

Data Structures Project

CSE102

Submitted to-

Dr. ASHOK KUMAR PRADHAN

OUR TEAM-

|  |  |  |
| --- | --- | --- |
| SNO | NAME | REG NO |
| 1 | M. Pardha saradhi | AP23110010056 |
| 2. | L. Prudhvi Sai | AP23110010076 |
| 3. | J. Rohith | AP23110010060 |
| 4. | M.RAM | AP23110010015 |
| 5. | C .HV. Arjun | AP23110010043 |
| 6. | M. Sai Surya | AP23110010034 |

Algorithm-

1. 1. Initialization:

* Set up order queues and menu items.
* set 1 in the orderCounter.

1. Main Menu Loop:

* Display options for placing orders, preparing orders, serving orders, or exiting the program.
* Prompt the user to choose an option.

1. Place Order Functionality:

* Display the menu with item numbers and prices.
* Prompt the user to input the number of items in the order and the details for each item.
* Validate inputs and create a new order.
* Enqueue the order into the main order queue.

Increment orderCounter.

1. Prepare Order Functionality:

* Dequeue an order from the main order queue.
* Validate if there are orders to prepare.
* Enqueue the order into the kitchen queue for preparation.

1. Serve Order Functionality:

* Dequeue an order from the kitchen queue.
* Validate if there are orders to serve.
* Display the served order details.

1. Display Menu Functionality:

* Display the menu with item numbers and prices.

1. Utility Functions:

* createNode(): Create a new node for the queue.
* createQueue(): Create an empty queue.
* isEmpty(): Check if the queue is empty.
* enqueue(): Add an order to the end of the queue.
* dequeue(): Remove and return the first order from the queue.
* calculateOrderTotal(): Calculate the total cost of an order based on menu prices.

1. Exit Program:

* Terminate the program when the user chooses the exit option.

Code-

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#define MAX\_ITEMS 100

#define MAX\_NAME\_LENGTH 50

#define MAX\_ORDER\_ITEMS 10 // Maximum number of items in an order

struct MenuItem {

char name[MAX\_NAME\_LENGTH];

float price;

};

struct OrderItem {

char itemName[MAX\_NAME\_LENGTH];

int quantity;

};

// Structure for an order

struct Order {

int orderNumber; // Order number

struct tm orderTime; // Order time

struct OrderItem items[MAX\_ORDER\_ITEMS];

int numItems;

};

// Structure for a queue node

struct Node {

struct Order data;

struct Node\* next;

};

// Structure for a queue

struct Queue {

struct Node \*front, \*rear;

};

int orderCounter = 1; // Global variable to track order numbers

// Function to create a new queue node

struct Node\* createNode(struct Order item) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = item;

newNode->next = NULL;

return newNode;

}

// Function to create an empty queue

struct Queue\* createQueue() {

struct Queue\* queue = (struct Queue\*)malloc(sizeof(struct Queue));

queue->front = queue->rear = NULL;

return queue;

}

// Function to check if queue is empty

int isEmpty(struct Queue\* queue) {

return queue->front == NULL;

}

// Function to enqueue an order

void enqueue(struct Queue\* queue, struct Order item) {

struct Node\* newNode = createNode(item);

if (isEmpty(queue)) {

queue->front = queue->rear = newNode;

} else {

queue->rear->next = newNode;

queue->rear = newNode;

}

}

// Function to dequeue an order

struct Order dequeue(struct Queue\* queue) {

if (isEmpty(queue)) {

struct Order emptyOrder;

emptyOrder.orderNumber = -1; // Invalid order number

return emptyOrder;

}

struct Node\* temp = queue->front;

struct Order item = temp->data;

queue->front = queue->front->next;

free(temp);

return item;

}

// Function to display menu

void displayMenu(struct MenuItem menu[], int n) {

printf("Menu:\n");

for (int i = 0; i < n; i++) {

printf("%d. %s - Rs%.2f\n", i + 1, menu[i].name, menu[i].price);

}

}

// Function to place an order

void placeOrder(struct MenuItem menu[], int n, struct Queue\* orderQueue) {

printf("Menu:\n");

for (int i = 0; i < n; i++) {

printf("%d. %s - Rs%.2f\n", i + 1, menu[i].name, menu[i].price);

}

struct Order newOrder;

newOrder.orderNumber = orderCounter++; // Assign order number and increment for next order

newOrder.numItems = 0;

printf("Enter number of items in order: ");

int numItemsInOrder;

scanf("%d", &numItemsInOrder);

if (numItemsInOrder < 1 || numItemsInOrder > MAX\_ORDER\_ITEMS) {

printf("Invalid number of items!\n");

return;

}

// Get current time

time\_t currentTime;

time(&currentTime);

newOrder.orderTime = \*localtime(&currentTime);

for (int i = 0; i < numItemsInOrder; i++) {

printf("Enter item number and quantity for item %d: ", i + 1);

int itemNum, quantity;

scanf("%d %d", &itemNum, &quantity);

if (itemNum < 1 || itemNum > n) {

printf("Invalid item number!\n");

return;

}

struct OrderItem newOrderItem;

strcpy(newOrderItem.itemName, menu[itemNum - 1].name);

newOrderItem.quantity = quantity;

newOrder.items[i] = newOrderItem;

newOrder.numItems++;

}

enqueue(orderQueue, newOrder);

printf("Order placed successfully! Order Number: %d\n", newOrder.orderNumber);

}

// Function to prepare an order

void prepareOrder(struct Queue\* orderQueue, struct Queue\* kitchenQueue) {

struct Order preparedOrder = dequeue(orderQueue);

if (preparedOrder.orderNumber == -1) {

printf("No orders to prepare!\n");

return;

}

enqueue(kitchenQueue, preparedOrder);

printf("Order with Order Number %d prepared and sent to kitchen!\n", preparedOrder.orderNumber);

}

// Function to serve an order

void serveOrder(struct Queue\* kitchenQueue) {

struct Order servedOrder = dequeue(kitchenQueue);

if (servedOrder.orderNumber == -1) {

printf("No orders to serve!\n");

return;

}

printf("Order Number %d served:\n", servedOrder.orderNumber);

for (int i = 0; i < servedOrder.numItems; i++) {

printf("Item: %s - Quantity: %d\n", servedOrder.items[i].itemName, servedOrder.items[i].quantity);

}

printf("Order Time: %d-%02d-%02d %02d:%02d:%02d\n",

servedOrder.orderTime.tm\_year + 1900, servedOrder.orderTime.tm\_mon + 1, servedOrder.orderTime.tm\_mday,

servedOrder.orderTime.tm\_hour, servedOrder.orderTime.tm\_min, servedOrder.orderTime.tm\_sec);

}

// Function to calculate the total cost of an order

float calculateOrderTotal(struct Order order, struct MenuItem menu[]) {

float total = 0.0;

for (int i = 0; i < order.numItems; i++) {

for (int j = 0; j < MAX\_ITEMS; j++) {

if (strcmp(order.items[i].itemName, menu[j].name) == 0) {

total += menu[j].price \* order.items[i].quantity;

break;

}

}

}

return total;

}

// Function to print receipt for an order including the bill

void printReceipt(struct Order order, struct MenuItem menu[]) {

printf("Receipt for Order Number %d:\n", order.orderNumber);

for (int i = 0; i < order.numItems; i++) {

printf("Item: %s - Quantity: %d\n", order.items[i].itemName, order.items[i].quantity);

}

printf("Order Time: %d-%02d-%02d %02d:%02d:%02d\n",

order.orderTime.tm\_year + 1900, order.orderTime.tm\_mon + 1, order.orderTime.tm\_mday,

order.orderTime.tm\_hour, order.orderTime.tm\_min, order.orderTime.tm\_sec);

printf("Total Cost: Rs%.2f\n", calculateOrderTotal(order, menu));

}

// Function to input menu items

void inputMenu(struct MenuItem menu[], int\* numItems) {

printf("Enter the number of menu items: ");

scanf("%d", numItems);

printf("Enter menu items:\n");

for (int i = 0; i < \*numItems; i++) {

printf("Item %d:\n", i + 1);

printf("Name: ");

scanf(" %[^\n]s", menu[i].name);

printf("Price: Rs");

scanf("%f", &menu[i].price);

}

}

int main() {

struct MenuItem menu[MAX\_ITEMS]; // Array to store menu items

int numItems = 0; // Number of items in the menu, initially 0

// Create queues

struct Queue\* orderQueue = createQueue(); // For customer orders

struct Queue\* kitchenQueue = createQueue(); // For kitchen tasks

int choice;

do {

printf("\nRestaurant Order Management System\n");

printf("1. Display Menu\n");

printf("2. Input Menu\n");

printf("3. Place Order\n");

printf("4. Prepare Order\n");

printf("5. Serve Order\n");

printf("6. Print Receipt\n");

printf("7. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

displayMenu(menu, numItems);

break;

case 2:

inputMenu(menu, &numItems);

break;

case 3:

placeOrder(menu, numItems, orderQueue);

break;

case 4:

prepareOrder(orderQueue, kitchenQueue);

break;

case 5:

serveOrder(kitchenQueue);

break;

case 6: {

printf("Enter the Order Number to print receipt: ");

int orderNumber;

scanf("%d", &orderNumber);

// Find the order and print its receipt

struct Node\* temp = orderQueue->front;

while (temp != NULL) {

if (temp->data.orderNumber == orderNumber) {

printReceipt(temp->data, menu);

break;

}

temp = temp->next;

}

// If order not found in orderQueue, search in kitchenQueue

if (temp == NULL) {

temp = kitchenQueue->front;

while (temp != NULL) {

if (temp->data.orderNumber == orderNumber) {

printReceipt(temp->data, menu);

break;

}

temp = temp->next;

}

}

// If order not found in either queue, print message

if (temp == NULL) {

printf("Order with Order Number %d not found!\n", orderNumber);

}

break;

}

case 7:

printf("Exiting...\n");

break;

default:

printf("Invalid choice. Please enter a valid option.\n");

}

} while (choice != 7);

// Free memory before exit

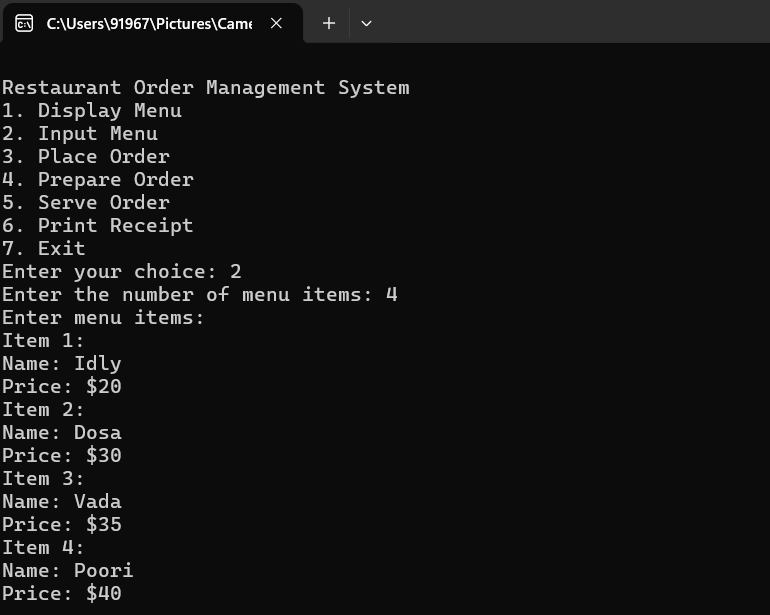
free(orderQueue);

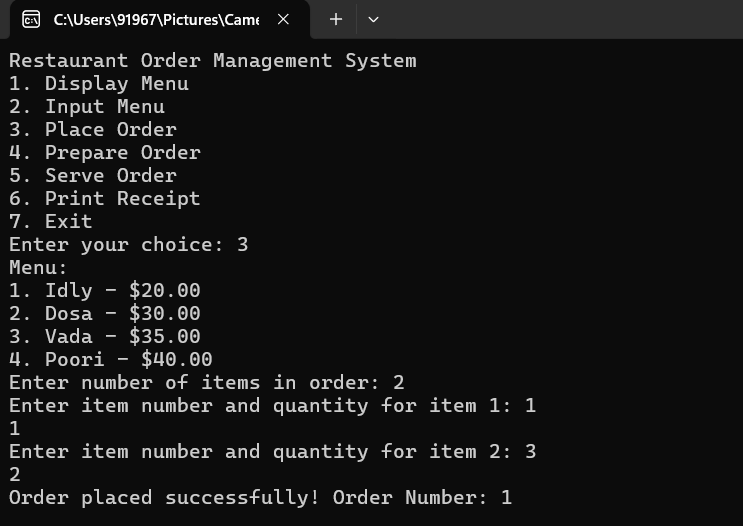
free(kitchenQueue);

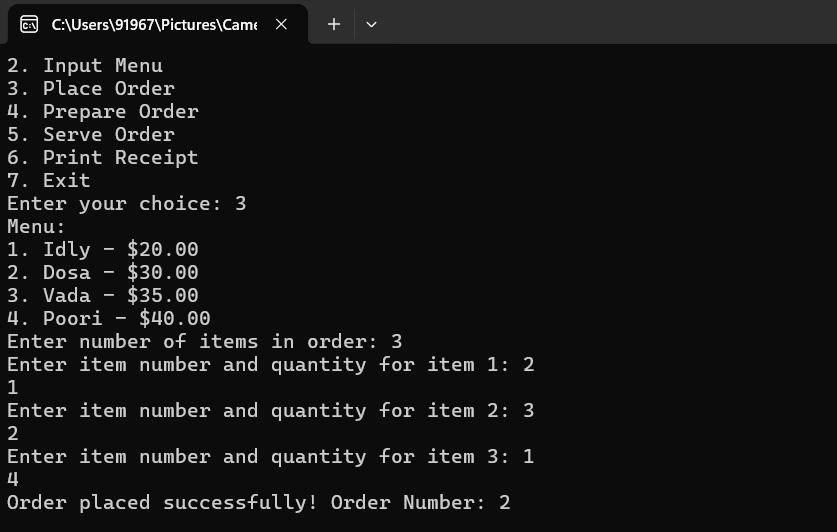
return 0;

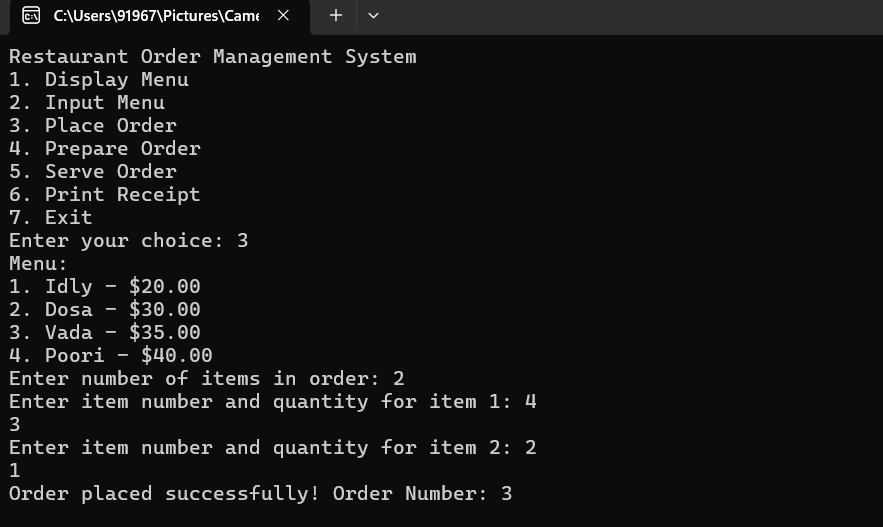
}

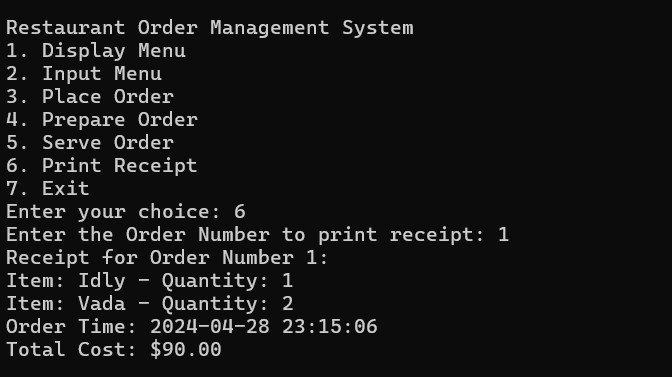
Output-

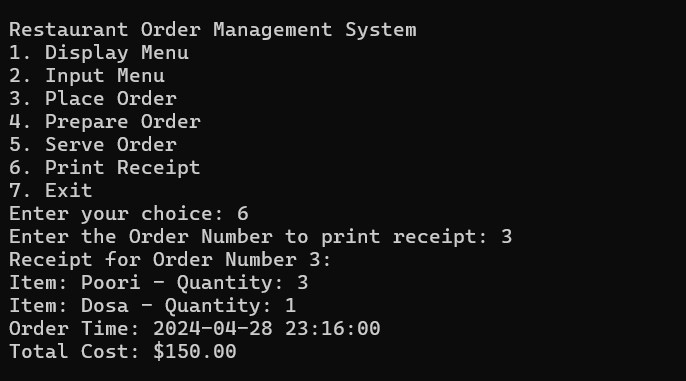


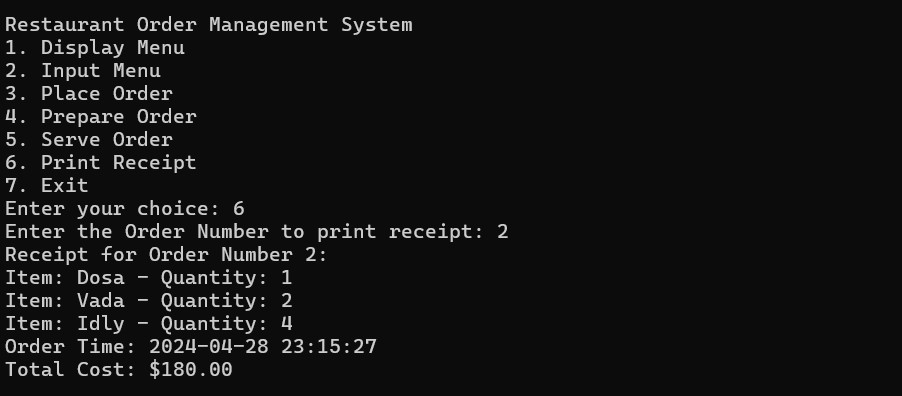


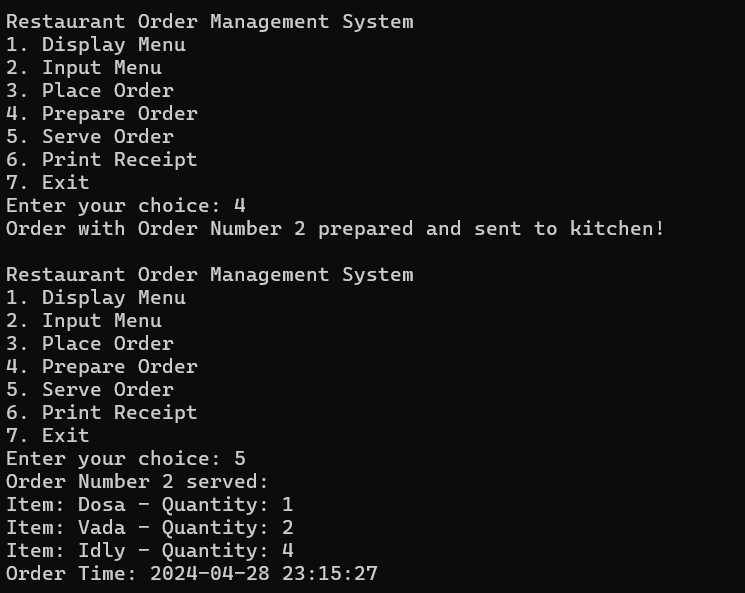


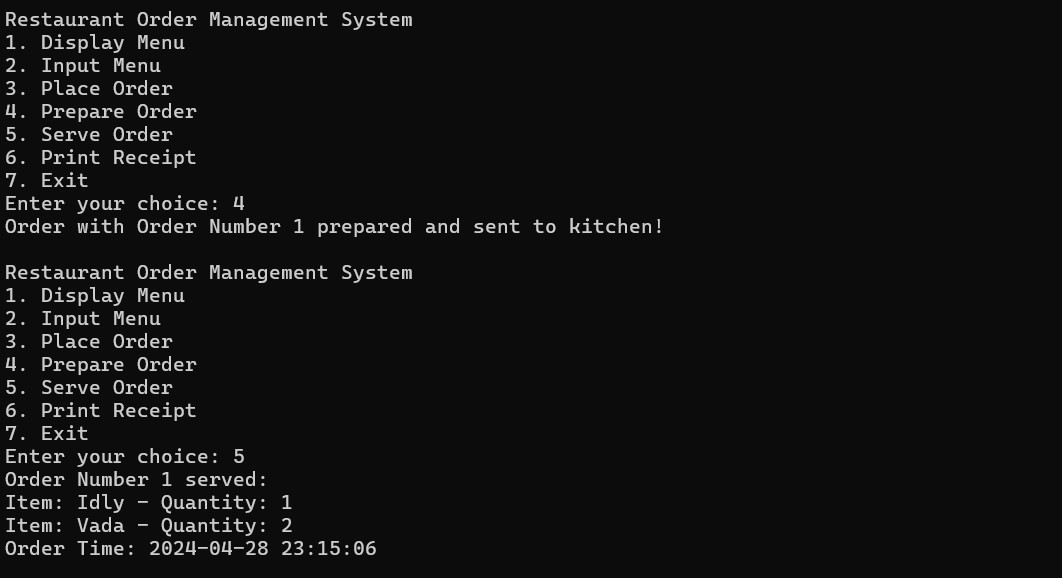


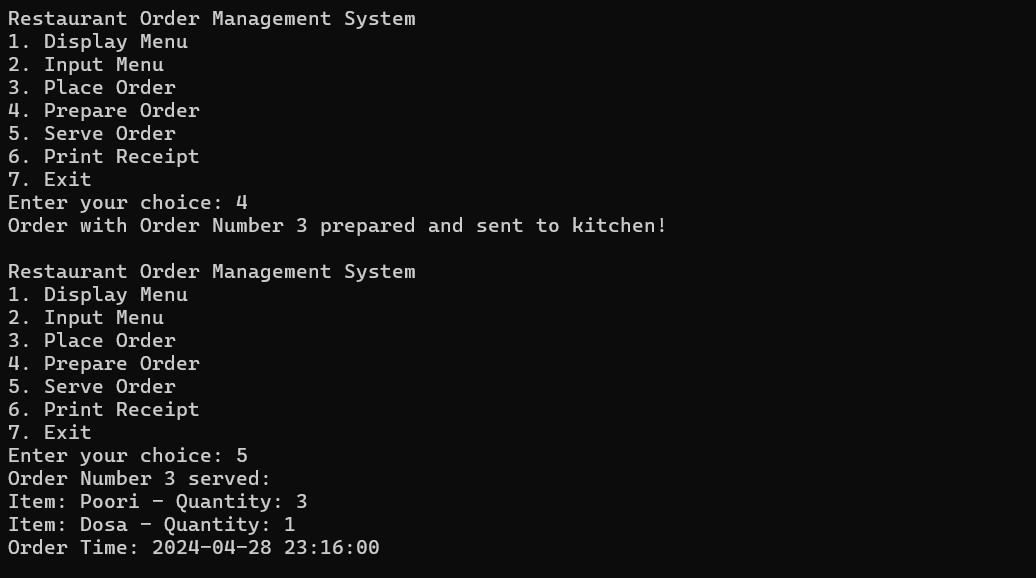


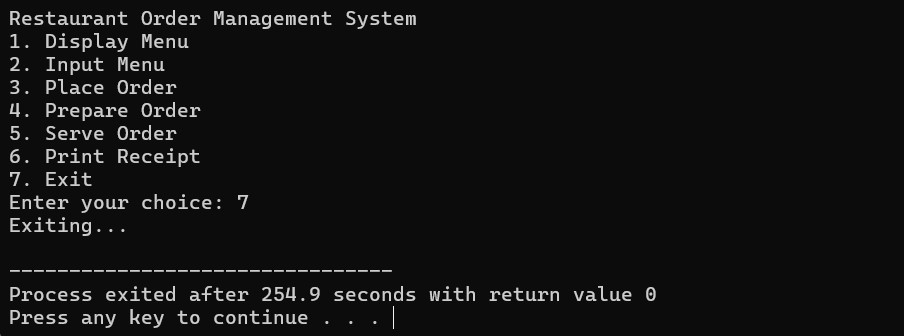












Project Video Link-

[https://drive.google.com/file/d/+12YIfq4XWfLUVPB\_58s0JqrNXmKjjFWNx/view?usp=drive\_link](https://drive.google.com/file/d/12YIfq4XWfLUVPB_58s0JqrNXmKjjFWNx/view?usp=drive_link)

…..……………..THE END…………………