**

Our Project is made of C++ things such as Methods, Arrays, doubly linked list, priority queue, BST, AVL, Stack and Max heap. This program is equipped by following:

METHODS	**ARRAYS**	**GRAPHS**	**QUEUES**	**BST**	**AVL**
STACK	**MAX HEAP**	**LOOPS**	**LINKED LIST**	**Functions**	**ALGORITHMS**

-STEP BY STEP PROCEDURES OF PROGRAM ARE AS FOLLOWS:

Our program runs in very chronological way which we described as follows:

- **ALGORITHM:**

THE ALGORITHM OF PROGRAM IS:

1. Start by including the necessary header files and defining constants.
2. Declare and initialize the graph and locations arrays.
3. Define the menu structure and create the menu linked list functions.
4. Define the order structure and create the order linked list functions.
5. Define the table structure and create the table binary search tree functions.
6. Implement the insertTable function to insert tables into the binary search tree.
7. Implement the rotateLeft and rotateRight functions to perform tree rotations.
8. Implement the insertTableNode function to insert nodes into the binary search tree while maintaining the BST property.
9. Implement the generateMenu function to generate the menu linked list with food items and prices.
10. Implement the displayMenu function to display the menu to the user.
11. Implement the findFoodPrice function to find the price of a food item in the menu.
12. Implement the takeOrder function to allow the user to place an order.
13. Implement the displayOrder function to display the current order to the user.
14. Implement the findShortestPath function to find the shortest path between two locations using Dijkstra's algorithm.
15. Implement the handleDeliveryOrder function to handle delivery orders.

16. Implement the displayServeOrder function to display and serve orders.
17. Implement the manageReservation function to manage reservations using a max-heap.
18. Implement the main function.
19. In the main function, initialize the table binary search tree, menu linked list, and reservation max-heap.
20. Display the welcome message and available options to the user.
21. Enter a loop to continuously prompt the user for options until they choose to exit.
22. Inside the loop, prompt the user for their choice and execute the corresponding functionality based on the input.
23. If the user chooses to insert a table, prompt for the table number and call the insertTable function.
24. If the user chooses to display the menu, call the displayMenu function.
25. If the user chooses to place an order, call the takeOrder function.
26. If the user chooses to display the current order, call the displayOrder function.
27. If the user chooses to handle a delivery order, call the handleDeliveryOrder function.
28. If the user chooses to display and serve orders, call the displayServeOrder function.
29. If the user chooses to manage reservations, call the manageReservation function.
30. If the user chooses to exit, display a goodbye message and break out of the loop.

- **PRE CONDITIONS AND POST CONDITIONS OF PROGRAM:**

Node class constructor:

Precondition: None

Postcondition: None

BinarySearchTree class constructor:

Precondition: None

Postcondition: None

BinarySearchTree class method insert:

Precondition: None

Postcondition: None

BinarySearchTree class method rotateLeft:

Precondition: None

Postcondition: None

BinarySearchTree class method rotateRight:

Precondition: None

Postcondition: None

LinkedList class constructor:

Precondition: None

Postcondition: None

LinkedList class method generateMenu:

Precondition: None

Postcondition: None

LinkedList class method displayMenu:

Precondition: None

Postcondition: None

LinkedList class method getPrice:

Precondition: The given food item exists in the menu.

Postcondition: The price of the food item is returned.

Order class method takeOrder:

Precondition: None

Postcondition: The user's order is taken and added to the current order list.

Order class method displayOrder:

Precondition: None

Postcondition: The current order list is displayed to the user.

Graph class method dijkstra:

Precondition: The graph and locations arrays are properly initialized.

Postcondition: The shortest path between two given locations is found using Dijkstra's algorithm.

Delivery class method processOrder:

Precondition: The delivery order is valid and contains valid locations.

Postcondition: The delivery order is processed and handled appropriately.

ServeOrder class method displayServeOrders:

Precondition: None

Postcondition: The serve order list is displayed, and the served orders are removed from the list.

Reservation class constructor:

Precondition: None

Postcondition: None

Reservation class method manageReservation:

Precondition: None

Postcondition: The reservations are managed using a max-heap data structure.

Restaurant class constructor:

Precondition: None

Postcondition: None

Restaurant class method main:

Precondition: None

Postcondition: None

Restaurant class method run:

Precondition: None

Postcondition: None

Restaurant class method displayMenu:

Precondition: None

Postcondition: None

Restaurant class method reserveTable:

Precondition: The table number to be inserted is valid and not already present in the binary search tree.

Postcondition: The table is inserted into the binary search tree in the appropriate position.

Restaurant class method takeOrder:

Precondition: None

Postcondition: None

Restaurant class method displayOrder:

Precondition: None

Postcondition: None

Restaurant class method processDelivery:

Precondition: The delivery order is valid and contains valid locations.

Postcondition: The delivery order is processed and handled appropriately.

Restaurant class method serveOrder:

Precondition: None

Postcondition: The serve order list is displayed, and the served orders are removed from the list.

Restaurant class method manageReservation:

Precondition: None

Postcondition: The reservations are managed using a max-heap data structure.

Restaurant class method exitProgram:

Precondition: None

Postcondition: None

- ## PSEUDO CODE OF PROGRAM:

- Here is the pseudo code of the program we made:

1. Declare the required data structures and variables.

2. Define functions for managing the menu:

- insert_Menu:
  - Create a new menu item.
  - Get the name and price of the food item from the user.
  - Insert the new item into the menu linked list.

- display_Menu:
  - Traverse the menu linked list and display each food item along with its price.

- find_food:
  - Take a food name as input.
  - Traverse the menu linked list to find the corresponding food item.
  - If found, return the food name; otherwise, indicate that the food is not available.

- find_price:
  - Take a food name as input.
  - Traverse the menu linked list to find the corresponding food item.
  - If found, return the price; otherwise, indicate that the food is not available.

3. Define functions for managing orders:
- take_Order:
  - Create a new order.
  - Get the table number from the user.
  - Prompt the user to enter food items until they enter "end" to finish.
  - For each food item entered, find its name and price using the find_food and find_price functions.
  - Update the total order and price in the new order.
  - Push the total price of the order into the total linked list.
  - Add the new order to the order linked list.

- display_Order:
  - Get the table number from the user.
  - Find the order in the order linked list based on the table number.
  - Traverse the order's total order string and display each food item along with its price.
  - Display the total bill for the order.

- servefood:
  - Check if there are any orders in the order linked list.
  - If yes, remove the first order from the order linked list.

4. Define functions for managing tables:
- addnewTable:
  - Create a new table node with the given capacity.
  - Initialize its left and right pointers to NULL.

- Return the new node.

- getrowOfTable:
  - Traverse the table AVL tree to find the row number of the given table number.
  - Return the row number.

- getBalanceFactor:
  - Calculate the balance factor of the given table node.
  - Return the balance factor.

- rightRotate:
  - Perform a right rotation on the given table node.
  - Update the left and right pointers accordingly.

- leftRotate:
  - Perform a left rotation on the given table node.
  - Update the left and right pointers accordingly.

- insert:
  - Take the root node and the new table node as input.
  - If the root node is NULL, return the new table node as the root.
  - If the new table's number is less than the current node's number, recursively call the function on the left subtree.
  - If the new table's number is greater than the current node's number, recursively call the function on the right subtree.
  - Update the height of the current node.
  - Calculate the balance factor of the current node.
  - If the balance factor is less than -1 or greater than 1, perform rotations.
  - Return the modified root node.

- preorder_traversal:
  - Perform a preorder traversal of the table AVL tree.
  - Display the details of each table node.

5. Define functions for managing reservations:
   - insert:
     - Take the max heap and the new reservation as input.
     - Insert the new reservation into the max heap.
     - Reorder the heap to maintain the heap property.

   - deleteRoot:
     - Remove the root (max value) from the max heap.
     - Reorder the heap to maintain the heap property.

   - displayReservations:
     - Display all the reservations in the max heap.

6. Define the main function:
   - Create the root nodes for staff and table.
   - Loop until the user chooses to exit:
     - Display the menu of options for the user.
     - Prompt the user to choose an option.
     - Based on the user's choice, call the corresponding functions to perform the desired operation.
   - Exit the program.

- **COMPLETE CODE OF PROGRAM:**

```
#include<iostream>
#include<vector>
using namespace std;

```

```
#define inf 99
#define vt 7
int graph[vt][vt] = {
        {0, 2, 1, 2, 0, 0, 0},
        {2, 0, 1, 0, 2, 0, 0},
        {1, 1, 0, 1, 0, 2, 0},
        {2, 0, 1, 0, 0, 0, 2},
        {0, 2, 0, 0, 0, 1, 0},
        {0, 0, 2, 0, 1, 0, 1},
        {0, 0, 0, 2, 0, 1, 0},
    };
string locations[vt] = {"Restaurant","G6","G7","G8","F6","F7","F8"};


struct menu{
    string food;
    int price;
    int count = 10;
    menu*next = NULL;
};
menu*menufirst = NULL;
menu*menulast = NULL;

struct order{
    int tableNo;
    string totalorder;
    int totalprice;
    order*next = NULL;
};
order*orderfront = NULL;
order*orderrear = NULL;

struct table
{
    int number;
    bool occupied;
    int row;
    int capacity;
    table *left;
    table *right;
};
table *tableroot = NULL;
int count = 0;

struct total{
```

```
    int orderNo;
    int price;
    total *next = NULL;
};
int totalcount = 0;
total* totalirst = NULL;

void insert_Menu(string name,int cost){

    menu * newNode = new menu;
    newNode->food = name;
    newNode->price = cost;
    if (menufirst == NULL)
    {
        menufirst = menulast = newNode;
    }
    else{
    menulast->next = newNode;
    menulast = newNode;
    }
}
void display_Menu(){
    if (menufirst == NULL)
    {
        cout<<"menu is empty"<<endl;
        return;
    }
    menu*p = menufirst;
    cout<<"NAME"<<" ---- "<<"PRICE"<<endl;
    while (p != NULL)
    {
        cout<<p->food<<" ---- "<<p->price<<endl;
        p = p->next;
    }
}

void generatMenu(){

    string n1 = "Biryani";
    int p1 = 200;
    insert_Menu(n1,p1);
    string n2 = "Karahi";
    int p2 = 1600;
    insert_Menu(n2,p2);
    string n3 = "Bar-b-que";
```

```
    int p3 = 350;
    insert_Menu(n3,p3);
    string n4= "Nihari";
    int p4 = 800;
    insert_Menu(n4,p4);
    string n5 = "Naan";
    int p5 = 40;
    insert_Menu(n5,p5);

}
void pushTotal(int price){
    total * newNode = new total;
    totalcount++;
    newNode->orderNo = totalcount;
    newNode->price = price;
    if (totalirst == NULL)
    {
        totalirst = newNode;
    }
    else
    {
        newNode->next = totalirst;
        totalirst = newNode;
    }
}
void displayTotal(){
    if (totalirst == NULL)
    {
        cout<<"no orders today"<<endl;
        return;
    }
    else
    {
        total * temp = totalirst;
        while (temp != NULL)
        {
            cout<<"Order Number : "<<temp->orderNo<<" Price : "<<temp-
>price<<endl;
            temp = temp->next;
        }
    }
}
void freeOrderList(){
    total *p;
    if (totalirst== NULL)
```

```
    {
        cout<<"no orders today"<<endl;
    }
    else{
     while (totalirst != NULL)
     {
        p = totalirst;
        totalirst = totalirst->next;
        free (p);
     }
     cout<<"today's all orders clear"<<endl;
    }
}

string find_food(string name){
    if (menufirst == NULL)
    {
        cout<<"menu is empty"<<endl;
        return "";
    }
    menu*p = menufirst;
    while (p != NULL && p->food != name)
    {
        p = p->next;
    }
    if (p == NULL)
    {
        cout<<name<<" is not available"<<endl;
        return "";
    }
    else{
        p->count = p->count-1;
        return p->food;
    }
}
int find_price(string name){
    if (menufirst == NULL)
    {
        cout<<"menu is empty"<<endl;
        return 0;
    }
    menu*p = menufirst;
    while (p != NULL && p->food != name)
    {
        p = p->next;
```

```
    }
    if (p == NULL)
    {
        return 0;
    }
    else{
        return p->price;
    }
}
table *addnewTable(int key){
    table* temp = new table;
    temp->occupied = true;
    temp->number = key;
    temp->capacity = 15;
    temp->left = NULL;
    temp->right = NULL;
    temp->row = 1;
    return temp;
}
int getrowOfTable(table* n) {
    if (n == NULL) {
        return -1;
    }
    return n->row;
}
int getBalanceFactor(table* n) {
    if (n == NULL) {
        return 0;
    }
    return getrowOfTable(n->left) - getrowOfTable(n->right);
}
table* rightRotate(table* y) {
    table* x = y->left;
    table* T2 = x->right;

    x->right = y;
    y->left = T2;

    y->row = 1 + max(getrowOfTable(y->right), getrowOfTable(y->left));
    x->row = 1 + max(getrowOfTable(x->right), getrowOfTable(x->left));

    return x;
}
table* leftRotate(table* x) {
    table* y = x->right;
```

```
    table* T2 = y->left;

    y->left = x;
    x->right = T2;

    y->row = 1 +max(getrowOfTable(y->right), getrowOfTable(y->left));
    x->row = 1 +max(getrowOfTable(x->right), getrowOfTable(x->left));

    return y;
}
table* insert(table* node, int key) {
    if (node == NULL) {
        return addnewTable(key);
    }

    if (key < node->number) {
        node->left = insert(node->left, key);
    } else if (key > node->number) {
        node->right = insert(node->right, key);
    } else {
        return node;
    }

    node->row = 1 + max(getrowOfTable(node->left), getrowOfTable(node->right));
    int balanceFactor = getBalanceFactor(node);

    if (balanceFactor > 1 && key < node->left->number) {
        return rightRotate(node);
    }

    if (balanceFactor < -1 && key > node->right->number) {
        return leftRotate(node);
    }
    if (balanceFactor > 1 && key > node->left->number) {
        node->left = leftRotate(node->left);
        return rightRotate(node);
    }

    if (balanceFactor < -1 && key < node->right->number) {
        node->right = rightRotate(node->right);
        return leftRotate(node);
    }

    return node;
```

```
}
void preorder_traversal(table* root) {
    if (root != nullptr) {
        cout << root->capacity <<"  "<<root->number<<"  "<<root-
>occupied<<endl;
        preorder_traversal(root->left);
        preorder_traversal(root->right);
    }
}
table* checkTabel(table* node, int key) {
    if (node == NULL || node->number == key) {
        return node;
    }

    if (key < node->number) {
        return checkTabel(node->left, key);
    } else {
        return checkTabel(node->right, key);
    }
}
order*gettableNo(int key){
    order*temp = orderfront;
    while (temp != NULL && temp->tableNo != key)
    {
        temp = temp->next;
    }
    if (temp == NULL)
    {
        cout<<"table not found"<<endl;
    }
    else{
        return temp;
    }
}

void take_Order(int table){
    order *newOrder= new order;

    int totalPriceOfOrder = 0;
    string totalOrderGiven;
    newOrder->tableNo = table;
    string order;
    cout<<"enter the food customer wants"<<endl;
    cin>>order;
    while (order != "end" )
```

```
    {
        totalOrderGiven = totalOrderGiven+","+(find_food(order));
        totalPriceOfOrder = totalPriceOfOrder + find_price(order);
        cout<<"enter the food customer wants"<<endl;
        cin>>order;
    }
    newOrder->totalprice = totalPriceOfOrder;
    pushTotal(newOrder->totalprice);
    totalOrderGiven = totalOrderGiven+",end";
    newOrder->totalorder = totalOrderGiven;
    if (orderfront == NULL)
    {
        orderfront = orderrear = newOrder;
    }
    else{
        orderrear->next = newOrder;
        orderrear = newOrder;
    }

}
int minDistance(int dist[vt], bool tarr[]) {
    int min = 9999999;
    int ind;

    for (int i = 0; i < vt; i++) {
        if (tarr[i] == false && dist[i] <= min) {
            min = dist[i];
            ind = i;
        }
    }
    return ind;
}

void printPath(int currentVertex, vector<int>& parents) {
    if (currentVertex == -1) {
        return;
    }
    printPath(parents[currentVertex], parents);
    if (currentVertex == 0) {
        cout << "";
    } else {
        cout << locations[currentVertex] << " ";
    }
}
```

```
void find_shortest_path(string n) {
    bool tarr[vt];
    int newdist[vt];
    vector<int> parents(vt);

    for (int i = 0; i < vt; i++) {
        newdist[i] = inf;
        tarr[i] = false;
        parents[i] = -1;
    }

    newdist[0] = 0;

    for (int i = 0; i < vt; i++) {
        int m = minDistance(newdist, tarr);
        tarr[m] = true;

        for (int k = 0; k < vt; k++) {
            if (!tarr[k] && graph[m][k] > 0 && newdist[m] + graph[m][k] <
newdist[k]) {
                parents[k] = m;
                newdist[k] = newdist[m] + graph[m][k];
            }
        }
    }
    cout << "Desired Sector\tShortet Distance\tpath" << endl;
    for (int i = 0; i < vt; i++) {
        if(locations[i] == n){
        cout << locations[i]<< "\t\t";
        cout << newdist[i] << "\t\t\t";
        printPath(i, parents);
        cout << endl;
        }
    }
}
void delivery_Order(string sector){
     order *newOrder= new order;

    int totalPriceOfOrder = 0;
    string totalOrderGiven;

    newOrder->tableNo = 0;
    string order;
    cout<<"enter the food customer wants"<<endl;
    cin>>order;
```

```
    while (order != "end" )
    {
        totalOrderGiven = totalOrderGiven+","+(find_food(order));
        totalPriceOfOrder = totalPriceOfOrder + find_price(order);
        cout<<"enter the food customer wants"<<endl;
        cin>>order;
    }
    newOrder->totalprice = totalPriceOfOrder;
    pushTotal(newOrder->totalprice);
    totalOrderGiven = totalOrderGiven+",end";
    newOrder->totalorder = totalOrderGiven;
    if (orderfront == NULL)
    {
        orderfront = orderrear = newOrder;
    }
    else{
        orderrear->next = newOrder;
        orderrear = newOrder;
    }
    cout<<"The shortest path To reach "<<sector<<"is :"<<endl;
    find_shortest_path(sector);
}
void display_Order(){
    if (orderfront == NULL)
    {
        cout<<"no orders has been taken"<<endl;
        return;
    }
    int t;
    cout<<"which table order you want to dispaly"<<endl;
    cin>>t;
    order *temp = gettableNo(t);

    cout<<"NAME"<<endl;
    string disordere;
    int disprice;
    temp->totalorder.erase(0,1);
    while (temp->totalorder != "end")
    {
        int pos = temp->totalorder.find(',');
        disordere = temp->totalorder.substr(0,pos);
        disprice = find_price(disordere);
        temp->totalorder.erase(0,pos);
        cout<<disordere<<" ---- "<<disprice<<endl;
        temp->totalorder.erase(0,1);
```

```
    }
    cout<<"TOTAL BILL"<<endl;
    cout<<temp->totalprice<<endl;

}
void servefood(){
    if (orderfront == NULL)
    {
        cout<<"order has not been taken"<<endl;
        return;
    }
    else{
        if (orderfront == orderrear)
        {
            order*temp = orderfront;
            orderfront = orderrear = NULL;
            delete temp;
        }
        else{
            order*temp = orderfront;
            orderfront = temp->next;
            delete temp;
        }
        cout<<"the order has been serverd"<<endl;
    }
}

#define MAX 100

int reservation[MAX];
int n = 0;

void makeReservation(int & n, int persons) {
    // Step 1: Add the new value and set its POS
    n = n + 1;
    int pos = n;
    reservation[n] = persons;
    while (pos > 1) {
        int par = pos / 2;
        if (reservation[pos] <= reservation[par]) {
            return;
        } else if (reservation[pos] > reservation[par]) {
            int temp = reservation[pos];
            reservation[pos] = reservation[par];
            reservation[par] = temp;
```

```
            pos = par;
        }
    }
}
void deleteReservation(int heap[], int& n) {
    int last = heap[n - 1];
    n = n - 1;

    int ptr = 1, left = 2, right = 3;

    heap[ptr] = last;

    while (left <= n) {
        if (right <= n) {
            if (heap[ptr] < heap[left] || heap[ptr] < heap[right]) {

                if (heap[right] >= heap[left]) {
                    swap(heap[ptr], heap[right]);
                    ptr = right;
                } else {
                    swap(heap[ptr], heap[left]);
                    ptr = left;
                }
            } else {
                break;
            }
 } else if (heap[ptr] < heap[left]) {
            swap(heap[ptr], heap[left]);
            ptr = left;
        } else {

            break;
        }

        left = 2 * ptr;
        right = left + 1;
    }
    cout<<"Reservation Deleted Sucessfully"<<endl;
}

void displayReservations(int &n){

    for (int i = 1; i <= n ; i++)
    {
        cout<<"reservation Of Age :"<<reservation[i]<<endl;
```

```
    }
    cout<<endl;
}

void orderManagement() {
    int choiceForOrderManagement;

    do {
        cout << "PRESS 1 TO TAKE ORDER" <<endl;
        cout << "PRESS 2 TO DISPLAY ORDER" << endl;
        cout << "PRESS 3 TO SERVE FOOD" <<endl;
        cout << "PRESS 0 TO EXIT ORDER MANAGEMENT" <<endl;
        cin >> choiceForOrderManagement;

        switch (choiceForOrderManagement) {
            case 1:{
                cout<<"Enter 1 if you want home delivery :"<<endl;
                int dil;
                cin>>dil;
                if (dil == 1)
                {
                cout<<"Enter your sector : "<<endl;
                string sec;
                cin>>sec;
                delivery_Order(sec);
                break;

                }else
                {
                    int tableNo;
                    cout<<"enter table Number"<<endl;
                    cin>>tableNo;

                    table *check = checkTabel(tableroot,tableNo);
                    do
                    {
                        int tableNo;
                        cout<<"enter table Number"<<endl;
                        cin>>tableNo;

                        if (check != NULL)
                        {
                            take_Order(tableNo);
                            break;
                        }else{
```

```
                            cout<<"table is not present\nPlease enter
valid table number"<<endl;
                            cout<<endl;
                            cout<<"The present Table are :"<<endl;
                            preorder_traversal(tableroot);
                        }
                    } while (check != NULL);
                break;
                }
            }
            case 2:{
                display_Order();
                break;
            }
            case 3:{
                servefood();
                break;
            }
            case 0:
                break;
            default:
                cout << "PRESS A VALID BUTTON BETWEEN 0 - 3" <<endl;
        }

    } while (choiceForOrderManagement != 0);
}

void staffManagement(){
    int choice;
    do
    {
        cout << "PRESS 1 TO ADD NEW DISH" << endl;
        cout << "PRESS 2 TO ADD NEW TABLE" << endl;
        cout << "PRESS 3 TO DISPLAY TOTAL ORDERS TODAY" << endl;
        cout << "PRESS 0 TO EXIT" << endl;
        cin>>choice;

        switch (choice) {
        case 1: {
            string foodName;
            int price;
            cout << "Enter name of food: " << endl;
            cin >> foodName;
            cout << "Enter its price: " << endl;
            cin >> price;
```

```
            insert_Menu(foodName, price);
            break;
        }
        case 2: {
            int tableNumber;
            cout << "Enter table number: " << endl;
            cin >> tableNumber;
            tableroot = insert(tableroot,tableNumber);
            int takeOrder;
            cout<<"enter 1 to take order at table"<<endl;
            cin>>takeOrder;
            if (takeOrder == 1)
            {
                take_Order(tableNumber);
            }

            break;
        }
        case 3:
            displayTotal();
            break;
        case 0:
            break;
        default:
            cout << "PRESS VALID BUTTON BETWEEN 0 - 3" << endl;
            break;
    }
} while (choice != 0);


}
void reservationManagement(){
    int choice;
    do
    {
        cout<<"PRESS 1  TO MAKE RESERVATION"<<endl;
        cout<<"PRESS 2 TO SHOW RESERVATION LIST"<<endl;
        cout<<"PRESS 3 TO DELETE RESERVATION"<<endl;
        cout<<"PRESS 0 TO EXIT RESERVATION MANAGEMENT"<<endl;

        cin>>choice;

        switch (choice) {
        case 1: {
            int ageOfCustomer;
```

```
            cout<<"What is age of Customer"<<endl;
            cin>>ageOfCustomer;
            makeReservation(n,ageOfCustomer);
            break;
        }
        case 2: {
            displayReservations(n);
            break;
        }
        case 3:
            deleteReservation(reservation,n);
            break;
        case 0:
            break;
        default:
            cout << "PRESS VALID BUTTON BETWEEN 0 - 3" << endl;
            break;
    }
} while (choice != 0);


}

int main(){
    generatMenu();
    int choice;
do {
    cout << "PRESS 1  FOR MENU MANAGEMENT" << endl;
    cout << "PRESS 2 FOR ORDER MANAGEMENT" << endl;
    cout << "PRESS 3 FOR RESERVATION MANAGEMENT" << endl;
    cout << "PRESS 4 FOR STAFF MANAGEMENT" << endl;
    cout << "PRESS 0 TO EXIT" << endl;

    cin >> choice;

    switch (choice) {
        case 1:
            display_Menu();
            break;
        case 2:
            orderManagement();
            break;
        case 3:
            reservationManagement();
            break;
```

```
            case 4:
                staffManagement();
                break;
            case 0:
                cout << "Exiting..." << endl;
                break;
            default:
                cout << "PRESS VALID BUTTON BETWEEN 0 - 5" << endl;
                break;
        }
    } while (choice != 0);

    return 0;

    }
```