

CIA 3 : PART-B

**TOPIC: RESTAURANT BILLING SYSTEM**

Submission Date: 9 September 2024

Under the guidance of

DR. SHONEY SEBASTIAN

Submitted By -

Group 3

Team Members:

1. Saumya Dwivedi- 2447149
2. Rakshit Srivastava- 2447140
3. Kavya Poddar- 2447126

**Restaurant Billing System**

The Restaurant Billing System (RBS) is a comprehensive software solution designed to streamline and manage restaurant operations. It enables users to interact with a digital menu, place orders, and generate invoices. The system supports both vegetarian and non-vegetarian menu categories, along with options for drinks and desserts, allowing customers to select items, specify quantities, and view the total cost. By automating these processes, RBS reduces manual handling and errors.

Central to the RBS are its data structures, which include items, orders, and tableBooking. The items structure holds details of each menu item, such as name, price, and quantity. The orders structure captures all relevant information about a customer's order, including the customer's name, order date, and a list of items ordered. The tableBooking structure manages table reservations, storing information about the customer, booking date, table number, number of people, and time slot. These structures facilitate organized and efficient data management.

The RBS offers several functionalities to enhance user experience. Users can choose menu items from Veg, Non-Veg, Drinks and Desserts, with the ability to view prices and make selections. Once an order is placed, it generates a detailed invoice, including itemized costs, discounts, taxes, and the final amount. Additionally, the RBS allows users to save invoices and provides an option to view past invoices and table bookings. This ensures that all records and reservations are accurately maintained and easily accessible.

In terms of table management, the RBS handles table bookings by allowing customers to select a table, specify the number of people, and choose a time slot. The system checks for existing bookings to avoid conflicts and ensures that tables are not double-booked.

Overall, the Restaurant Billing System is designed to simplify and improve the efficiency of restaurant operations. By integrating menu management, order processing, and table reservations into a single application, the RBS enhances customer service, reduces manual workload, and ensures accurate record-keeping. The system’s robust functionality make it a valuable tool for modern restaurants seeking to modernize their operations and provide a better dining experience.

**Features Used in The Program**

1.Preprocessor Directives

These are the preprocessor directives that instruct the compiler before the commencement of compilation. They are: #include, #define, declarations of constants.

Usage:  
#include <stdio.h>: Includes the standard input-output library for basic operations like printf and scanf.

#include <stdlib.h>: Includes the standard library for functions like malloc, free, and exit.

#include<string.h>: Include the standard library for functions like strcmp,

Strcpy.

Example:  
#include <stdio.h>

#include <stdlib.h>

#include<string.h>

2. Data Types & Variables

Description: Variables store data of different types, such as int, char, float, and more. These types define the format and size of the data.

Usage:  
int choice; : Stores the choice input by user.

int no\_of\_item; : Stores the number of item.

char name[30]; : Stores the name.

Example:

char temp = 'y'; // Use a single character for the loop condition

3. Menu Display

**Description**:

* Purpose: To display different categories of food and beverages available at the restaurant.
* How it Works: The program includes separate functions to display the Veg, Non-Veg, Drinks, and Desserts menus. Each menu function (displayVegMenu, displayNonVegMenu, displayDrinksMenu, displayDessertsMenu) iterates through predefined arrays of menu items and prints them with their prices.

4. Table Booking

**Description:**

* Purpose: To allow customers to book tables at the restaurant, with the ability to check table availability.
* How it Works: The program prompts the user for details such as the customer’s name, table number, time slot, and number of people. It checks existing bookings in the TableBooking.txt file to ensure the table is not already reserved for the requested time slot. If the table is available, the booking is saved to the file.

5. Order Placement and Invoice Generation

**Description:**

* Purpose: To allow customers to place orders from different menus and generate an invoice for the order.
* How it Works**:** Users select items from the Veg, Non-Veg, Drinks, or Desserts menus and specify quantities. The order details are stored in an orders structure and written to the RestaurantBill.txt file. The invoice is then generated and displayed, including itemized details and final amounts with calculations for subtotal, discount, taxes, and grand total.

6. Invoice Management

**Description:**

* Purpose: To manage and retrieve previously generated invoices.
* How it Works: The program can display all previous invoices from the RestaurantBill.txt file or search for invoices based on the customer’s name. This functionality allows users to view past orders and generate a summary of each invoice.

7. File Operations

**Description:**

* Purpose: To save and retrieve data related to table bookings and invoices.
* How it Works:
  + Saving Bookings: Table bookings are appended to TableBooking.txt using binary write operations.
  + Saving Invoices**:** Orders and invoices are saved to RestaurantBill.txt using binary write operations.
  + Reading Bookings and Invoices: The program reads from these files using binary read operations to display existing bookings and previous invoices.

8. User Interaction

**Description:**

* Purpose: To provide an interactive interface for the user to perform various operations.
* How it Works: The program uses a loop to present a menu of options to the user. Based on the user's choice, different functionalities (like displaying menus, booking tables, placing orders, etc.) are executed. It prompts the user for input and provides feedback based on the user's actions.

9.Data Structures

**Description:**

* Purpose: To organize and manage data related to menu items, orders, and bookings.
* How it Works:
  + items Structure: Holds details about individual menu items including name, price, and quantity.
  + orders Structure: Stores information about customer orders including the customer’s name, date, number of items, and the items ordered.
  + tableBooking Structure: Contains details about table bookings including the customer’s name, table number, time slot, number of people, and date.

10. Invoice Formatting and Calculation

**Description:**

* Purpose: To format and calculate the details of the invoice.
* How it Works: The program calculates the total amount based on the ordered items and their quantities, applies a discount, calculates taxes (CGST and SGST), and displays the final invoice with all relevant details. Functions like generateBillHeader, generateBillBody, and generateBillFooter handle formatting and calculation.

11. Error Handling

**Description:**

* Purpose: To handle errors gracefully, such as issues with file operations or invalid user input.
* How it Works: The program includes checks to ensure that files are opened successfully, inputs are validated, and proper messages are displayed if errors occur (e.g., unable to open files, invalid menu selections, etc.).

12. Loop and Conditional Statements

**Description:**

* Purpose: To control the flow of the program based on user choices and input.
* How it Works: The main menu loop repeatedly presents options to the user until they choose to exit. Inside the loop, switch and if-else statements are used to handle different operations based on user input.

**Functionalities Of The Program**

1. displayVegMenu()

Description: Displays the vegetarian menu. It iterates through the vegMenu array and prints each item with its index, name, and price.

2. displayNonVegMenu()

Description: Displays the non-vegetarian menu. It iterates through the nonVegMenu array and prints each item with its index, name, and price.

3. displayDrinksMenu()

Description: Displays the drinks menu. It iterates through the drinksMenu array and prints each item with its index, name, and price.

4. displayDessertsMenu()

Description: Displays the desserts menu. It iterates through the dessertsMenu array and prints each item with its index, name, and price.

5. generateBillHeader(char name[50], char date[30])

Description: Generates the header for the bill. It prints the restaurant name, date of the invoice, the customer's name, and column headers for item details. This function prepares the invoice for the itemized billing.

6. generateBillBody(char item[20], int qty, float price)

Description: Generates a line item in the bill. It prints the item's name, quantity, price per unit, and the total price for that item. This function provides a detailed breakdown of each item ordered.

7. generateBillFooter(float total)

Description: Generates the footer for the bill. It calculates and prints the subtotal, discount (10%), net total after discount, and taxes (CGST and SGST). It then prints the grand total, providing a final summary of the invoice.

8. displayBookings()

Description: Displays all table bookings. It reads from the TableBooking.txt file and prints the details of each booking including the customer's name, table number, time slot, and number of people. This function is used to show existing table reservations.

**Additional Descriptions for Main Functionality:**

* main(): The main function controls the flow of the program. It presents a menu to the user with options to display menus, book tables, place orders and generate invoices, view previous invoices, search invoices by customer name, display all table bookings, or exit the program. It uses a loop to repeatedly present the menu until the user chooses to exit.

**Functions in main():**

* case 1: Displays menu options. Calls appropriate menu display functions based on user choice.
* case 2: Books a table by checking availability in the TableBooking.txt file and saving new bookings.
* case 3: Handles order placement. Allows users to select items from the menu, specify quantities, and generate an invoice saved in RestaurantBill.txt.
* case 4: Reads and displays all previously generated invoices from the RestaurantBill.txt file.
* case 5: Searches for and displays invoices for a specific customer by name.
* case 6: Displays all table bookings by reading from TableBooking.txt.
* case 7: Exits the program.

Each function is integral to managing different aspects of the restaurant's operations, including menu display, order management, invoicing, and booking.

**Source code:**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

//Functions used in the code.

void displayVegMenu();

void displayNonVegMenu();

void displayDrinksMenu();

void displayDessertsMenu();

void generateBillHeader(char name[50], char date[30]);

void generateBillBody(char item[20], int qty, float price);

void generateBillFooter(float total);

void displayBookings();

typedef struct items {

char item[20];

float price;

int qty;

} items;

typedef struct orders {

char customer[50];

char date[50];

int numOfItems;

items itm[50]; // array of structures

} orders;

typedef struct tableBooking {

char customer[50];

char date[50];

int tableNumber;

int numberOfPeople;

char timeSlot[20];

} tableBooking;

// Initializes an array of items structures representing the Veg menu. Each item has a name and a price.

items vegMenu[] = {

{"Pizza", 149},

{"Pasta", 99},

{"Burger", 149},

{"Uttpam", 100},

{"Noodles", 49},

{"VegIdli", 20},

{"Veggie", 79}

};

// Initializes an array of items structures representing the Non-Veg menu. Each item has a name and a price.

items nonVegMenu[] = {

{"Chicken", 99},

{"Kawab", 149},

{"Mutton", 149},

{"Fishes", 199},

{"Biryani", 149}

};

// Initializes an array of items structures representing the Drinks menu. Each item has a name and a price.

items drinksMenu[] = {

{"Water", 20},

{"Sprite", 49},

{"Juice", 99},

{"Tea ", 59},

{"Coffee", 25},

{"Lassi", 75}

};

// Initializes an array of items structures representing the Desserts menu. Each item has a name and a price.

items dessertsMenu[] = {

{"Cake " , 99},

{"Brownie", 89},

{"Pastry", 79},

{"Sweets", 30},

};

int vegMenuSize = sizeof(vegMenu) / sizeof(vegMenu[0]);

int nonVegMenuSize = sizeof(nonVegMenu) / sizeof(nonVegMenu[0]);

int drinksMenuSize = sizeof(drinksMenu) / sizeof(drinksMenu[0]);

int dessertsMenuSize = sizeof(dessertsMenu) / sizeof(dessertsMenu[0]);

int main() {

int choice, no\_of\_item, menuType, menuChoice;

char temp = 'y'; // Use a single character for the loop condition

char name[30]; // Declare the name variable here

FILE \*ptr; // Used to save bill

orders ord;

orders ord1;

tableBooking booking;

tableBooking existingBooking;

while (temp == 'y' || temp == 'Y') {

float total = 0.0;

int invoiceFound = 0;

printf("============ The Only Place ============");

printf("\n\nPlease select your preferred operation");

printf("\n\n1. Display menus.");

printf("\n2. Book your table.");

printf("\n3. Place order and generate invoice.");

printf("\n4. Show all Invoices.");

printf("\n5. Search for Invoices by customer name.");

printf("\n6. Display all table bookings");

printf("\n7. Exit");

printf("\n\nYour Choice:\t");

scanf("%d", &choice);

fgetc(stdin); // Consume newline character left by scanf

switch (choice) {

//Display all menus.

case 1:

system("cls");

// Display all sub-menu options once

printf("\nPlease choose a menu to display:");

printf("\n1. Veg Menu");

printf("\n2. Non-Veg Menu");

printf("\n3. Drinks Menu");

printf("\n4. Desserts Menu");

printf("\n5. Back to main menu\n");

// Loop to handle multiple sub-menu displays

while (1) {

int menuType;

printf("\nEnter the number of the sub-menu you want to see:\t");

scanf("%d", &menuType);

switch(menuType) {

case 1:

displayVegMenu();

break;

case 2:

displayNonVegMenu();

break;

case 3:

displayDrinksMenu();

break;

case 4:

displayDessertsMenu();

break;

case 5:

break; // Exit the loop to return to the main menu

default:

printf("\nInvalid selection. Try again.\n");

continue;

}

if(menuType == 5){

break;

}

printf("\nDo you want to see another menu? (y/n):\t");

scanf(" %c", &choice);

// Exit the loop if the user doesn't want to see another menu

if (choice != 'y' && choice != 'Y') {

break;

}

}

break;

// Book a Table with Condition

case 2:

system("cls");

printf("Existing Booked Tables:\n");

displayBookings();

printf("\nEnter name of customer: ");

fgets(booking.customer, 50, stdin);

booking.customer[strlen(booking.customer) - 1] = 0; // Remove newline character

printf("Enter table number (1-10): ");

scanf("%d", &booking.tableNumber);

fgetc(stdin); // Consume newline character

printf("Enter time slot (e.g., 6:00 PM - 8:00 PM): ");

fgets(booking.timeSlot, 20, stdin);

booking.timeSlot[strlen(booking.timeSlot) - 1] = 0; // Remove newline character

// Check if the table is already booked for the entered time slot

int isBooked = 0;

ptr = fopen("TableBooking.txt", "rb");

if (ptr != NULL) {

while (fread(&existingBooking, sizeof(tableBooking), 1, ptr)) {

if (existingBooking.tableNumber == booking.tableNumber &&

strcmp(existingBooking.timeSlot, booking.timeSlot) == 0) {

isBooked = 1;

break;

}

}

fclose(ptr);

}

if (isBooked) {

printf("\nSorry, Table %d is already booked for the time slot %s.\n", booking.tableNumber, booking.timeSlot);

}

else {

printf("Enter number of people (Maximum 5): ");

scanf("%d", &booking.numberOfPeople);

fgetc(stdin); // Consume newline character

if (booking.numberOfPeople > 5 || booking.numberOfPeople < 0 ) {

printf("You can book a maximum of 5 people only.\n");

}

else {

ptr = fopen("TableBooking.txt", "ab+");

if (ptr == NULL) {

printf("Error: Unable to open file for writing.\n");

break;

}

fwrite(&booking, sizeof(tableBooking), 1, ptr);

fclose(ptr);

printf("Table booked successfully.\n");

}

}

break;

// Place an Order and Generate Invoice

case 3:

system("cls");

printf("\nEnter name of customer: ");

fgets(ord.customer, 50, stdin);

ord.customer[strlen(ord.customer) - 1] = 0; // Remove newline character

//Used to diaply current date.

strcpy(ord.date, \_DATE\_); // (destination, source)

printf("How many items would you like to order?\t");

scanf("%d", &ord.numOfItems);

fgetc(stdin); // Consume newline character

for (int i = 0; i < ord.numOfItems; i++) {

printf("\nYour menus are:");

printf("\n1. Veg Menu");

printf("\n2. Non-Veg Menu");

printf("\n3. Drinks Menu");

printf("\n4. Desserts Menu\n");

printf("\nPlease choose a menu for item %d: ", i + 1);

int menuType;

scanf("%d", &menuType);

fgetc(stdin); // Consume newline character

int menuSize;

if (menuType == 1) {

menuSize = vegMenuSize;

displayVegMenu();

}

else if (menuType == 2) {

menuSize = nonVegMenuSize;

displayNonVegMenu();

}

else if (menuType == 3) {

menuSize = drinksMenuSize;

displayDrinksMenu();

}

else if (menuType == 4) {

menuSize = dessertsMenuSize;

displayDessertsMenu();

}

else {

printf("\nInvalid selection. Try again.\n");

i--; // Repeat this iteration

continue;

}

printf("\nEnter item number from the selected menu:\t");

int itemIndex;

scanf("%d", &itemIndex);

fgetc(stdin); // Consume newline character

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

printf("Invalid item number. Please try again.\n");

i--; // Repeat this iteration

continue;

}

if (menuType == 1) {

strcpy(ord.itm[i].item, vegMenu[itemIndex - 1].item);

ord.itm[i].price = vegMenu[itemIndex - 1].price;

}

else if (menuType == 2) {

strcpy(ord.itm[i].item, nonVegMenu[itemIndex - 1].item);

ord.itm[i].price = nonVegMenu[itemIndex - 1].price;

}

else if (menuType == 3) {

strcpy(ord.itm[i].item, drinksMenu[itemIndex - 1].item);

ord.itm[i].price = drinksMenu[itemIndex - 1].price;

}

else if (menuType == 4) {

strcpy(ord.itm[i].item, dessertsMenu[itemIndex - 1].item);

ord.itm[i].price = dessertsMenu[itemIndex - 1].price;

}

printf("Enter quantity of %s:\t", ord.itm[i].item);

scanf("%d", &ord.itm[i].qty);

fgetc(stdin); // Consume newline character

}

// Generate and save the bill

FILE \*ptr = fopen("RestaurantBill.txt", "ab+");

if (ptr == NULL) {

printf("Error: Unable to open file for writing.\n");

break;

}

fwrite(&ord, sizeof(orders), 1, ptr);

fclose(ptr);

printf("\n ========== Generating Your Invoice ==========\n");

float total = 0;

generateBillHeader(ord.customer, ord.date);

for (int i = 0; i < ord.numOfItems; i++) {

generateBillBody(ord.itm[i].item, ord.itm[i].qty, ord.itm[i].price);

total += ord.itm[i].qty \* ord.itm[i].price;

}

generateBillFooter(total);

printf("\nInvoice generated successfully.\n");

break;

//Display all invoices

case 4:

system("cls");

ptr = fopen("RestaurantBill.txt", "rb"); // Open file in read mode

if (ptr == NULL) {

printf("\nError: Unable to open file or no invoices found.\n");

break;

}

printf("\n ========== Your Previous Invoices ==========\n");

while (fread(&ord1, sizeof(orders), 1, ptr)) {

float tot = 0;

generateBillHeader(ord1.customer, ord1.date);

for (int i = 0; i < ord1.numOfItems; i++) {

generateBillBody(ord1.itm[i].item, ord1.itm[i].qty, ord1.itm[i].price);

tot += ord1.itm[i].qty \* ord1.itm[i].price;

}

generateBillFooter(tot);

}

fclose(ptr);

break;

//Search for the invoices by customer name.

case 5:

printf("Enter the name of the customer:\t");

fgets(name, 30, stdin);

name[strlen(name) - 1] = 0; // Remove newline character

system("cls");

ptr = fopen("RestaurantBill.txt", "rb"); // Open file in read mode

if (ptr == NULL) {

printf("\nError: Unable to open file or no invoices found.\n");

break;

}

printf("\t ========== Invoice of %s ==========\n", name);

invoiceFound = 0; // Initialize invoiceFound = 0 (false)

while (fread(&ord1, sizeof(orders), 1, ptr)) {

float tot = 0;

if (strcmp(ord1.customer, name) == 0) {

generateBillHeader(ord1.customer, ord1.date);

for (int i = 0; i < ord1.numOfItems; i++) {

generateBillBody(ord1.itm[i].item, ord1.itm[i].qty, ord1.itm[i].price);

tot += ord1.itm[i].qty \* ord1.itm[i].price;

}

generateBillFooter(tot);

invoiceFound = 1; // Set flag to true if invoice is found

}

}

if (!invoiceFound) {

printf("Sorry the invoice for %s does not exist", name);

}

fclose(ptr);

break;

// Display All Bookings

case 6:

system("cls");

displayBookings();

break;

//Exit the program

case 7:

printf("\n\t\t Bye Bye :)\n\n");

exit(0);

break;

default:

printf("Sorry invalid option.\n");

break;

}

printf("\nDo you want to perform another operation (y/n):\t");

scanf(" %c", &temp); // Note the space before %c to consume any leftover newline

}

printf("\n\t\t Bye Bye :)\n\n");

return 0;

}

// Function to display the Veg menu

void displayVegMenu() {

printf("\n============== Veg Menu =============\n");

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

printf("%d. %s\t\tRs. %.2f /-\n", i + 1, vegMenu[i].item, vegMenu[i].price);

}

printf("=====================================\n");

}

// Function to display the Non-Veg menu

void displayNonVegMenu() {

printf("\n============== Non-Veg Menu ========\n");

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

printf("%d. %s\t\tRs.%.2f /-\n", i + 1, nonVegMenu[i].item, nonVegMenu[i].price);

}

printf("====================================\n");

}

//Function to display drinks menu

void displayDrinksMenu(){

printf("\n============== Drinks Menu ========\n");

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

printf("%d. %s\t\tRs.%.2f /-\n", i + 1, drinksMenu[i].item, drinksMenu[i].price);

}

printf("===================================\n");

}

//Function to display desserts menu

void displayDessertsMenu(){

printf("\n============== Desserts Menu =======\n");

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

printf("%d. %s\t\tRs.%.2f /-\n", i + 1, dessertsMenu[i].item, dessertsMenu[i].price);

}

printf("===================================\n");

}

// Function to generate bill header

void generateBillHeader(char name[50], char date[30]) {

printf("\n\n");

printf("\t\t The Only Place");

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

printf("\nDate: %s", date);

printf("\nInvoice To: %s", name);

printf("\n");

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

printf("Items\t\t");

printf("Qty\t\t");

printf("Price\t\t");

printf("Total\t\t");

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

printf("\n\n");

}

// Function to generate bill body

void generateBillBody(char item[20], int qty, float price) {

printf("%s\t\t", item);

printf("%d\t\t", qty);

printf("%.2f /-\t", price);

printf("%.2f /-\t", qty \* price);

printf("\n");

}

// Function to generate bill footer

void generateBillFooter(float total) {

printf("\n");

float dis = 0.1 \* total;

float netTotal = total - dis;

float cgst = 0.09 \* netTotal, grandTotal = netTotal + 2 \* cgst;

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

printf("Sub Total\t\t\t\t\t%.2f /-", total);

printf("\nDiscount @10%%\t\t\t\t\t%.2f /-", dis);

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

printf("\nNet Total\t\t\t\t\t%.2f /-", netTotal);

printf("\nCGST @9%%\t\t\t\t\t%.2f /-", cgst);

printf("\nSGST @9%%\t\t\t\t\t%.2f /-", cgst);

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

printf("\nGrand Total\t\t\t\t\t%.2f /-", grandTotal);

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

}

void displayBookings() {

FILE \*ptr;

tableBooking booking;

ptr = fopen("TableBooking.txt", "rb");

if (ptr == NULL) {

printf("Error: Unable to open file.\n");

return;

}

// Existing Table Bookings:

while (fread(&booking, sizeof(tableBooking), 1, ptr)) {

printf("Customer: %s, Table Number: %d, Time Slot: %s, Number of People: %d\n",

booking.customer, booking.tableNumber, booking.timeSlot, booking.numberOfPeople);

}

  fclose(ptr);

}

**OUTPUT:**





































