**Part:02**

**ID: 232-35-746**

**How It Works:**

1. **Menu Item Management**:
   * You can add, delete, and update menu items.
   * Menu items are stored in an array of “MenuItem” structures.
2. **Order Management**:
   * You can place orders by selecting items from the menu and specifying the quantity.
   * Orders are stored in an array of “Order” structures.
3. **Menu and Order Display**:
   * The menu can be displayed with item names and prices.
   * Orders are displayed with the item names and their quantities.

**Order and Menu Management(code):**

#include <stdio.h>

#include <string.h>

#define MAX\_MENU\_ITEMS 50

#define MAX\_ORDER\_ITEMS 50

// Structure for Menu Item

struct MenuItem {

char name[50];

float price;

};

// Structure for Order

struct Order {

char itemName[50];

int quantity;

};

// Global variables

struct MenuItem menu[MAX\_MENU\_ITEMS];

struct Order orders[MAX\_ORDER\_ITEMS];

int menuCount = 0;

int orderCount = 0;

// Function declarations

void addMenuItem();

void displayMenu();

void deleteMenuItem();

void updateMenuItem();

void placeOrder();

void displayOrders();

int main() {

int choice;

while(1) {

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

printf("1. Add Menu Item\n");

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

printf("3. Delete Menu Item\n");

printf("4. Update Menu Item\n");

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

printf("6. Display Orders\n");

printf("7. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch(choice) {

case 1:

addMenuItem();

break;

case 2:

displayMenu();

break;

case 3:

deleteMenuItem();

break;

case 4:

updateMenuItem();

break;

case 5:

placeOrder();

break;

case 6:

displayOrders();

break;

case 7:

printf("Thanks for using the system.\n");

return 0;

default:

printf("Invalid choice, please try again.\n");

}

}

return 0;

}

// Function to add a menu item

void addMenuItem() {

if(menuCount < MAX\_MENU\_ITEMS) {

printf("Enter the name of the menu item: ");

getchar(); // Clear the buffer

fgets(menu[menuCount].name, 50, stdin);

menu[menuCount].name[strcspn(menu[menuCount].name, "\n")] = 0; // Remove newline character

printf("Enter the price of the item: ");

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

menuCount++;

printf("Menu item added successfully!\n");

} else {

printf("Menu is full. Cannot add more items.\n");

}

}

// Function to display the menu

void displayMenu() {

if(menuCount == 0) {

printf("The menu is currently empty.\n");

} else {

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

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

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

}

}

}

// Function to delete a menu item

void deleteMenuItem() {

if(menuCount == 0) {

printf("Menu is empty. Nothing to delete.\n");

return;

}

int itemIndex;

printf("Enter the index of the menu item to delete (1 to %d): ", menuCount);

scanf("%d", &itemIndex);

if(itemIndex < 1 || itemIndex > menuCount) {

printf("Invalid index!\n");

} else {

// Shift all items after the deleted one

for(int i = itemIndex - 1; i < menuCount - 1; i++) {

menu[i] = menu[i + 1];

}

menuCount--;

printf("Menu item deleted successfully.\n");

}

}

// Function to update a menu item

void updateMenuItem() {

if(menuCount == 0) {

printf("Menu is empty. Nothing to update.\n");

return;

}

int itemIndex;

printf("Enter the index of the menu item to update (1 to %d): ", menuCount);

scanf("%d", &itemIndex);

if(itemIndex < 1 || itemIndex > menuCount) {

printf("Invalid index!\n");

} else {

itemIndex--; // Adjust index to 0-based

printf("Updating item: %s\n", menu[itemIndex].name);

printf("Enter the new name: ");

getchar(); // Clear the buffer

fgets(menu[itemIndex].name, 50, stdin);

menu[itemIndex].name[strcspn(menu[itemIndex].name, "\n")] = 0; // Remove newline character

printf("Enter the new price: ");

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

printf("Menu item updated successfully.\n");

}

}

// Function to place an order

void placeOrder() {

if(menuCount == 0) {

printf("The menu is empty, cannot place an order.\n");

return;

}

int orderIndex, quantity;

printf("Enter the index of the item to order (1 to %d): ", menuCount);

scanf("%d", &orderIndex);

if(orderIndex < 1 || orderIndex > menuCount) {

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

} else {

orderIndex--; // Adjust to 0-based index

printf("Enter the quantity of %s: ", menu[orderIndex].name);

scanf("%d", &quantity);

if(orderCount < MAX\_ORDER\_ITEMS) {

strcpy(orders[orderCount].itemName, menu[orderIndex].name);

orders[orderCount].quantity = quantity;

orderCount++;

printf("Order placed successfully!\n");

} else {

printf("Order list is full. Cannot place more orders.\n");

}

}

}

// Function to display all orders

void displayOrders() {

if(orderCount == 0) {

printf("No orders have been placed yet.\n");

} else {

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

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

printf("%d. %s - Quantity: %d\n", i+1, orders[i].itemName, orders[i].quantity);

}

}

}

**Output:**

The system provides the following choices:

1. **Add Menu Item**: Add a new item to the menu.
2. **Display Menu**: Show all the items in the menu.
3. **Delete Menu Item**: Remove an item from the menu.
4. **Update Menu Item**: Update the name or price of a menu item.
5. **Place Order**: Place an order by choosing items from the menu.
6. **Display Orders**: Show all the orders that have been placed.
7. **Exit**: Exit the program.

**Reporting and Analysis Management:**

#include <stdio.h>

#define MAX\_ORDERS 100

// Define the structure for an Order

typedef struct {

int orderId;

float totalAmount;

} Order;

// Function to generate the report based on sales

void generateReport(Order orders[], int orderCount) {

float totalSales = 0.0;

// Calculate total sales

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

totalSales += orders[i].totalAmount;

}

// Display the report

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

printf("Total Orders: %d\n", orderCount);

printf("Total Sales: $%.2f\n", totalSales);

printf("Average Sales per Order: $%.2f\n", orderCount > 0 ? totalSales / orderCount : 0.0);

}

int main() {

Order orders[MAX\_ORDERS];

int orderCount = 0;

// Loop to continuously accept orders

while (1) {

printf("\nEnter Order ID (Enter 0 to stop): ");

int orderId;

scanf("%d", &orderId);

// Exit condition: if the user enters 0, stop entering orders

if (orderId == 0) {

break;

}

printf("Enter Total Amount: ");

float totalAmount;

scanf("%f", &totalAmount);

// Store the order

orders[orderCount].orderId = orderId;

orders[orderCount].totalAmount = totalAmount;

orderCount++;

}

// Generate the sales report

generateReport(orders, orderCount);

return 0;

}

**Output:**

The system provides the following choices:

Enter Order ID (Enter 0 to stop): 101

Enter Total Amount: 45.50

Enter Order ID (Enter 0 to stop): 102

Enter Total Amount: 30.25

Enter Order ID (Enter 0 to stop): 103

Enter Total Amount: 55.75

Enter Order ID (Enter 0 to stop): 0

**After pressing enter:**

----- Sales Report -----

Total Orders: 3

Total Sales: $131.50

Average Sales per Order: $43.83

**Explanation of the Output:**

* **Total Orders**: There are 3 orders entered (Order ID 101, 102, 103).
* **Total Sales**: The total sales amount is the sum of all the order amounts: 45.50+30.25+55.75=131.5045.50 + 30.25 + 55.75 = 131.5045.50+30.25+55.75=131.50
* **Average Sales per Order**: The average is calculated by dividing the total sales by the number of orders: 131.503=43.83\frac{131.50}{3} = 43.833131.50​=43.83 The result is rounded to two decimal places.