

Name :- Ishtiaq Ahmad Khan

Roll.No:- SP24-BCS-053

Section B

DSA PROJECT

Food Ordering Management System:-

CODE:

```
#include <iostream>

#include <stdlib.h>

#include <bits/stdc++.h>

const int infinity = INT_MAX;

using namespace std;
```

```
// Defining Data Types
```

```
struct customer
{
    int age;
```

```
string name;  
string dishname;  
int quantity;  
double bill;
```

```
customer() {}
```

```
customer(int age, string name, int quantity, string dishname, double bill)  
{  
    this->age = age;  
    this->name = name;  
    this->dishname = dishname;  
    this->quantity = quantity;  
    this->bill = bill;  
}  
};
```

```
struct takeAwayCustomer
```

```
{  
    customer cusotomer;  
    takeAwayCustomer *next = NULL;
```

```
takeAwayCustomer(int age, string name, int quantity, string dishname, double bill)  
{  
    this->cusotomer = customer(age, name, quantity, dishname, bill);  
}
```

```
};
```

```
struct dineInCustomer
```

```
{
```

```
    customer cusotomer;
```

```
    dineInCustomer *next = NULL;
```

```
    dineInCustomer(int age, string name, int quantity, string dishname, double bill)
```

```
{
```

```
    this->cusotomer = customer(age, name, quantity, dishname, bill);
```

```
}
```

```
};
```

```
struct homeDeliveryCustomer
```

```
{
```

```
    customer cusotomer;
```

```
    string address;
```

```
    double deliveryCharges;
```

```
    int distanceDelivery;
```

```
    struct homeDeliveryCustomer *next = NULL;
```

```
    homeDeliveryCustomer(int age, string name, int quantity, string dishname, double bill,  
string address, double deliveryCharges, int distanceDelivery)
```

```
{
```

```
    this->cusotomer = customer(age, name, quantity, dishname, bill);
```

```
    this->address = address;

    this->deliveryCharges = deliveryCharges;

    this->distanceDelivery = distanceDelivery;
}
};
```

```
struct Resturant
```

```
{

    string shopName;

    string *menu;

    int *price;

    string address;

    takeAwayCustomer *nextTakeAwayCustomer = NULL;

    dineInCustomer *nextDineInCustomer = NULL;

    homeDeliveryCustomer *nextHomeDeliveryCustomer = NULL;
};
```

```
// Globally declaring the pizza Shop's pointer
```

```
Resturant *myResturant = NULL;
```

```
// Globally Declaring the Current Customer's Pointers for all three Types
```

```
takeAwayCustomer *currentTakeAwayCustomer = NULL;
```

```
dineInCustomer *currentDineInCustomer = NULL;
```

```
homeDeliveryCustomer *currentHomeDeliveryCustomer = NULL;
```

```
// Globally declaring the variables for the total of all the orders in queue!
double total, takeAway, dineIn, homeDelivery;

// Globally declaring the variables for the total of all the orders served!
double servedTotal;


// In case of Serving , to keep the record of Customers Served, implementing AVL Tree for
efficient Search

// to search the record of Customers by Name

// It can also Display all the customers Served


struct servedCustomer
{

    int age;
    string name;
    int quantity;
    string dishname;
    double bill;
    string customerType;
    servedCustomer *left;
    servedCustomer *right;


    // Constructor
    servedCustomer(int age, string name, int quantity, string dishname, double bill, string
customerType)
    {
```

```
    this->age = age;

    this->name = name;

    this->quantity = quantity;

    this->dishname = dishname;

    this->bill = bill;

    this->customerType = customerType;

    // child to NULL

    this->left = NULL;

    this->right = NULL;
}
};

servedCustomer *root = NULL; // Global pointer for the root of AVLTree


// isEmpty
int isEmpty(servedCustomer *root)
{
    return root == NULL;
}


// display Customers Details

void display(servedCustomer *root)
{
    cout << "Name :> " << root->name << endl;
```

```
    cout << "Age :" << root->age << endl;
    cout << "Pizza :" << root->dishname << endl;
    cout << "Quantity : " << root->quantity << endl;
    cout << "Bill : " << root->bill << endl;
    cout << "Customer Type: " << root->customerType << endl;
}

// Traversal for the served Customers

servedCustomer *displayAllServedOrders(servedCustomer *root)
{

    if (root)
    {
        displayAllServedOrders(root->left);
        display(root);
        displayAllServedOrders(root->right);
    }

    return root;
}

// Height of servedCustomer tree

int height(servedCustomer *root)
{
```

```
    if (!root)
        return 0;

    return max(height(root->left), height(root->right)) + 1;
}

// Balance Factor for each ServedCustomer node

int balanceFactor(servedCustomer *root)
{
    if (!root)
        return 0;

    return height(root->left) - height(root->right);
}

// Maximum of two integers as a helper function for height
int max(int a, int b)
{
    return (a > b) ? a : b;
}

// Searching in servedCustomer tree

servedCustomer *search(servedCustomer *root, string keyName)
{

```



```
if (root == NULL)
{
    return NULL;
}
else if (root->name == keyName)
{
    return root;
}
else if (root->name < keyName)
{
    return search(root->right, keyName);
}
else if (root->name > keyName)
{
    return search(root->left, keyName);
}

return root;
}
```

// Finding Maximum Node of servedCustomer tree

```
servedCustomer *maxservedCustomer(servedCustomer *root)
{
    // Maximum Node is Present in the most Right Node
```

```
servedCustomer *p = root;
servedCustomer *temp = NULL;

while (p != NULL)
{
    temp = p;
    p = p->right;
}

return temp;
}

// Balancing the ServedCustomer's Tree thorough AVL Rotations

// LL Rotation
servedCustomer *LLRotation(servedCustomer *root)
// rotate wese right per krna hai!
{
    // saving the new root and the lost element in case of rotation
    servedCustomer *x = root->left;
    servedCustomer *temp = x->right;

    // Performing rotation
    x->right = root;
    root->left = temp;
```

```

// updating the root
root = x;

// returning the root
return x;
}

// RR Rotation
servedCustomer *RRRotation(servedCustomer *root)
{
    // rotate wese left per krna hai!
    servedCustomer *x = root->right;
    servedCustomer *temp = x->left;

    // Performing rotation
    x->left = root;
    root->right = temp;

    // updating the root
    root = x;

    // returning the root
    return x;
}

// LR Rotation
servedCustomer *LRRotation(servedCustomer *root)

```

```
{  
    root->left = RRRotation(root->left);  
    return LLRotation(root);  
}
```

// RL Rotation

```
servedCustomer *RLRotation(servedCustomer *root)  
{  
    root->right = LLRotation(root->right);  
    return RRRotation(root);  
}
```

// INSERTION in servedCustomer Tree

```
servedCustomer *insertion(int age, string name, int quantity, string dishname, double bill,  
string customerType, servedCustomer *root)
```

```
{  
    servedCustomer *newNode = new servedCustomer(age, name, quantity, dishname, bill,  
customerType);
```

```
    if (root == NULL)
```

```
{  
    root = newNode;  
}
```

```
    else if (root->name > newNode->name)
```

```
{
```

```
    root->left = insertion(age, name, quantity, dishname, bill, customerType, root->left);
}
else if (root->name < newNode->name)
{
    root->right = insertion(age, name, quantity, dishname, bill, customerType, root->right);
}

else
{
    cout << "No duplicates Values are Allowed " << endl;
    return root;
}

int bf = balanceFactor(root);

if (bf == 2)
{
    // LL
    if (root->left->name > newNode->name)
    {
        return LLRotation(root);
    }

    // LR
    if (root->left->name < newNode->name)
    {
```

```

        return LRRotation(root);
    }
}
else if (bf == -2)
{
    // RR
    if (root->right->name < newNode->name)
    {
        return RRRotation(root);
    }

    // RL
    if (root->right->name > newNode->name)
    {
        return RLRotation(root);
    }
}

return root; // in case there is no need of rotation
}

servedCustomer *deleteNode(servedCustomer *root, string key)
{
    if (root == NULL)
        return root;
    else if (key < root->name)

```

```
    root->left = deleteNode(root->left, key);
else if (key > root->name)
    root->right = deleteNode(root->right, key);
else
{
    // if deleteroot has one child or zero child
    if ((root->left == NULL) || (root->right == NULL))
    {
        servedCustomer *temp = root->left ? root->left : root->right;

        if (temp == NULL)
        {
            temp = root;
            root = NULL;
        }
        else
            *root = *temp;

        delete (temp);
    }
    else
    {
        // if deleteroot has two or more childs
        servedCustomer *temp = maxservedCustomer(root->right);
        root->name = temp->name;
        root->right = deleteNode(root->right, temp->name);
    }
}
```

```
}
```

```
if (root == NULL)
```

```
    return root;
```

```
// getting the balance factor
```

```
int balance = balanceFactor(root);
```

```
// LEFT LEFT CASE
```

```
if (balance > 1 && key < root->left->name)
```

```
    return LLRotation(root);
```

```
// LEFT RIGHT CASE
```

```
if (balance > 1 && key > root->left->name)
```

```
{
```

```
    root->left = LLRotation(root->left);
```

```
    return LRRotation(root);
```

```
}
```

```
// RIHGT RIGHT CASE
```

```
if (balance < -1 && key > root->right->name)
```

```
    return RRRotation(root);
```

```
// RIGHT LEFT CASE
```

```
if (balance < -1 && key < root->right->name)
```

```
{
```



```

        return RLRotation(root);
    }

    return root;
}

void deleteAllServedCustomers(servedCustomer *root)
{

    while (root)
    {
        root = deleteNode(root, root->name);
    }

    cout << "The Served Customer's List is Cleared " << endl;
}

// Based on : Older person will be served first (PRIORITY QUEUE)

void placeOrderTakeAwayCustomer(int age, string name, string dishname, int quantity,
double bill)
{

    currentTakeAwayCustomer = new takeAwayCustomer(age, name, quantity, dishname,
bill);

    if (myResturant->nextTakeAwayCustomer == NULL)

```

```

{
    // if first then insert in front
    myResturant->nextTakeAwayCustomer = currentTakeAwayCustomer;
}
else
{
    // finding the last Node
    takeAwayCustomer *temp = myResturant->nextTakeAwayCustomer;
    while (temp->next != NULL)
    {
        temp = temp->next;
    }

    if (temp->cusotomer.age < currentTakeAwayCustomer->cusotomer.age)
    {
        // insert at front
        takeAwayCustomer *firstCustomer = myResturant->nextTakeAwayCustomer;
        myResturant->nextTakeAwayCustomer = currentTakeAwayCustomer;
        currentTakeAwayCustomer->next = firstCustomer;
    }
    else
    {
        // insert at end
        temp->next = currentTakeAwayCustomer;
        currentTakeAwayCustomer->next = NULL;
    }
}

```

```
}  
  
    cout << "Your Order has been Placed MR/MRS " << name << " and your order is " <<  
    dishname << " with " << quantity << " quantity and total bill is " << bill << endl;  
}
```

```
void serveOrderTakeAwayCustomer()  
{  
    if (myResturant->nextTakeAwayCustomer == NULL)  
    {  
        cout << "No Take Away Customer to Serve" << endl;  
    }  
    else  
    {  
        // serving the first customer  
        takeAwayCustomer *temp = myResturant->nextTakeAwayCustomer;  
        // if it has some next element  
        if(temp->next != NULL){  
            myResturant->nextTakeAwayCustomer = temp->next;  
        }  
        else{  
            myResturant->nextTakeAwayCustomer = NULL;  
        }  
  
        cout << "Take Away Customer Served : " << temp->cusotomer.name << endl;  
  
        string customerType = "Take-Away";
```

```

        // Now before deleting the node we need to update the servedCustomer Tree

        root = insertion(temp->cusotomer.age, temp->cusotomer.name, temp->cusotomer.quantity, temp->cusotomer.dishname, temp->cusotomer.bill, customerType, root);

        delete temp;
    }
}

```

// Based on : First Come First Served (FIFO) (QUEUE)

```

void placeOrderDineInCustomer(int age, string name, string dishname, int quantity, double bill)
{

    currentDineInCustomer = new dineInCustomer(age, name, quantity, dishname, bill);

    if (myResturant->nextDineInCustomer == NULL)
    {
        // if first insert in front
        myResturant->nextDineInCustomer = currentDineInCustomer;
    }
    else
    {
        // finding the last Node
        dineInCustomer *temp = myResturant->nextDineInCustomer;
        while (temp->next != NULL)

```

```

    {
        temp = temp->next;
    }

    temp->next = currentDineInCustomer;
    currentDineInCustomer->next = NULL;
}

cout << "Your Order has been Placed MR/MRS " << name << " and your order is " <<
dishname << " with " << quantity << " quantity and total bill is " << bill << endl;
}

void serveOrderDineInCustomer()
{
    if (myResturant->nextDineInCustomer == NULL)
    {
        cout << "No Dine-In Customer to Serve" << endl;
    }
    else
    {
        // serving the first customer
        dineInCustomer *temp = myResturant->nextDineInCustomer;
        if(temp->next != NULL){
            myResturant->nextDineInCustomer = temp->next;
        }
        else{
            myResturant->nextDineInCustomer = NULL;

```

```
}
```

```
cout << "Dine-In Customer Served : " << temp->cusotomer.name << endl;
```

```
string customerType = "Dine-In";
```

```
// Now before deleting the node we need to update the servedCustomer Tree
```

```
root = insertion(temp->cusotomer.age, temp->cusotomer.name, temp->cusotomer.quantity, temp->cusotomer.dishname, temp->cusotomer.bill, customerType, root);
```

```
delete temp; // deleting the customer
```

```
}
```

```
}
```

```
// Based on : (when all orders are ready)(LIFO)(Stack)
```

```
void placeOrderHomeDeliveryCustomer(int age, string name, string dishname, int quantity, double bill, string address, int deliveryCharges, int distanceDelivery)
```

```
{
```

```
    // making new Customer
```

```
    currentHomeDeliveryCustomer = new homeDeliveryCustomer(age, name, quantity, dishname, bill, address, deliveryCharges, distanceDelivery);
```

```
    if (myResturant->nextHomeDeliveryCustomer == NULL)
```

```
    {
```

```
        // if first insert in front
```

```
        myResturant->nextHomeDeliveryCustomer = currentHomeDeliveryCustomer;
```

```

    }
    else
    {
        // finding the last Node
        homeDeliveryCustomer *temp = myResturant->nextHomeDeliveryCustomer;
        while (temp->next != NULL)
        {
            temp = temp->next;
        }

        temp->next = currentHomeDeliveryCustomer;
        currentHomeDeliveryCustomer->next = NULL;
    }

    cout << "Your Order has been Placed MR/MRS " << name << " and your order is " <<
dishname << " with " << quantity << " quantity and total bill is " << bill << endl;
}

void serveOrderHomeDeliveryCustomer()
{
    if (myResturant->nextHomeDeliveryCustomer == NULL)
    {
        cout << "No Home Delivery Customer to Serve" << endl;
    }
    else
    {

```

```
// serving the last customer first

homeDeliveryCustomer *first = myResturant->nextHomeDeliveryCustomer;

if (first->next == NULL)
{
    // if it is the only customer

    myResturant->nextHomeDeliveryCustomer = NULL;

    cout << "Home Delivery Customer Served : " << first->cusotomer.name << endl;
    string customerType = "Home-Delivery Customer";

    root = insertion(first->cusotomer.age, first->cusotomer.name, first-
>cusotomer.quantity, first->cusotomer.dishname, first->cusotomer.bill, customerType,
root);

    // now deleting the node

    delete (first);
}
else {

    homeDeliveryCustomer *s = first->next;

    while(s->next !=NULL){

        first = first->next;

        s = s->next;

    }

    first->next = NULL;
```



```

        cout << "Home Delivery Customer Served : " << s->cusotomer.name << endl;

        string customerType = "Home-Delivery Customer";

        root = insertion(s->cusotomer.age, s->cusotomer.name, s->cusotomer.quantity, s-
>cusotomer.dishname, s->cusotomer.bill, customerType, root);

        // deleting the node

        delete (s);
    }

}

}

// It will serve all the waiting orders

void serveAllOrders()
{

    while (myResturant->nextTakeAwayCustomer != NULL)
    {
        serveOrderTakeAwayCustomer();
    }

    while (myResturant->nextDineInCustomer != NULL)
    {
        serveOrderDineInCustomer();
    }
}

```

```

    }
    while (myResturant->nextHomeDeliveryCustomer != NULL)
    {
        serveOrderHomeDeliveryCustomer();
    }
}

void displayAllOrdersTakeAwayCustomers()
{
    if (myResturant->nextTakeAwayCustomer == NULL)
    {
        cout << "There is no Order for Walking Customer till yet" << endl;
    }
    else
    {
        takeAwayCustomer *traversal = myResturant->nextTakeAwayCustomer;
        while (traversal != NULL)
        {

            cout << "-----" << endl;
            cout << "Take-Away Customer : " << traversal->cusotomer.name << endl;
            cout << "Age : " << traversal->cusotomer.age << endl;
            cout << "Dish Name : " << traversal->cusotomer.dishname << endl;
            cout << "Quantity : " << traversal->cusotomer.quantity << endl;
            cout << "Bill : " << traversal->cusotomer.bill << " RS/_ " << endl;
            cout << "-----" << endl;
        }
    }
}

```

```
        traversal = traversal->next;
    }
}
}
```

```
void displayAllOrdersHomeDeliveryCustomers()
```

```
{
    if (myResturant->nextHomeDeliveryCustomer == NULL)
    {
        cout << "There is no Order for Home Delivery Customer till yet" << endl;
    }
    else
    {
        homeDeliveryCustomer *traversal = myResturant->nextHomeDeliveryCustomer;
        while (traversal != NULL)
        {
            cout << "-----" << endl;
            cout << "Home Delivery Customer : " << traversal->cusotomer.name << endl;
            cout << "Age : " << traversal->cusotomer.age << endl;
            cout << "Dish Name : " << traversal->cusotomer.dishname << endl;
            cout << "Quantity : " << traversal->cusotomer.quantity << endl;
            cout << "Delivery Distance : " << traversal->deliveryCharges << "KM"<<endl;
            cout << "Delivery Charges : " << traversal->distanceDelivery << endl;
            cout << "Bill : " << traversal->cusotomer.bill << " RS/_ " << endl;
            cout << "Delivery Address : " << traversal->address << endl;
        }
    }
}
```

```

        cout << "-----" << endl;

        traversal = traversal->next;
    }
}

void displayAllOrdersDineInCustomers()
{
    if (myResturant->nextDineInCustomer == NULL)
    {
        cout << "There is no Order for Dine-In Customer till yet" << endl;
    }
    else
    {
        dineInCustomer *traversal = myResturant->nextDineInCustomer;
        while (traversal != NULL)
        {
            cout << "-----" << endl;
            cout << "Walking Customer : " << traversal->cusotomer.name << endl;
            cout << "Age : " << traversal->cusotomer.age << endl;
            cout << "Dish Name : " << traversal->cusotomer.dishname << endl;
            cout << "Quantity : " << traversal->cusotomer.quantity << endl;
            cout << "Bill : " << traversal->cusotomer.bill << " RS/_ " << endl;
            cout << "-----" << endl;

```

```
        traversal = traversal->next;
    }
}
}
```

```
void displayAllOrders()
```

```
{
```

```
    cout << "The Take-Away Customers Are :" << endl;
```

```
    displayAllOrdersTakeAwayCustomers();
```

```
    cout << "The Dine-IN Customers Are :" << endl;
```

```
    displayAllOrdersDineInCustomers();
```

```
    cout << "The Home Delivery Customers Are :" << endl;
```

```
    displayAllOrdersHomeDeliveryCustomers();
```

```
}
```

```
void totalbillofPendingOrders()
```

```
{
```

```
    takeAwayCustomer *p = myResturant->nextTakeAwayCustomer;
```

```
    while (p != NULL)
```

```
    {
```

```
        takeAway += p->cusotomer.bill;
```

```
        p = p->next;
```

```
    }
```

```

dineInCustomer *q = myResturant->nextDineInCustomer;
while (q != NULL)
{
    dineIn += q->cusotomer.bill;
    q = q->next;
}
homeDeliveryCustomer *r = myResturant->nextHomeDeliveryCustomer;
while (r != NULL)
{
    homeDelivery += r->cusotomer.bill;
    r = r->next;
}
total = takeAway + dineIn + homeDelivery;

cout << "The total bill of pending orders for Take-Away customers are : " << takeAway << "
RS/_ " << endl;

cout << "The total bill of pending orders for Dine-In customers are : " << dineIn << " RS/_ "
<< endl;

cout << "The total bill of pending orders for Delivery customers are : " << homeDelivery <<
" RS/_ " << endl;

cout << "The Total orders pending are : " << total << " RS/_ " << endl;
}

double calculateTotalServedEarnings(servedCustomer *root){

if(root){
    calculateTotalServedEarnings(root->left);

```

```

        servedTotal += root->bill;

        calculateTotalServedEarnings(root->right);
    }

    return servedTotal;
}

// making a graph for the available delivery options

//          0      1      2      3      4      5
string deliveryPoints[] = {"PizzaSHOP", "Chauburji", "Shadman", "Islampura", "JoharTown",
    "Anarkali"};

// first value in the pair is vertex and second is the distance (weight) in KM

vector<vector<pair<int, int>>> deliveryMap = {

    // first value in the pair is vertex and second is the distance (weight) in KM
    {{1, 2}, {2, 3}, {3, 5}, {5, 4}}, // 0 (Pizza Shop)
    {{0, 2}, {5, 1}},                // 1 (Chauburji)
    {{0, 3}, {3, 1}},                // 2 (Shadman)
    {{0, 5}, {4, 2}, {5, 2}, {2, 1}}, // 3 (Islampura)
    {{3, 2}, {5, 2}},                // 4 (Johar Town)
    {{0, 4}, {1, 1}, {3, 2}, {4, 2}} // 5 (Anarkali)

};

```

```

vector<int> dijkstra(int sourceNode)
{
    vector<int> distance(6, infinity);
    set<pair<int, int>> s;
    distance[sourceNode] = 0; //
    s.insert(make_pair(0, sourceNode));

    while (!s.empty())
    {
        auto top = *(s.begin());
        int u = top.first; // current weight
        int v = top.second; // current vertex

        s.erase(s.begin());

        // traversing the adjacency list of v
        for (auto child : deliveryMap[v])
        {
            int childVertex = child.first;
            int childWeight = child.second;

            if (u + childWeight < distance[childVertex])
            {
                distance[childVertex] = u + childWeight;
                s.insert(make_pair(distance[childVertex], childVertex));
            }
        }
    }
}

```



```
    }  
    }  
}  
  
    return distance;  
}  
  
int main() {  
    // making shop  
    myResturant = new Resturant;  
  
    myResturant->shopName = " Quetta Cafe";  
  
    myResturant->address = "Liberty Chowk, Lahore";  
  
    // Setting the menu  
    myResturant->menu = new string[11]{  
        "chickenBiryani", "beefKebab",  
        "muttonKarahi", "butterChicken",  
        "dalMakhani", "paneerTikka",  
        "fishCurry", "vegPulao",  
        "grilledSandwich", "chowMein"};  
  
    // setting the price  
    myResturant->price = new int[11]{0, 2000, 2500, 2400, 2200, 2700, 2000, 2100, 3000,  
3000, 2800};
```

```

int option = -99;

do {
    cout << "\n===== Welcome to " << myResturant->shopName << "
===== \n";

    cout << "Located at " << myResturant->address << "\n";

    cout << "-----\n";

    cout << "Main Menu:\n";

    cout << "1. View Menu\n";

    cout << "2. Place Order\n";

    cout << "3. Serve Order\n";

    cout << "4. Display Pending Orders\n";

    cout << "5. Display Served Orders\n";

    cout << "6. Billing Information\n";

    cout << "0. Exit\n";

    cout << "Enter your choice: ";

    cin >> option;

    if (option == 1) {
        cout << "\n===== MENU =====\n";

        for (int i = 1; i <= 10; i++) {
            cout << i << ". " << myResturant->menu[i] << " - " << myResturant->price[i] << " RS/_ "
<< endl;

            }

        }
    }
}

```

```
else if (option == 2) {  
    int subOption;  
    cout << "\n== Place Order Menu ==\n";  
    cout << "1. Take-Away Customer\n";  
    cout << "2. Home Delivery Customer\n";  
    cout << "3. Dine-In Customer\n";  
    cout << "Enter your choice: ";  
    cin >> subOption;  
  
    int age, quantity, pizzaIndex;  
    string name, address;  
    double bill;  
  
    switch (subOption) {  
        case 1:  
            cout << "Enter the name of the customer: ";  
            cin >> name;  
            cout << "Enter the age of the customer: ";  
            cin >> age;  
            cout << "Enter the quantity of the dish: ";  
            cin >> quantity;  
            cout << "Enter the option for the dish : ";  
            cin >> pizzaIndex;  
  
            if (pizzaIndex >= 1 && pizzaIndex <= 10) {  
                bill = quantity * myResturant->price[pizzaIndex];  
            }  
        }  
    }
```

```
        placeOrderTakeAwayCustomer(age, name, myResturant->menu[pizzaIndex],  
quantity, bill);
```

```
    } else {
```

```
        cout << "Invalid pizza option." << endl;
```

```
    }
```

```
    break;
```

```
case 2: {
```

```
    vector<int> distanceFromShop = dijkstra(0);
```

```
    int optionDelivery = -999;
```

```
    do {
```

```
        cout << "The delivery is available for following Areas : " << endl;
```

```
        for (int i = 1; i <= 5; i++) {
```

```
            cout << i << ". " << deliveryPoints[i] << " (Distance: " << distanceFromShop[i]  
<< " KM)" << endl;
```

```
        }
```

```
        cout << "Enter your option (0 to 5): ";
```

```
        cin >> optionDelivery;
```

```
        if (!(optionDelivery >= 0 && optionDelivery <= 5)) {
```

```
            cout << "Invalid delivery option. Please choose between 0 and 5." << endl;
```

```
        }
```

```
    } while (!(optionDelivery >= 0 && optionDelivery <= 5));
```

```
    address = deliveryPoints[optionDelivery];
```

```
    cout << "Enter the name of the customer: ";
```

```

    cin >> name;

    cout << "Enter the age of the customer: ";

    cin >> age;

    cout << "Enter the quantity of the pizza: ";

    cin >> quantity;

    cout << "Enter the option for the pizza: ";

    cin >> pizzaIndex;

    if (pizzaIndex >= 1 && pizzaIndex <= 10) {

        int deliveryChargesPerKM = 50;

        int deliveryCharges = deliveryChargesPerKM *
distanceFromShop[optionDelivery];

        bill = (quantity * myResturant->price[pizzaIndex]) + deliveryCharges;

        int distanceFromTheShop = distanceFromShop[optionDelivery];

        placeOrderHomeDeliveryCustomer(age, name, myResturant-
>menu[pizzaIndex], quantity, bill, address, deliveryCharges, distanceFromTheShop);

    } else {

        cout << "Invalid pizza option." << endl;

    }

} break;

case 3:

    cout << "Enter the name of the customer: ";

    cin >> name;

    cout << "Enter the age of the customer: ";

    cin >> age;

```

```
    cout << "Enter the quantity of the pizza: ";
    cin >> quantity;
    cout << "Enter the option for the pizza: ";
    cin >> pizzaIndex;

    if (pizzaIndex >= 1 && pizzaIndex <= 10) {
        bill = quantity * myResturant->price[pizzaIndex];
        placeOrderDineInCustomer(age, name, myResturant->menu[pizzaIndex],
quantity, bill);
    } else {
        cout << "Invalid pizza option." << endl;
    }
    break;

default:
    cout << "Invalid option.\n";
}
}

else if (option == 3) {
    int subOption;
    cout << "\n== Serve Order Menu ==\n";
    cout << "1. Take-Away\n2. Home Delivery\n3. Dine-In\n4. Serve All Orders\nEnter
your choice: ";
    cin >> subOption;

    switch (subOption) {
```

```
        case 1: serveOrderTakeAwayCustomer(); break;
        case 2: serveOrderHomeDeliveryCustomer(); break;
        case 3: serveOrderDineInCustomer(); break;
        case 4: serveAllOrders(); break;
        default: cout << "Invalid option.\n";
    }
}
```

```
else if (option == 4) {
    int subOption;

    cout << "\n== Display Pending Orders ==\n";

    cout << "1. Take-Away\n2. Home Delivery\n3. Dine-In\n4. All Pending Orders\nEnter
your choice: ";

    cin >> subOption;
```

```
    switch (subOption) {
        case 1: displayAllOrdersTakeAwayCustomers(); break;
        case 2: displayAllOrdersHomeDeliveryCustomers(); break;
        case 3: displayAllOrdersDineInCustomers(); break;
        case 4: displayAllOrders(); break;
        default: cout << "Invalid option.\n";
    }
}
```

```
else if (option == 5) {
    int subOption;
```

```
cout << "\n== Served Orders Menu ==\n";
cout << "1. Display All Served Orders\n";
cout << "2. Search Served Order by Name\n";
cout << "3. Clear Served Orders List\n";
cin >> subOption;

if (subOption == 1) {
    if (isEmpty(root)) {
        cout << "No Served Customer yet.\n";
    } else {
        displayAllServedOrders(root);
    }
} else if (subOption == 2) {
    string name;
    cout << "Enter the name of the customer to search: ";
    cin >> name;
    servedCustomer *searchedCustomer = search(root, name);
    if (searchedCustomer == NULL)
        cout << "No such Customer was Served.\n";
    else
        display(searchedCustomer);
} else if (subOption == 3) {
    deleteAllServedCustomers(root);
    cout << "Served Orders List Cleared.\n";
} else {
    cout << "Invalid option.\n";
}
```



```

    }
}

else if (option == 6) {
    int subOption;

    cout << "\n== Billing Information ==\n";

    cout << "1. Display total bill of Pending Orders\n";
    cout << "2. Display total Earnings from Served Orders\n";
    cin >> subOption;

    if (subOption == 1) {
        totalbillofPendingOrders();
    } else if (subOption == 2) {
        double total = calculateTotalServedEarnings(root);
        cout << "Total Earnings: " << total << " RS/_ " << endl;
    } else {
        cout << "Invalid option.\n";
    }
}

else if (option == 0) {
    cout << "\n Restaurant Management System -- Terminated\n";
    cout << "Thank you for Using our Services\n";
}

else {

```

```
        cout << "Invalid main option. Please try again.\n";
    }

    cout << "\nPress Enter to continue...";
    cin.ignore(numeric_limits<streamsize>::max(), '\n');
    cin.get();

} while (option != 0);

// Cleanup
delete[] myResturant->menu;
delete[] myResturant->price;
delete myResturant;
myResturant = NULL;

serveAllOrders();
deleteAllServedCustomers(root);

return 0;
}
```

OUTPUT SCREENS:-

```
===== Welcome to Quetta Cafe =====  
Located at Liberty Chowk, Lahore  
-----
```

```
Main Menu:  
1. View Menu  
2. Place Order  
3. Serve Order  
4. Display Pending Orders  
5. Display Served Orders  
6. Billing Information  
0. Exit
```

Enter your choice: 1

```
===== MENU =====
```

```
1. chickenBiryani - 2000 RS/_  
2. beefKebab - 2500 RS/_  
3. muttonKarahi - 2400 RS/_  
4. butterChicken - 2200 RS/_  
5. dalMakhani - 2700 RS/_  
6. paneerTikka - 2000 RS/_  
7. fishCurry - 2100 RS/_  
8. vegPulao - 3000 RS/_  
9. grilledSandwich - 3000 RS/_  
10. chowMein - 2800 RS/_
```

Press Enter to continue...

```
===== Welcome to Quetta Cafe =====  
Located at Liberty Chowk, Lahore  
-----
```

```
Main Menu:  
1. View Menu
```

```
2. Place Order  
3. Serve Order  
4. Display Pending Orders  
5. Display Served Orders  
6. Billing Information  
0. Exit
```

Enter your choice: 2

```
== Place Order Menu ==
```

```
1. Take-Away Customer  
2. Home Delivery Customer  
3. Dine-In Customer
```

Enter your choice: 1

Enter the name of the customer: is

Enter the age of the customer: 5

Enter the quantity of the dish: 2

Enter the option for the dish : 9

Your Order has been Placed MR/MRS is and your order is grilledSandwich with 2 quantity and total bill is 6000

Press Enter to continue...

```
===== Welcome to Quetta Cafe =====  
Located at Liberty Chowk, Lahore  
-----
```

```
Main Menu:  
1. View Menu  
2. Place Order  
3. Serve Order  
4. Display Pending Orders  
5. Display Served Orders  
6. Billing Information  
0. Exit
```

Enter your choice: 4

```

== Display Pending Orders ==
1. Take-Away
2. Home Delivery
3. Dine-In
4. All Pending Orders
Enter your choice: 4
The Take-Away Customers Are :
-----
Take-Away Customer : is
Age : 5
Dish Name : grilledSandwich
Quantity : 2
Bill : 6000 RS/_
-----
The Dine-IN Customers Are :
There is no Order for Dine-In Customer till yet
The Home Delivery Customers Are :
There is no Order for Home Delivery Customer till yet

Press Enter to continue...

===== Welcome to Quetta Cafe =====
Located at Liberty Chowk, Lahore
-----
Main Menu:
1. View Menu
2. Place Order
3. Serve Order
4. Display Pending Orders
5. Display Served Orders
6. Billing Information
0. Exit
Enter your choice: 3

```

```

== Serve Order Menu ==
1. Take-Away
2. Home Delivery
3. Dine-In
4. Serve All Orders
Enter your choice: 4
Take Away Customer Served : is

Press Enter to continue...

===== Welcome to Quetta Cafe =====
Located at Liberty Chowk, Lahore
-----
Main Menu:
1. View Menu
2. Place Order
3. Serve Order
4. Display Pending Orders
5. Display Served Orders
6. Billing Information
0. Exit
Enter your choice: 5

== Served Orders Menu ==
1. Display All Served Orders
2. Search Served Order by Name
3. Clear Served Orders List
1
Name :is
Age :5
Pizza :grilledSandwich
Quantity : 2
Bill : 6000

```

```
5. Display Served Orders
6. Billing Information
0. Exit
```

Enter your choice: 0

Restaurant Management System -- Terminated

Thank you for Using our Services

== Billing Information ==

```
1. Display total bill of Pending Orders
2. Display total Earnings from Served Orders
2
```

Total Earnings: 6000 RS/_

Press Enter to continue...

===== Welcome to Quetta Cafe =====

Located at Liberty Chowk, Lahore

Main Menu:

```
1. View Menu
2. Place Order
3. Serve Order
4. Display Pending Orders
5. Display Served Orders
6. Billing Information
0. Exit
```

Enter your choice: 0

Restaurant Management System -- Terminated

Thank you for Using our Services

Total Earnings: 6000 RS/_

Press Enter to continue...

```
===== Welcome to Quetta Cafe =====  
Located at Liberty Chowk, Lahore
```

```
-----
```

```
Main Menu:
```

1. View Menu
2. Place Order
3. Serve Order
4. Display Pending Orders
5. Display Served Orders
6. Billing Information
0. Exit

```
Enter your choice: 0
```

```
Restaurant Management System -- Terminated
```

```
Thank you for Using our Services
```

```
Main Menu:
```

1. View Menu
2. Place Order
3. Serve Order
4. Display Pending Orders
5. Display Served Orders
6. Billing Information
0. Exit

```
Enter your choice: 0
```

```
Restaurant Management System -- Terminated
```

```
Thank you for Using our Services
```

```
Enter your choice: 0
```

```
Restaurant Management System -- Terminated
```

```
Thank you for Using our Services
```

```
Restaurant Management System -- Terminated
```

```
Thank you for Using our Services
```