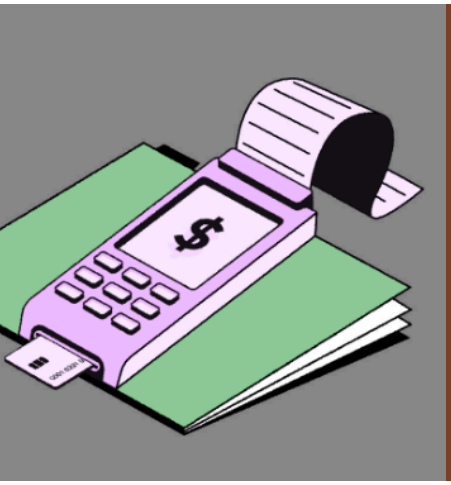


C PROGRAMMING PROJECT

CANTEEN BILLING SYSTEM

STUDENT DETAILS

SUBMITTED BY:	ANJALI KALRA
SAP ID / BATCH:	590024902/ B-20
SUBMITTED TO:	MR RAHUL PRASAD
ACADEMIC YEAR	2025-29

GITHUB LINK:

GitHub - anjalikalra1975/Canteen-billing-system_-C-project

ABSTRACT

The Canteen Billing System is a simple C-based program designed to automate the process of creating customer bills in a college canteen. The project loads menu items such as item ID, name, and price from an external text file and allows the user to select items, enter quantities, and generate a final bill. It uses structures to organize menu and order information, file handling to store and retrieve past bills, and basic validation to ensure correct user input. The system also records the date, time, and unique bill number for every transaction, giving it a realistic functioning similar to an actual billing machine. Overall, the project demonstrates practical use of C concepts such as functions, arrays, structures, loops, file operations, and time handling to build a simple yet functional application.

PROBLEM DEFINITION

The canteen needs a simple system to **display the menu**, **take customer orders**, **calculate bills**, and **store each transaction** for future reference. Current manual billing is slow, error-prone, and does not maintain proper records.

This project solves the problem by:

- Loading menu items from a file
- Allowing users to select items and quantities
- Automatically calculating totals
- Generating a bill with date, time, and unique ID
- Saving each bill in a history file for tracking

SYSTEM DESIGN

The system design explains how the canteen billing software is structured, how data flows, and how each part of the program works together to generate bills smoothly.

1. Architecture Overview

1. Follows a **modular architecture** with separate files for main logic, functions, and declarations.

2. Uses **three core components**:

    a. User Interface (main menu)

    b. Processing Unit (billing & searching logic)

    c. Storage Layer (menu.txt and history.txt)

3. Storage Layer (menu.txt and history.txt)

2. Data Flow Design

- **Step 1:** Program loads menu items from menu.txt into memory.
- **Step 2:** User selects actions from main menu.
- **Step 3:** During billing, item IDs and quantities are taken as input.
- **Step 4:** The system calculates subtotal + total.
- **Step 5:** Bill details are saved to history.txt.
- **Step 6:** Program returns to main menu.

3. Functional Design (Core Modules)

1. **load Menu()** – Reads the menu file & fills the menu array.
2. **main Menu()** – Handles user navigation.

3. 1. **generate Bill()** – Takes orders, calculates totals, prints the bill, saves it.
4. 2. **view History()** – Reads and displays past bill records.
5. **get Current Date Time()** – Generates formatted date & time for receipts.

4. File Structure Design

- **menu.txt**: Stores item ID, item name, price.
- **history.txt**: Stores bill ID, date, time, total amount.
- **main .c**: Controls program flow.
- **functions .c**: Contains all logic functions.
- **headers .h**: Contains struct definitions & function prototypes.

5. User Experience Design

- Simple **text-based interface** with clear menu options.
- Error messages for invalid inputs.
- Clean, readable bill format.
- Sequential bill IDs for clarity.

IMPLEMENTATION DETAILS

I) Structures Used

struct MenuItem { Untitled-1

```c
struct MenuItem {
    int id;
    char name[50];
    float price;
};

struct OrderItem {
    int id;
    char name[50];
    int quantity;
    float subTotal;
};
```

i.Menu Item stores details of each item available in the canteen.

ii. Order Item stores the details of each item ordered by the customer during billing

II) **Loading Menu**

Function: load Menu()

Role:

- Opens menu.txt
- Reads each item line-by-line
- Stores data in menu [] array
- Updates menu Count
- **Key Steps**
- f open ("menu.txt", "r") → open file for reading
- f scanf () → extract ID, name, & price
- Checks for file errors
- Closes file after reading.

III) **Displaying Menu**

Function: display Menu()

Role:

- Prints the menu in table format
- Helps user see items before ordering

Key Steps

- Checks if menu is empty
- Loops through menu []
- Prints ID, NAME, PRICE

IV) **Generating Bill**

Function: generate Bill()

Role:

- Core logic of the entire program
- Takes orders from the user
- Calculates subtotal & total
- Prints receipt
- Saves bill to *history.txt*

Key Steps

- Displays menu
- Accepts input: Item ID & Quantity
- Validates ID & quantity

- Finds item using linear search
- Stores results in currentOrder[]
- Calculates subtotal → adds to final total
- Writes final bill with date/time to file

V) Saving Bill to History

Function: save Bill() / inside generate Bill

Role:

- Opens *history.txt* in append mode
- Writes:
- Bill ID
- Date
- Time
- Total amount
- **Key Points**
- Uses f printf() for formatted writing
- Ensures previous bills are not erased ("a" mode)

VI) Viewing Bill History

Function: view History()

Role:

- Reads *history.txt*
- Displays past bills to user
- Finds the last used Bill ID for continuous numbering

Key Steps

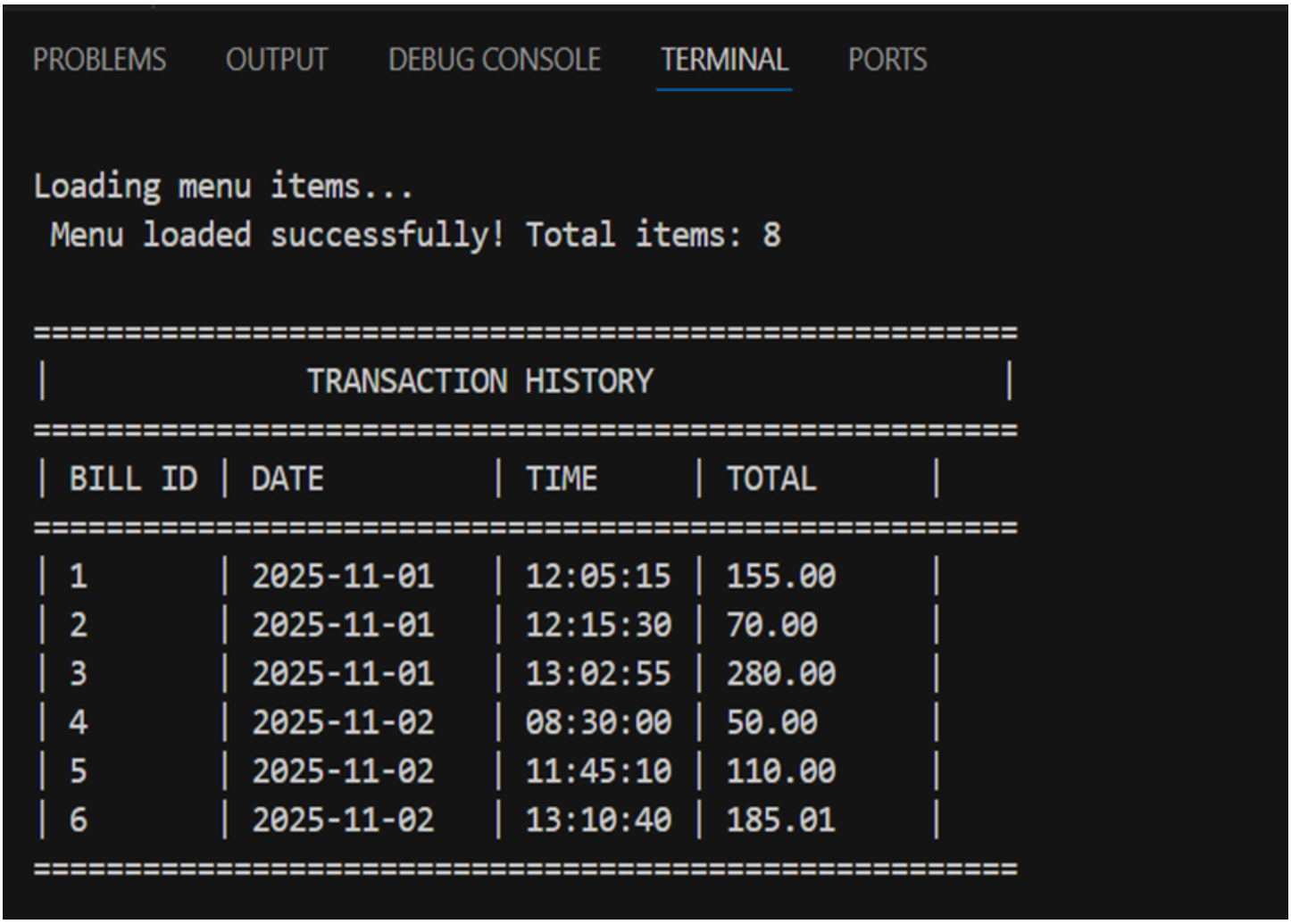
- Opens file in "r" mode
- Uses f scanf() to read each record
- Updates last Bill Number

VII) Input Validation & Error Handling

Included in all major functions

- **Checks invalid menu file**
- **Validates numeric input using scanf()**
- **Clears input buffer after failed input**
- **Prevents negative quantities**
- **Checks array limits (MAX_ITEMS, MAX_ORDER_ITEMS)**

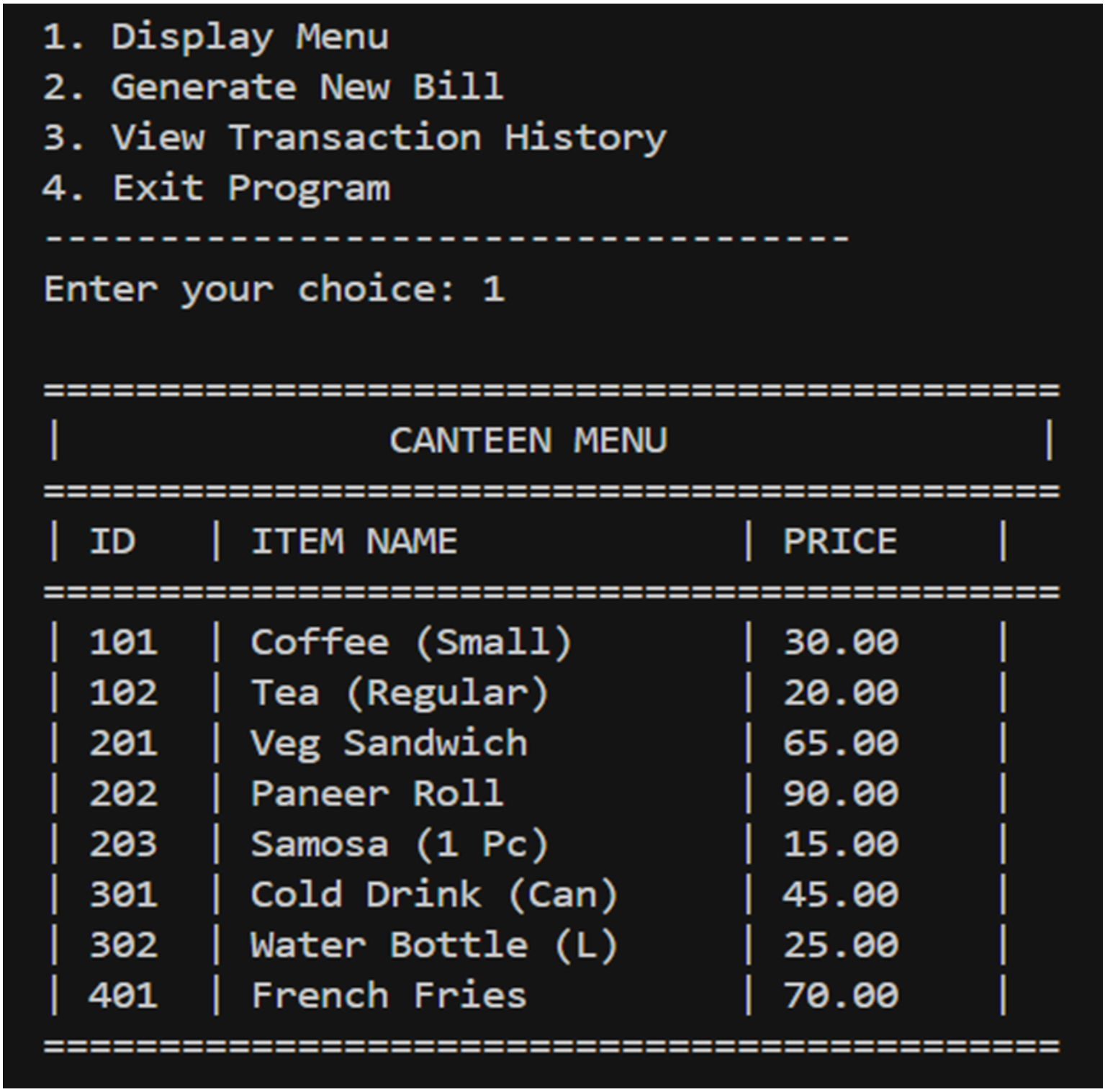
TESTING & RESULTS



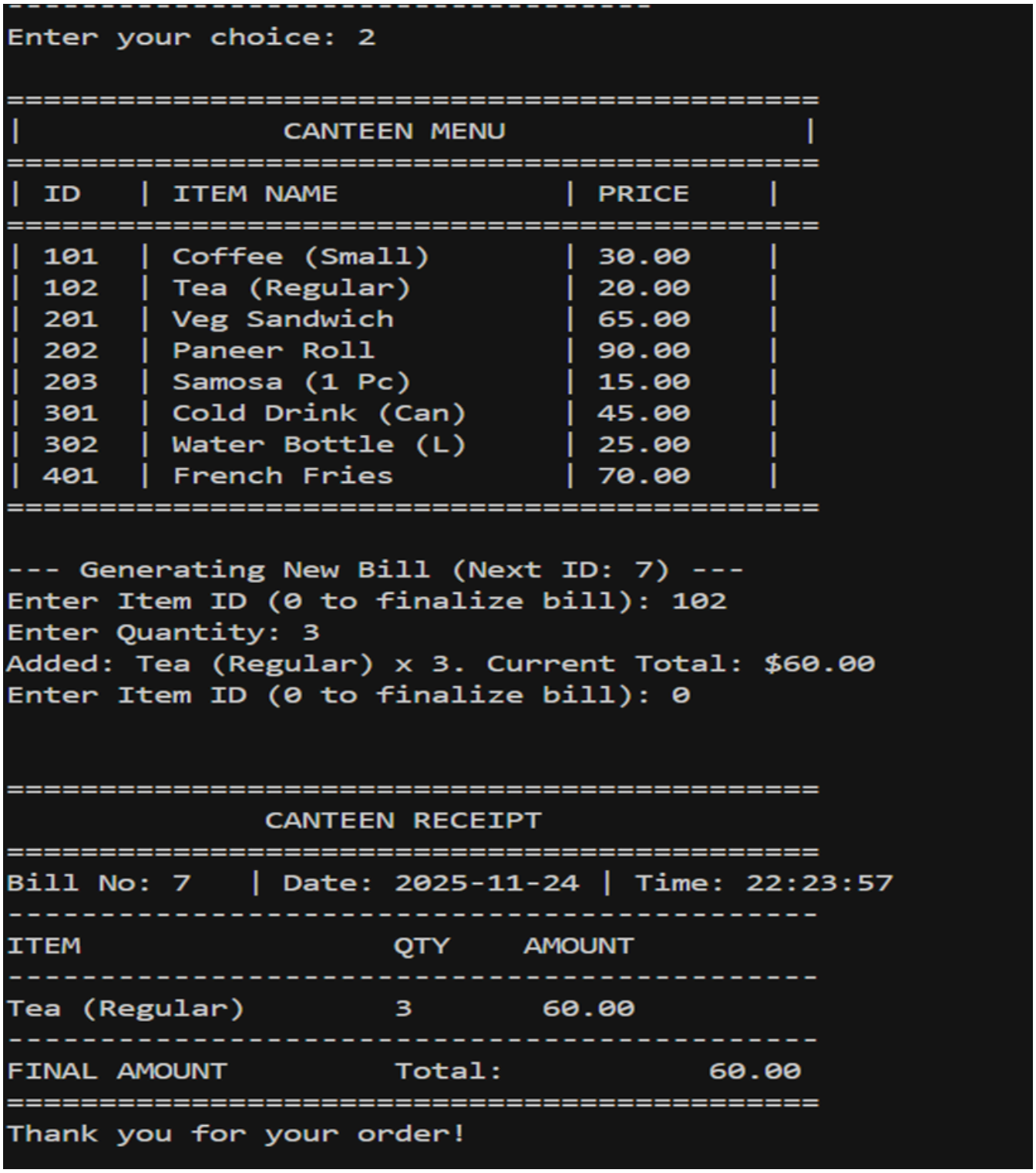
i) We have run the code using:

1. gcc *.c -o canteen.exe

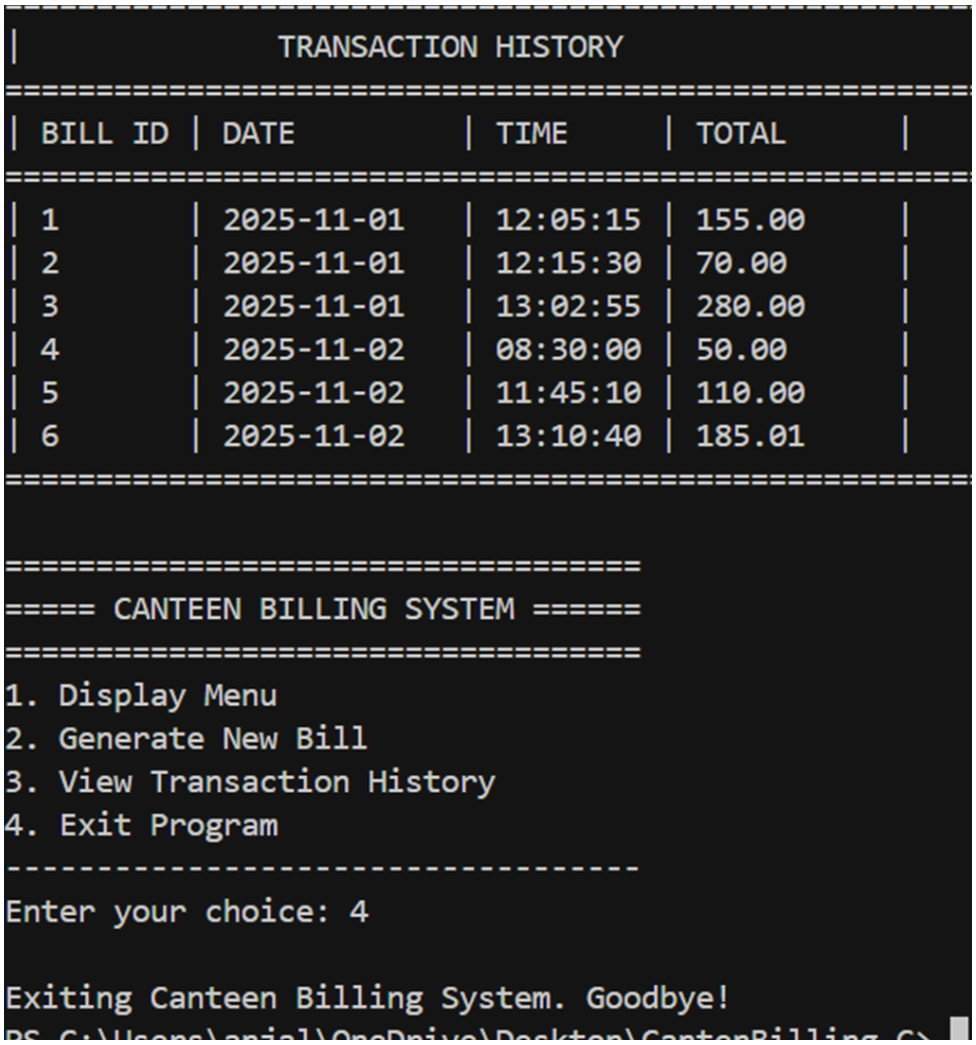
2. ./canteen.exe



ii) The user has selected the display menu option so the menu has been displayed for the user to choose from.



iii) The system displays the full canteen menu, allows the user to enter item IDs and quantities, and automatically calculates the total. After confirming the order, it generates a formatted receipt showing the bill number, date, time, purchased items, and final amount.



iv) This section displays all previously generated bills by reading them from *history.txt*. Each record shows the Bill ID, Date, Time, and Total amount, helping the user review past transactions easily. It demonstrates successful file reading, formatting, and tabular display in the system.

CONCLUSION

The Canteen Billing System successfully automates basic billing operations such as displaying the menu, taking customer orders, generating bills, and maintaining transaction history. By using structures, file handling, and modular functions, the system ensures accurate calculations and organized record-keeping. Overall, the project provides a simple, reliable, and efficient solution for managing canteen transactions.

FUTURE WORK

1. Implement **item editing and dynamic menu updates** instead of static menu files.

2. Introduce **search and filter options** in transaction history.

3. Upgrade the UI with **graphical menus** or a web-based interface.

APPENDIX

```c
// functions.c

#include "header.h"

MenuItem menu[MAX_ITEMS];
int menuCount = 0;
int lastBillNumber = 0; // Tracks the last Bill ID used for continuity

/*
Helper function to get the current date and time.
 */
void getCurrentDateTime(char *dateStr, char *timeStr) {
    time_t timer;
    struct tm* tm_info;

    time(&timer);
    tm_info = localtime(&timer);

    // Format date as YYYY-MM-DD
    strftime(dateStr, 11, "%Y-%m-%d", tm_info);
    // Format time as HH:MM:SS
    strftime(timeStr, 9, "%H:%M:%S", tm_info);
}

/**
 * Reads menu items from the 'menu.txt' file.
 */
void loadMenu() {
    FILE *fp = fopen("menu.txt", "r");

    if (fp == NULL) {
        printf("\n  ERROR: 'menu.txt' file not found. Ensure the file exists.\n");
        exit(1);
    }
```

```c
    printf("Loading menu items...\n");

    while (menuCount < MAX_ITEMS &&
           fscanf(fp, "%d,%49[^,],%f\n",
                  &menu[menuCount].id,
                  menu[menuCount].name,
                  &menu[menuCount].price) == 3) {
        menuCount++;
    }

    fclose(fp);
    printf(" Menu loaded successfully! Total items: %d\n", menuCount);
}

/**
 * Displays the loaded menu items to the user.
 */
void displayMenu() {
    if (menuCount == 0) {
        printf("\nMenu is empty. Load failed or file is blank.\n");
        return;
    }

    printf("\n============================================\n");
    printf("|              CANTEEN MENU                |\n");
    printf("============================================\n");
    printf("| %-4s | %-20s | %-8s |\n", "ID", "ITEM NAME", "PRICE");
    printf("============================================\n");
```

```
    for (int i = 0; i < menuCount; i++) {
        printf("| %-4d | %-20s | %-8.2f |\n",
               menu[i].id,
               menu[i].name,
               menu[i].price);
    }
    printf("=============================================\n");
}


/**
 * Core logic for taking orders, calculating the total, printing the receipt,
 * and saving the transaction to history.txt.
 */
void generateBill() {
    if (menuCount == 0) {
        printf("\nCannot generate bill: Menu is empty. Please load menu first.\n");
        return;
    }

    // Prepare bill variables
    OrderItem currentOrder[MAX_ORDER_ITEMS];
    int currentItemCount = 0;
    float finalTotal = 0.0;
    int itemID, quantity;
    int foundIndex = -1;

    displayMenu();
    printf("\n--- Generating New Bill (Next ID: %d) ---\n", lastBillNumber + 1);

```

```c
    // Order input loop
    do {
        printf("Enter Item ID (0 to finalize bill): ");
        if (scanf("%d", &itemID) != 1) {
            printf("Invalid input. Please enter a number.\n");
            while (getchar() != '\n'); // Clear buffer
            continue;
        }

        if (itemID == 0)
        break; // Exit loop

        printf("Enter Quantity: ");
        if (scanf("%d", &quantity) != 1 || quantity <= 0) {
            printf("Invalid quantity. Must be a positive number.\n");
            while (getchar() != '\n');
            continue;
        }

        // Search for the item in the loaded menu
        foundIndex = -1;
        for (int i = 0; i < menuCount; i++) {
            if (menu[i].id == itemID) {
                foundIndex = i;
                break;
            }
        }
```

```
123         if (foundIndex != -1) {
124             if (currentItemCount >= MAX_ORDER_ITEMS) {
125                 printf("Cannot add more items. Bill limit reached.\n");
126                 break;
127             }
128 
129             // Calculate and record the item
130             float price = menu[foundIndex].price;
131             float subTotal = price * quantity;
132             finalTotal += subTotal;
133 
134             // Save details to the current order array
135             currentOrder[currentItemCount].itemId = itemID;
136             strcpy(currentOrder[currentItemCount].name, menu[foundIndex].name);
137             currentOrder[currentItemCount].quantity = quantity;
138             currentOrder[currentItemCount].subTotal = subTotal;
139             currentItemCount++;
140 
141             printf("Added: %s x %d. Current Total: $%.2f\n",
142                    menu[foundIndex].name, quantity, finalTotal);
143 
144         } else {
145             printf(" Error: Item ID %d not found in menu.\n", itemID);
146         }
147 
148     } while (1); // Loop until 0 is entered
149 
150     // Finalize and print receipt if items were ordered
```

```
151     if (finalTotal > 0.0) {
152         lastBillNumber++;
153         char dateStr[11], timeStr[9];
154         getCurrentDateTime(dateStr, timeStr);
155 
156         printf("\n\n==========================================\n");
157         printf("              CANTEEN RECEIPT              \n");
158         printf("==========================================\n");
159         printf("Bill No: %-3d | Date: %s | Time: %s\n", lastBillNumber, dateStr, timeStr);
160         printf("------------------------------------------\n");
161         printf("%-20s %-4s %8s\n", "ITEM", "QTY", "AMOUNT");
162         printf("------------------------------------------\n");
163 
164         for (int i = 0; i < currentItemCount; i++) {
165             printf("%-20s %-4d %8.2f\n",
166                    currentOrder[i].name,
167                    currentOrder[i].quantity,
168                    currentOrder[i].subTotal);
169         }
170 
171         printf("------------------------------------------\n");
172         printf("%-20s Total: %15.2f\n", "FINAL AMOUNT", finalTotal);
173         printf("==========================================\n");
174         printf("Thank you for your order!\n");
175 
176         // Save bill to history.txt
177         FILE *fp_history = fopen("history.txt", "a");
178         if (fp_history != NULL) {
179             // Format: BillID,Date,Time,TotalAmount
180             fprintf(fp_history, "%d,%s,%s,%.2f\n",
181                     lastBillNumber, dateStr, timeStr, finalTotal);
182             fclose(fp_history);
```

```
            printf("\n Bill saved to history.txt.\n");
        } else {
            printf("\n WARNING: Could not save bill to history.txt.\n");
        }

    } else {
        printf("\nBill cancelled. No items ordered.\n");
    }
}

/**
 * Reads and displays the transaction history from 'history.txt'.
 */
void viewHistory() {
    FILE *fp = fopen("history.txt", "r");
    if (fp == NULL) {
        printf("\nHistory file 'history.txt' not found or empty.\n");
        return;
    }

    int billID;
    char date[11], time[9];
    float total;
    int recordCount = 0;

    printf("\n=========================================================\n");
    printf("|              TRANSACTION HISTORY                      |\n");
    printf("=========================================================\n");
    printf("| %-7s | %-12s | %-8s | %-10s |\n",
           "BILL ID", "DATE", "TIME", "TOTAL");
```

```
213     printf("=========================================================\n");
214
215     // Loop to read data until End Of File
216     while (fscanf(fp, "%d,%10[^,],%8[^,],%f\n",
217            &billID, date, time, &total) == 4) {
218
219         printf("| %-7d | %-12s | %-8s | %-10.2f |\n",
220                billID, date, time, total);
221         recordCount++;
222         // This updates lastBillNumber based on file contents
223         if (billID > lastBillNumber) {
224             lastBillNumber = billID;
225         }
226     }
227
228     if (recordCount == 0 && lastBillNumber == 0) {
229         printf("| No records found in history.txt.                    |\n");
230     }
231
232     fclose(fp);
233     printf("=========================================================\n");
234 }
```

REFRENCES

1.Let Us C – Yashwant Kanetkar

2. Class Notes

3.GitHub Open-Source Examples – Simple C Projects using File Handling.