



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
---------------	---------

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

```
C struct MenuItem { Untitled-1 ●
1 struct MenuItem {
2     int id;
3     char name[50];
4     float price;
5 };
6
7 struct OrderItem {
8     int id;
9     char name[50];
10    int quantity;
11    float subTotal;
12 };
13
```

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

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Loading menu items...
Menu loaded successfully! Total items: 8

=====
|                TRANSACTION HISTORY                |
=====
| BILL ID | DATE          | TIME          | TOTAL          |
=====
| 1        | 2025-11-01    | 12:05:15     | 155.00         |
| 2        | 2025-11-01    | 12:15:30     | 70.00          |
| 3        | 2025-11-01    | 13:02:55     | 280.00         |
| 4        | 2025-11-02    | 08:30:00     | 50.00          |
| 5        | 2025-11-02    | 11:45:10     | 110.00         |
| 6        | 2025-11-02    | 13:10:40     | 185.01         |
=====
```

i) We have run the code using:

1. gcc *.c -o canteen.exe

2. ./canteen.exe

1. Display Menu
2. Generate New Bill
3. View Transaction History
4. Exit Program

Enter your choice: 1

=====

| CANTEEN MENU |

=====

ID	ITEM NAME	PRICE
101	Coffee (Small)	30.00
102	Tea (Regular)	20.00
201	Veg Sandwich	65.00
202	Paneer Roll	90.00
203	Samosa (1 Pc)	15.00
301	Cold Drink (Can)	45.00
302	Water Bottle (L)	25.00
401	French Fries	70.00

=====

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

Enter your choice: 2

CANTEEN MENU			
ID	ITEM NAME	PRICE	
101	Coffee (Small)	30.00	
102	Tea (Regular)	20.00	
201	Veg Sandwich	65.00	
202	Paneer Roll	90.00	
203	Samosa (1 Pc)	15.00	
301	Cold Drink (Can)	45.00	
302	Water Bottle (L)	25.00	
401	French Fries	70.00	

--- Generating New Bill (Next ID: 7) ---

Enter Item ID (0 to finalize bill): 102

Enter Quantity: 3

Added: Tea (Regular) x 3. Current Total: \$60.00

Enter Item ID (0 to finalize bill): 0

CANTEEN RECEIPT		
Bill No: 7	Date: 2025-11-24	Time: 22:23:57
ITEM	QTY	AMOUNT
Tea (Regular)	3	60.00
FINAL AMOUNT	Total:	60.00
Thank you for your order!		

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.

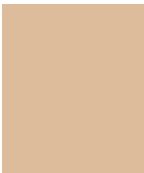
```
|
| TRANSACTION HISTORY
|=====
| BILL ID | DATE           | TIME       | TOTAL      |
|=====
| 1       | 2025-11-01    | 12:05:15  | 155.00     |
| 2       | 2025-11-01    | 12:15:30  | 70.00      |
| 3       | 2025-11-01    | 13:02:55  | 280.00     |
| 4       | 2025-11-02    | 08:30:00  | 50.00      |
| 5       | 2025-11-02    | 11:45:10  | 110.00     |
| 6       | 2025-11-02    | 13:10:40  | 185.01     |
|=====

=====
===== CANTEEN BILLING SYSTEM =====
=====

1. Display Menu
2. Generate New Bill
3. View Transaction History
4. Exit Program
-----
Enter your choice: 4

Exiting Canteen Billing System. Goodbye!
PS C:\Users\anil\OneDrive\Desktop\CanteenBilling>
```

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

```
1 // FUNCTIONS.C
2
3 #include "header.h"
4
5 MenuItem menu[MAX_ITEMS];
6 int menuCount = 0;
7 int lastBillNumber = 0; // Tracks the last Bill ID used for continuity
8
9 /*
10  Helper function to get the current date and time.
11  */
12 void getCurrentDateTime(char *dateStr, char *timeStr) {
13     time_t timer;
14     struct tm* tm_info;
15
16     time(&timer);
17     tm_info = localtime(&timer);
18
19     // Format date as YYYY-MM-DD
20     strftime(dateStr, 11, "%Y-%m-%d", tm_info);
21     // Format time as HH:MM:SS
22     strftime(timeStr, 9, "%H:%M:%S", tm_info);
23 }
24
25 /**
26  * Reads menu items from the 'menu.txt' file.
27  */
28 void loadMenu() {
29     FILE *fp = fopen("menu.txt", "r");
30
31     if (fp == NULL) {
32         printf("\n ERROR: 'menu.txt' file not found. Ensure the file exists.\n");
33         exit(1);
34     }
35 }
```

```

36     printf("Loading menu items...\n");
37
38     while (menuCount < MAX_ITEMS &&
39           fscanf(fp, "%d,%49[^\n],%f\n",
40                &menu[menuCount].id,
41                menu[menuCount].name,
42                &menu[menuCount].price) == 3) {
43         menuCount++;
44     }
45
46     fclose(fp);
47     printf(" Menu loaded successfully! Total items: %d\n", menuCount);
48 }
49
50 /**
51  * Displays the loaded menu items to the user.
52  */
53 void displayMenu() {
54     if (menuCount == 0) {
55         printf("\nMenu is empty. Load failed or file is blank.\n");
56         return;
57     }
58
59     printf("\n===== \n");
60     printf("|                CANTEEN MENU                | \n");
61     printf("===== \n");
62     printf("| %-4s | %-20s | %-8s | \n", "ID", "ITEM NAME", "PRICE");
63     printf("===== \n");

```

```

65     for (int i = 0; i < menuCount; i++) {
66         printf("| %-4d | %-20s | %-8.2f |\n",
67             menu[i].id,
68             menu[i].name,
69             menu[i].price);
70     }
71     printf("=====\n");
72 }
73
74
75 /**
76  * Core logic for taking orders, calculating the total, printing the receipt,
77  * and saving the transaction to history.txt.
78  */
79 void generateBill() {
80     if (menuCount == 0) {
81         printf("\nCannot generate bill: Menu is empty. Please load menu first.\n");
82         return;
83     }
84
85     // Prepare bill variables
86     OrderItem currentOrder[MAX_ORDER_ITEMS];
87     int currentItemCount = 0;
88     float finalTotal = 0.0;
89     int itemID, quantity;
90     int foundIndex = -1;
91
92     displayMenu();
93     printf("\n--- Generating New Bill (Next ID: %d) ---\n", lastBillNumber + 1);
94

```

```
95 // Order input loop
96 do {
97     printf("Enter Item ID (0 to finalize bill): ");
98     if (scanf("%d", &itemID) != 1) {
99         printf("Invalid input. Please enter a number.\n");
100         while (getchar() != '\n'); // Clear buffer
101         continue;
102     }
103
104     if (itemID == 0)
105         break; // Exit loop
106
107     printf("Enter Quantity: ");
108     if (scanf("%d", &quantity) != 1 || quantity <= 0) {
109         printf("Invalid quantity. Must be a positive number.\n");
110         while (getchar() != '\n');
111         continue;
112     }
113
114     // Search for the item in the loaded menu
115     foundIndex = -1;
116     for (int i = 0; i < menuCount; i++) {
117         if (menu[i].id == itemID) {
118             foundIndex = i;
119             break;
120         }
121     }
```

```
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);

```

```

183         printf("\n Bill saved to history.txt.\n");
184     } else {
185         printf("\n WARNING: Could not save bill to history.txt.\n");
186     }
187
188 } else {
189     printf("\nBill cancelled. No items ordered.\n");
190 }
191 }
192
193 /**
194  * Reads and displays the transaction history from 'history.txt'.
195  */
196 void viewHistory() {
197     FILE *fp = fopen("history.txt", "r");
198     if (fp == NULL) {
199         printf("\nHistory file 'history.txt' not found or empty.\n");
200         return;
201     }
202
203     int billID;
204     char date[11], time[9];
205     float total;
206     int recordCount = 0;
207
208     printf("\n===== \n");
209     printf("|          TRANSACTION HISTORY          | \n");
210     printf("===== \n");
211     printf("| %-7s | %-12s | %-8s | %-10s | \n",
212         "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 }

```

REFERENCES

1. Let Us C – Yashwant Kanetkar
 2. Class Notes
 3. GitHub Open-Source Examples – Simple C Projects using File Handling.
-