**Restaurant Management System**

**Others part & Overall All Code**

**ID:232-35-100**

**Final Code:**

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#include <string.h>

#define MAX\_MENU\_ITEMS 100

#define MENU\_FILE "menu.txt"

#define ORDER\_FILE "orders.txt"

#define USERS\_FILE "users.txt"

// ANSI color codes

#define COLOR\_RESET "\033[0m"

#define COLOR\_RED "\033[31m"

#define COLOR\_GREEN "\033[32m"

#define COLOR\_YELLOW "\033[33m"

#define COLOR\_BLUE "\033[34m"

#define COLOR\_MAGENTA "\033[35m"

#define COLOR\_CYAN "\033[36m"

#define COLOR\_WHITE "\033[37m"

// Structure to represent a menu item

typedef struct {

int id;

char name[100];

float price;

} MenuItem;

typedef struct {

int orderId;

char customerName[100];

char itemName[100];

int quantity;

float itemPrice;

float totalCost;

char paymentMethod[20];

} Order;

// Global variable to keep track of next available item ID

int nextItemId = 1;

char loggedInUser[50];

// Function prototypes

void displayMainMenu();

void login();

void registerUser();

bool authenticate(const char username[], const char password[]);

void hidePassword(char \*password);

void displayAdminMenu();

void displayCustomerMenu();

void displayStaffMenu();

void addMenuItem();

void removeMenuItem();

void updateMenuItem();

void displayMenu();

void placeOrder();

void viewOrderHistory();

void generateInvoice(Order orders[], int orderCount);

void choosePaymentMethod(char paymentMethod[], float grandTotal);

void terminateProgram();

void printCentered(const char \*text, const char \*color);

void printBordered(const char \*text, const char \*color, int width);

int main() {

FILE \*menuFile = fopen(MENU\_FILE, "r");

if (menuFile != NULL) {

MenuItem item;

char line[150];

while (fgets(line, sizeof(line), menuFile)) {

sscanf(line, "%d|%99[^|]|%f", &item.id, item.name, &item.price);

if (item.id >= nextItemId) {

nextItemId = item.id + 1; // Set nextItemId to the next available ID

}

}

fclose(menuFile);

}

displayMainMenu();

return 0;

}

// Function to print centered text with color

void printCentered(const char \*text, const char \*color) {

int width = 80; // Assuming console width is 80 characters

int len = strlen(text);

int mid = (width - len) / 2;

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

printf(" ");

}

printf("%s%s%s\n", color, text, COLOR\_RESET);

}

// Function to print bordered and centered text with color

void printBordered(const char \*text, const char \*color, int width) {

int len = strlen(text);

int mid = (width - len - 2) / 2; // -2 for border space

printf("%s", color);

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

printf("=");

}

printf("\n|");

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

printf(" ");

}

printf("%s", text);

for (int i = 0; i < width - len - mid - 2; i++) {

printf(" ");

}

printf("|\n");

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

printf("=");

}

printf("%s\n", COLOR\_RESET);

}

// Function to display the main menu

void displayMainMenu() {

int choice;

do {

printf("\n");

printBordered("Welcome to Restaurant Management System", COLOR\_GREEN, 80);

printf("\n");

printCentered("1.Login", COLOR\_CYAN);

printCentered("2.Register", COLOR\_CYAN);

printCentered("3.Exit", COLOR\_CYAN);

printf("\n");

printCentered("Enter your choice: ", COLOR\_WHITE);

scanf("%d", &choice);

clearScreen();

switch (choice) {

case 1:

login();

break;

case 2:

registerUser();

break;

case 3:

terminateProgram();

break;

default:

printCentered("Invalid choice. Please try again.", COLOR\_RED);

break;

}

} while (1);

}

void clearScreen()

{

system("cls");

}

void hidePassword(char \*password) {

int i = 0;

char ch;

while ((ch = getch()) != '\r' && i < 49) { // Enter key is '\r' in Windows

if (ch == '\b' && i > 0) { // Handle backspace

printf("\b \b");

i--;

} else if (ch != '\b') {

printf("\*");

password[i++] = ch;

}

}

password[i] = '\0';

printf("\n");

}

void login() {

char username[50], password[50];

clearScreen();

printf("\nEnter username: ");

scanf("%s", username);

printf("Enter password: ");

hidePassword(password);

if (authenticate(username, password)) {

strcpy(loggedInUser, username); // Store logged-in username

printf("\nLogin successful! Welcome, %s.\n", loggedInUser);

// Determine user role and display respective menu

char role[20];

FILE \*userFile = fopen(USERS\_FILE, "r");

while (fscanf(userFile, "%s %s %s", username, password, role) == 3) {

if (strcmp(username, loggedInUser) == 0) {

fclose(userFile);

if (strcmp(role, "Admin") == 0) {

displayAdminMenu();

} else if (strcmp(role, "Customer") == 0) {

displayCustomerMenu();

} else if (strcmp(role, "Staff") == 0) {

displayStaffMenu();

}

return;

}

}

fclose(userFile);

} else {

printf("\nInvalid username or password!\n");

}

}

// Function to register a new user

void registerUser() {

char username[50], password[50], role[20];

clearScreen();

printf("\nEnter username: ");

scanf("%s", username);

printf("Enter password: ");

hidePassword(password);

printf("Enter role (Admin/Customer/Staff): ");

scanf("%s", role);

FILE \*userFile = fopen(USERS\_FILE, "a");

if (userFile == NULL) {

printf("Error opening users file.\n");

return;

}

fprintf(userFile, "%s %s %s\n", username, password, role);

fclose(userFile);

printf("\nRegistration successful!\n");

}

// Function to authenticate user credentials

bool authenticate(const char username[], const char password[]) {

char storedUsername[50], storedPassword[50], role[20];

FILE \*userFile = fopen(USERS\_FILE, "r");

if (userFile == NULL) {

printf("Error opening users file.\n");

return false;

}

while (fscanf(userFile, "%s %s %s", storedUsername, storedPassword, role) == 3) {

if (strcmp(storedUsername, username) == 0 && strcmp(storedPassword, password) == 0) {

fclose(userFile);

return true;

}

}

fclose(userFile);

return false;

}

// Function to display the admin menu

void displayAdminMenu() {

int choice;

do {

printf("\n");

printBordered("Admin Menu", COLOR\_BLUE, 80);

printf("\n");

printCentered("1. Add Menu Item", COLOR\_CYAN);

printCentered("2. Remove Menu Item", COLOR\_CYAN);

printCentered("3. Update Menu Item", COLOR\_CYAN);

printCentered("4. Display Menu", COLOR\_CYAN);

printCentered("0. Logout", COLOR\_CYAN);

printf("\n");

printCentered("Enter your choice: ", COLOR\_WHITE);

scanf("%d", &choice);

clearScreen();

switch (choice) {

case 1:

addMenuItem();

break;

case 2:

removeMenuItem();

break;

case 3:

updateMenuItem();

break;

case 4:

displayMenu();

break;

case 0:

strcpy(loggedInUser, ""); // Clear logged-in username

displayMainMenu();

return;

default:

printCentered("Invalid choice. Please try again.", COLOR\_RED);

break;

}

} while (1);

}

// Function to display the customer menu

void displayCustomerMenu() {

int choice;

do {

printf("\n");

printBordered("Customer Menu", COLOR\_BLUE, 80);

printf("\n");

printCentered("1. Place Order", COLOR\_CYAN);

printCentered("2. View Order History", COLOR\_CYAN);

printCentered("3. Display Menu", COLOR\_CYAN);

printCentered("0. Logout", COLOR\_CYAN);

printf("\n");

printCentered("Enter your choice: ", COLOR\_WHITE);

scanf("%d", &choice);

clearScreen();

switch (choice) {

case 1:

placeOrder();

break;

case 2:

viewOrderHistory();

break;

case 3:

displayMenu();

break;

case 0:

strcpy(loggedInUser, ""); // Clear logged-in username

displayMainMenu();

return;

default:

printCentered("Invalid choice. Please try again.", COLOR\_RED);

break;

}

} while (1);

}

void displayStaffMenu() {

int choice;

do {

printf("\n");

printBordered("Staff Menu", COLOR\_BLUE, 80);

printf("\n");

printCentered("1. View Order History", COLOR\_CYAN);

printCentered("2. Display Menu", COLOR\_CYAN);

printCentered("0. Logout", COLOR\_CYAN);

printf("\n");

printCentered("Enter your choice: ", COLOR\_WHITE);

scanf("%d", &choice);

clearScreen();

switch (choice) {

case 1:

viewOrderHistory();

break;

case 2:

displayMenu();

break;

case 0:

strcpy(loggedInUser, ""); // Clear logged-in username

displayMainMenu();

return;

default:

printCentered("Invalid choice. Please try again.", COLOR\_RED);

break;

}

} while (1);

}

// Function to add a new menu item

void addMenuItem() {

MenuItem item;

item.id = nextItemId++;

printf("\nEnter item name: ");

getchar(); // Consume the newline character left by previous input

fgets(item.name, sizeof(item.name), stdin);

item.name[strcspn(item.name, "\n")] = '\0'; // Remove the trailing newline character

printf("Enter item price: ");

scanf("%f", &item.price);

FILE \*file = fopen(MENU\_FILE, "a");

if (file == NULL) {

printf("Error opening menu file.\n");

return;

}

// Write the new item to the file

fprintf(file, "%d|%s|%.2f\n", item.id, item.name, item.price);

fclose(file);

printf("\nItem added successfully.\n");

}

// Function to remove a menu item

void removeMenuItem() {

int itemId, found = 0;

MenuItem items[MAX\_MENU\_ITEMS];

int itemCount = 0;

printf("\nEnter the ID of the item to remove: ");

scanf("%d", &itemId);

FILE \*menuFile = fopen(MENU\_FILE, "r");

if (menuFile == NULL) {

printf("Error opening menu file.\n");

return;

}

// Read all items into memory

while (fscanf(menuFile, "%d|%[^|]|%f", &items[itemCount].id, items[itemCount].name, &items[itemCount].price) == 3) {

if (items[itemCount].id == itemId) {

found = 1;

} else {

itemCount++;

}

}

fclose(menuFile);

if (!found) {

printf("\nItem with ID %d not found.\n", itemId);

return;

}

// Write updated list back to file with updated IDs

menuFile = fopen(MENU\_FILE, "w");

if (menuFile == NULL) {

printf("Error opening menu file.\n");

return;

}

nextItemId = 1; // Reset next available ID

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

items[i].id = nextItemId++;

fprintf(menuFile, "%d|%s|%.2f\n", items[i].id, items[i].name, items[i].price);

}

fclose(menuFile);

printf("\nMenu item removed and IDs updated successfully!\n");

}

// Function to update a menu item

void updateMenuItem() {

int itemId;

printf("\nEnter the ID of the item to update: ");

scanf("%d", &itemId);

FILE \*file = fopen(MENU\_FILE, "r");

FILE \*tempFile = fopen("temp.txt", "w");

if (file == NULL || tempFile == NULL) {

printf("Error opening menu file.\n");

return;

}

MenuItem item;

char line[150];

bool found = false;

while (fgets(line, sizeof(line), file)) {

sscanf(line, "%d|%99[^|]|%f", &item.id, item.name, &item.price);

if (item.id == itemId) {

found = true;

printf("\nEnter new item name: ");

getchar(); // Consume the newline character left by previous input

fgets(item.name, sizeof(item.name), stdin);

item.name[strcspn(item.name, "\n")] = '\0'; // Remove the trailing newline character

printf("Enter new item price: ");

scanf("%f", &item.price);

}

fprintf(tempFile, "%d|%s|%.2f\n", item.id, item.name, item.price);

}

fclose(file);

fclose(tempFile);

remove(MENU\_FILE);

rename("temp.txt", MENU\_FILE);

if (found) {

printf("\nItem updated successfully.\n");

} else {

printf("\nItem with ID %d not found.\n", itemId);

}

}

// Function to display the menu

void displayMenu() {

FILE \*file = fopen(MENU\_FILE, "r");

if (file == NULL) {

printf("Error opening menu file.\n");

return;

}

MenuItem item;

char line[150];

clearScreen();

printf("\n");

printBordered("Menu", COLOR\_MAGENTA, 80);

printf("\n");

printf("%s", COLOR\_WHITE);

printf("+----+------------------------------+--------------+\n");

printf("| %-2s | %-28s | %-9s |\n", "ID", "Item", "Price");

printf("+----+------------------------------+--------------+\n");

printf("%s", COLOR\_RESET);

while (fgets(line, sizeof(line), file)) {

sscanf(line, "%d|%99[^|]|%f", &item.id, item.name, &item.price);

printf("| %-2d | %-28s | %8.2f TK |\n", item.id, item.name, item.price);

}

printf("%s", COLOR\_WHITE);

printf("+----+------------------------------+--------------+\n");

printf("%s", COLOR\_RESET);

fclose(file);

}

void placeOrder() {

Order orders[MAX\_MENU\_ITEMS];

int orderCount = 0;

while (true) {

displayMenu(); // Display the menu for easy reference

printf("\nEnter the name of the item (or type 'done' to finish): ");

getchar(); // Consume the newline character left by previous input

fgets(orders[orderCount].itemName, sizeof(orders[orderCount].itemName), stdin);

orders[orderCount].itemName[strcspn(orders[orderCount].itemName, "\n")] = '\0'; // Remove the trailing newline character

if (strcmp(orders[orderCount].itemName, "done") == 0) {

break;

}

printf("Enter the quantity: ");

scanf("%d", &orders[orderCount].quantity);

// Find the item price from the menu

FILE \*menuFile = fopen(MENU\_FILE, "r");

if (menuFile == NULL) {

printf("Error opening menu file.\n");

return;

}

MenuItem item;

bool found = false;

char line[150];

while (fgets(line, sizeof(line), menuFile)) {

sscanf(line, "%d|%99[^|]|%f", &item.id, item.name, &item.price);

if (strcmp(item.name, orders[orderCount].itemName) == 0) {

orders[orderCount].itemPrice = item.price;

orders[orderCount].totalCost = orders[orderCount].itemPrice \* orders[orderCount].quantity;

found = true;

break;

}

}

fclose(menuFile);

if (!found) {

printf("\nItem not found in menu. Please try again.\n");

continue; // Allow user to enter another item

}

printf("\n'%s' added to cart. Quantity: %d\n", orders[orderCount].itemName, orders[orderCount].quantity);

orderCount++;

}

if (orderCount > 0) {

float grandTotal = 0.0f;

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

grandTotal += orders[i].totalCost;

}

printf("\nTotal amount to be paid: Taka %.2f\n", grandTotal);

choosePaymentMethod(orders[0].paymentMethod, grandTotal);

FILE \*file = fopen(ORDER\_FILE, "a");

if (file == NULL) {

printf("Error opening orders file.\n");

return;

}

int orderId = nextOrderId(); // Generate unique order ID for the entire order

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

orders[i].orderId = orderId;

strcpy(orders[i].customerName, loggedInUser); // Set customer name for each item

strcpy(orders[i].paymentMethod, orders[0].paymentMethod); // Set payment method for each item

fprintf(file, "%d|%s|%s|%d|%.2f|%s|%.2f\n", orders[i].orderId, orders[i].customerName, orders[i].itemName, orders[i].quantity, orders[i].itemPrice, orders[i].paymentMethod, orders[i].totalCost);

}

fclose(file);

generateInvoice(orders, orderCount);

} else {

printf("\nNo items ordered.\n");

}

}

int nextOrderId() {

FILE \*file = fopen(ORDER\_FILE, "r");

if (file == NULL) {

return 1;

}

int id = 1;

Order order;

char line[300];

while (fgets(line, sizeof(line), file)) {

sscanf(line, "%d|%99[^|]|%99[^|]|%d|%f|%19[^|]|%f", &order.orderId, order.customerName, order.itemName, &order.quantity, &order.itemPrice, order.paymentMethod, &order.totalCost);

// if (order.orderId >= id) {

// id = order.orderId + 1;

// }

id = order.orderId + 1;

}

fclose(file);

return id;

}

void choosePaymentMethod(char paymentMethod[], float grandTotal) {

int paymentChoice;

float cashReceived, change;

char phoneNumber[20];

printf("\nChoose payment method:\n");

printf("1. Cash\n");

printf("2. Credit Card\n");

printf("3. Bkash\n");

printf("4. Nagad\n");

printf("Enter your choice: ");

scanf("%d", &paymentChoice);

switch (paymentChoice) {

case 1:

printf("\nEnter cash received: ");

scanf("%f", &cashReceived);

change = cashReceived - grandTotal;

if (change < 0) {

printf("\nInsufficient cash received. Transaction cancelled.\n");

exit(0);

}

strcpy(paymentMethod, "Cash");

printf("\nChange: %.2f\n", change);

break;

case 2:

printf("\nEnter your credit card number: ");

scanf("%s", paymentMethod);

strcpy(paymentMethod, "Credit Card");

printf("\nSuccessfully paid %.2f Taka using Credit Card.\n", grandTotal);

break;

case 3:

printf("\nEnter your Bkash phone number: ");

scanf("%s", phoneNumber);

strcpy(paymentMethod, "Bkash");

strcat(paymentMethod, " (Phone: ");

strcat(paymentMethod, phoneNumber);

strcat(paymentMethod, ")");

printf("\nSuccessfully paid %.2f Taka using Bkash.\n", grandTotal);

break;

case 4:

printf("\nEnter your Nagad phone number: ");

scanf("%s", phoneNumber);

strcpy(paymentMethod, "Nagad");

strcat(paymentMethod, " (Phone: ");

strcat(paymentMethod, phoneNumber);

strcat(paymentMethod, ")");

printf("\nSuccessfully paid %.2f Taka using Nagad.\n", grandTotal);

break;

default:

printf("\nInvalid choice. Transaction cancelled.\n");

exit(0);

}

}

void viewOrderHistory() {

FILE \*file = fopen(ORDER\_FILE, "r");

if (file == NULL) {

printf("Error opening orders file.\n");

return;

}

Order order;

char line[300];

printf("\nOrder History:\n");

printf("ID | Customer | Item | Quantity | Price | Payment | Total Cost\n");

printf("---------------------------------------------------------------------\n");

while (fgets(line, sizeof(line), file)) {

sscanf(line, "%d|%99[^|]|%99[^|]|%d|%f|%19[^|]|%f", &order.orderId, order.customerName, order.itemName, &order.quantity, &order.itemPrice, order.paymentMethod, &order.totalCost);

printf("%d | %s | %s | %d | %.2f | %s | %.2f\n", order.orderId, order.customerName, order.itemName, order.quantity, order.itemPrice, order.paymentMethod, order.totalCost);

}

fclose(file);

}

// Function to generate an invoice

void generateInvoice(Order orders[], int orderCount)

{

printf("\n");

printBordered("Invoice", COLOR\_MAGENTA, 80);

printf("\n");

printf("%s", COLOR\_WHITE);

printf("+----+------------------------------+-----------+----------+--------------+\n");

printf("| %-2s | %-28s | %-9s | %-8s | %-12s |\n", "No", "Item", "Quantity", "Price", "Total Cost");

printf("+----+------------------------------+-----------+----------+--------------+\n");

printf("%s", COLOR\_RESET);

float grandTotal = 0.0f;

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

printf("| %-2d | %-28s | %-9d | $%-8.2f| $%-11.2f |\n", orders[i].orderId, orders[i].itemName, orders[i].quantity, orders[i].itemPrice, orders[i].totalCost);

grandTotal += orders[i].totalCost;

}

printf("%s", COLOR\_WHITE);

printf("+----+------------------------------+-----------+----------+--------------+\n");

printf("| %-57s| %-10.2f TK|\n", "Total", grandTotal);

printf("+----+------------------------------+-----------+----------+--------------+\n");

printf("%s", COLOR\_RESET);

}

// Function to terminate the program

void terminateProgram() {

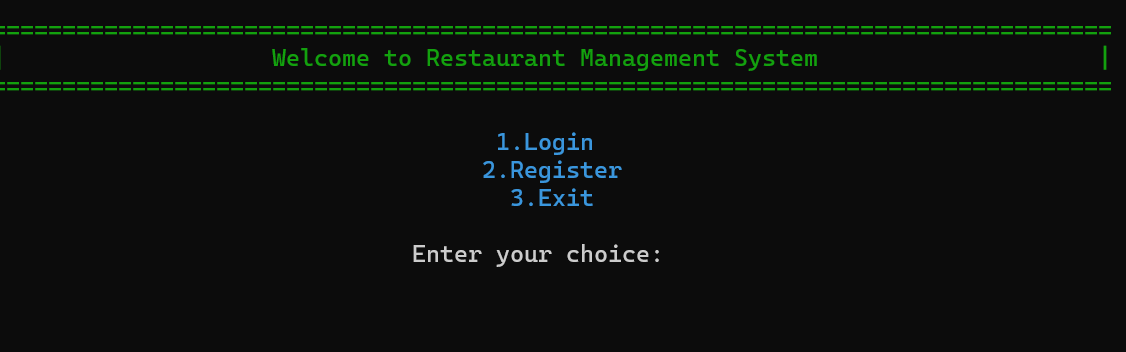
printf("Thank you for using the Restaurant Management System.\n");

exit(0);

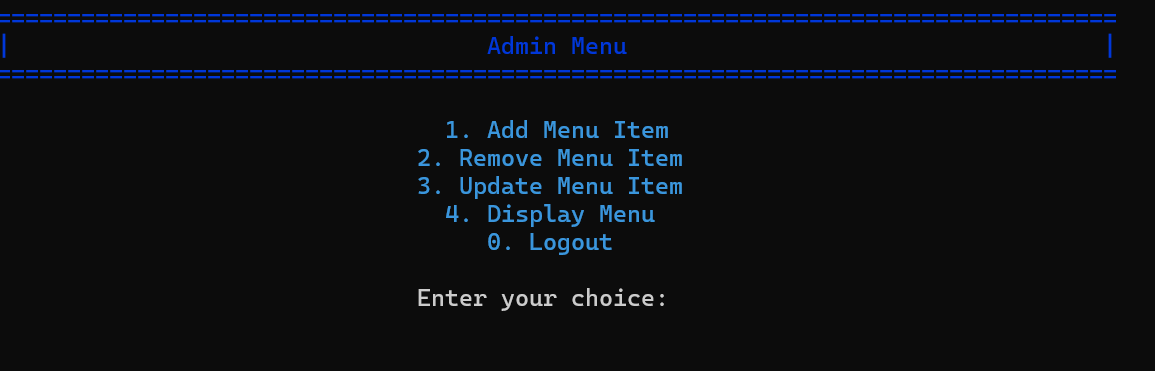
}

**Sample Output:**

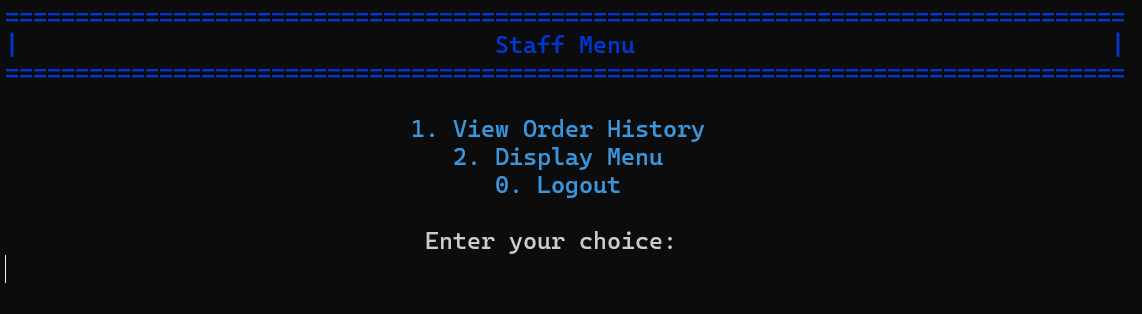
**First Glance:**

****

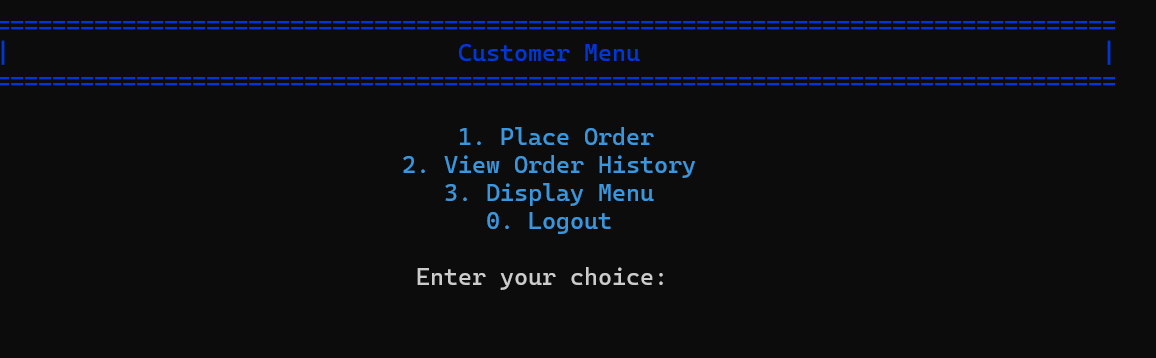
**Admin:**

****

**Staff:**

****

**Customer:**

****