
PROJECT REPORT

Course Code: CSEG1041
Course Title: Programming in C
Project Title: Restaurant Management System
Student Name: Navya Tomar
Roll Number: 590028390
Semester: 1

1. Abstract

This project presents a simple Restaurant Billing and Menu Management System developed using the C programming language. The system reads menu items from a text file, allows users to select products by entering their IDs, stores selected items in a cart, and generates a final bill including GST. The project emphasizes modular C programming through the use of multiple source files, header files, structures, and file handling. The system is designed to be simple, efficient, and easy to use, providing practical exposure to real-world problem solving using C.

==========

2. Objective

The objective of this project is to design and implement a simple, modular, and user-friendly Restaurant Billing and Menu Management System using the C programming language. The system should automate menu loading, item selection, order storage, bill calculation, and invoice generation. The aim is to replace manual billing with a faster, more accurate, and systematic approach while reinforcing key programming concepts such as file handling, structures, arrays, functions, and multi-file program organization.

3. Introduction & Problem Statement

In many small restaurants or fast-food outlets, billing is done manually, which is error-prone and time-consuming.
Our goal was to build a simple, console-based billing system in C that automates billing, ensures accurate calculation, and produces a neatly formatted invoice.

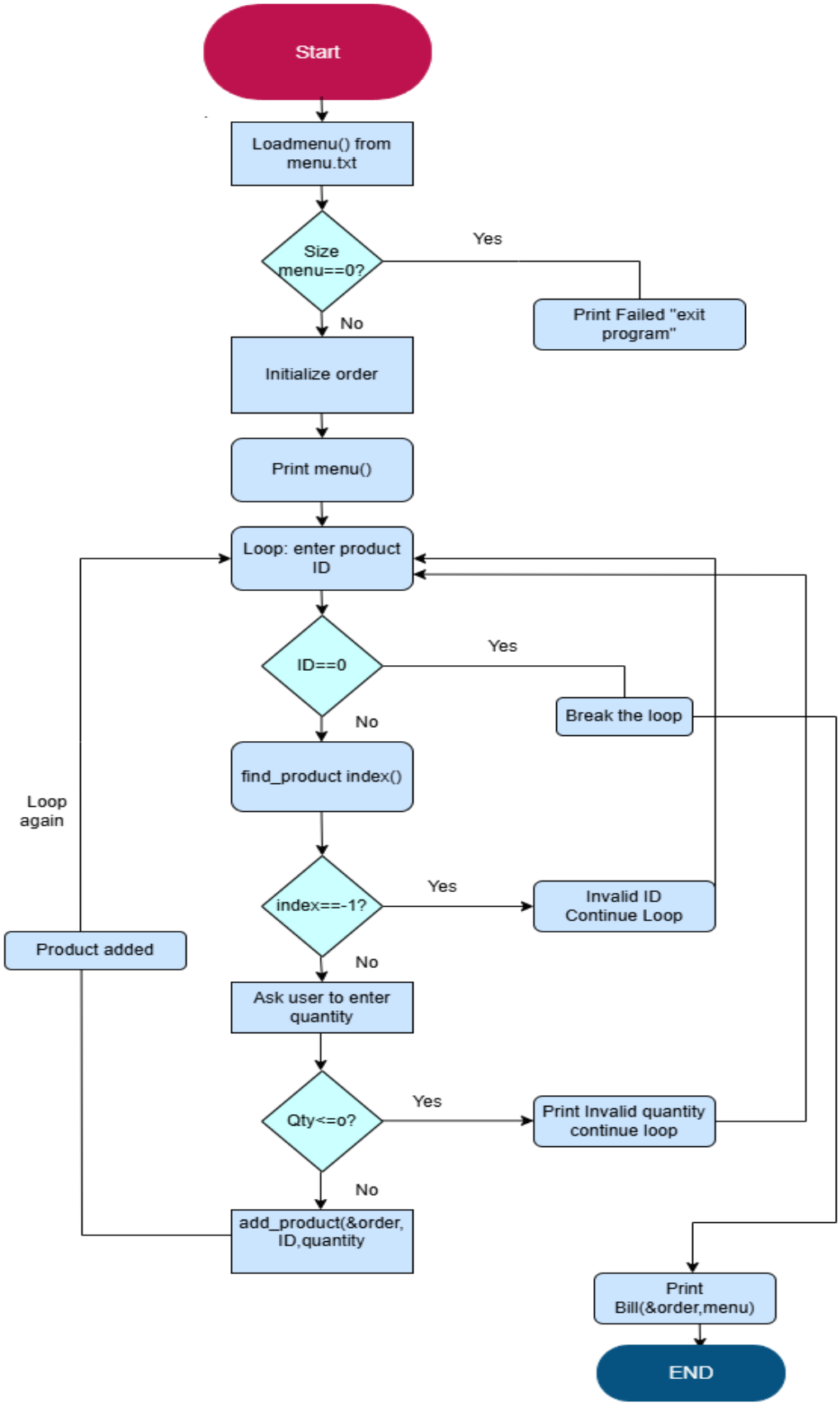
The system shall:

- Read menu products (ID, name, price) from a file (menu.txt)
- Display the menu to the user
- Allow selection of products by ID and specify quantity
- Maintain a cart (order) of selected products
- Calculate subtotal, apply GST (5%), and compute final amount
- Display a well-formatted invoice listing products, quantities, rates, subtotal, tax, and final payable amount

This project also serves as a practice in modular C programming: using multiple source and header files, file handling, structures, arrays, and user interaction.

==========

4. Flowchart


Start
Loadmenu() from menu.txt
Size menu==0?
Yes
Print Failed "exit program"
No
Initialize order
Print menu()
Loop: enter product ID
ID==0
Yes
Break the loop
No
find_product index()
Loop again
index==-1?
Yes
Invalid ID Continue Loop
Product added
No
Ask user to enter quantity
Qty<=o?
Yes
Print Invalid quantity continue loop
No
add_product(&order, ID,quantity
Print Bill(&order,menu)
END

```
-----------------------------------------
                START
-----------------------------------------------
Loadmenu() from menu.txt
-----------------------------------------------
Is size_menu == 0?
    ├── Yes → Print "Failed to load menu" → EXIT program
    └── No  → Continue
-----------------------------------------------
Initialize order
-----------------------------------------------
print_menu()
-----------------------------------------------
LOOP: Ask user to enter Product ID
-----------------------------------------------
Is ID == 0 ?
    ├── Yes → Break the loop
    └── No  → Continue
-----------------------------------------------
find_product_index()
-----------------------------------------------
Is index == -1 ?
    ├── Yes → Print "Invalid ID" → Continue loop
    └── No  → Product exists
-----------------------------------------------
Ask user to enter quantity
-----------------------------------------------
Is quantity <= 0 ?
    ├── Yes → Print "Invalid quantity" → Continue loop
    └── No  → Continue
-----------------------------------------------
add_product(&order, ID, quantity)
→ Product added to order
-----------------------------------------------
Loop continues until user enters 0
-----------------------------------------------
print_bill(&order, menu)
-----------------------------------------------
                    END
-----------------------------------------------
=========================================
```

5. Algorithms

Load Menu
1. Open “menu.txt” in read mode.
2. For each line, read: product ID (int), name (string), price (float).
3. Store in array of structures.
4. On EOF or error, close file and return count of products.

Add products to Cart
1. Prompt user for product ID.
2. If ID = 0 → finish selection.
3. Else search menu array for matching ID.
4. If found → prompt for quantity → store ID and quantity in order/cart structure.
5. Repeat.

Print Bill
1. For each entry in cart: find menu product by ID.
2. Compute amount = price × quantity; accumulate subtotal.
3. After all products, compute GST = 5% of subtotal.
4. Final amount = subtotal + GST.
5. Print a formatted bill listing: product name, qty, rate, amount; then subtotal, GST, final amount.

==========

KEY FEATURES USED
- **Structures** to represent product and order details.
- **File Handling** functions (`fopen`,`fscanf`,`fclose`) to store and retrieve data.
- **User-Defined Functions** for modularity, easier readability and clarity.

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

typedef struct {
    int productId[MAX_ORDER_PRODUCTS];
    int qty[MAX_ORDER_PRODUCTS];
    int count;
} Order;

FILE *fptr = fopen(filename, "r");
    if (!fptr) {
        printf("Error in opening the file %s\n", filename);
        return 0;
    }
 int  count= 0;
    while (fscanf(fptr, "%d %s %f",
                  &menu[count].id,
                  menu[count].name,
                  &menu[count].price) == 3) {
        count++;
    }
    fclose(fptr);
```

6.Problems Faced

-Understanding multi-file compilation was challenging. Managing separate
 .c and .h files and ensuring proper inclusion required repeated debugging.

-File handling needed careful attention, especially ensuring that menu.txt followed
 the correct format. Any spacing or formatting mistake caused incorrect menu loading.

-Product names in the menu had to be single-word due to fscanf("%s").
 Handling multi-word items required a different approach, which added
 limitations during development.

-Formatting the final bill neatly was challenging at first, since
 alignment required proper use of printf() format specifiers.
==========

7.Future Scope

-Let users modify or delete items already added to the cart before generating the bill.

-Store Bills in a file for record-keeping and daily sales tracking.

-Admin Mode for Menu Editing.Password-protected admin access to add, delete, or update menu items directly.

-Export Bill to File Formats.Generate receipts in text, CSV, or PDF format.

-Database Integration (Long-Term).Replace text files with SQLite or MySQL for scalable storage.

==========

8. References

-Course Lecture Notes & Class Material. Guided the structure of the project and requirements like multi-file programming

-E.Balagurusamy – Programming in ANSI C

-Instructor-Provided Guidelines-Used to standardize folder structure, coding style, and project formatting.

-Online C Programming Resources (GeeksforGeeks, Tutorialspoint)
Helped in understanding examples of file handling, formatted output, and modular C program design.

==========

9. Conclusion

The Restaurant Billing and Menu Management System successfully fulfills the objective of automating the billing process using the C programming language. It demonstrates the use of structures, file handling, modular functions, and multi-file program organization in an efficient and practical way.

The system accurately loads menu items, accepts user input, validates product IDs, calculates totals and GST, and generates a clear and readable invoice, proving its usefulness for small restaurant setups.

The project also strengthened understanding of core C concepts and helped build problem-solving, debugging, and modular design skills. Overall, the implementation meets all requirements and provides a strong foundation for further enhancements.

==========

==========

main.c

```
#include <stdio.h>
#include "menu.h"
#include "order.h"
#include "util.h"

int main() {
    Product menu[MAX_PRODUCTS];   //menu is an array instance of structure-
Product.
    Order order;            //order is an instance of structure-Order.
    int size_menu;           //Stores the number of products in the menu.

    size_menu = update_menu("menu.txt", menu);    //Get menu from menu.txt file
and get the number of products in it.

    if (size_menu == 0) {
        printf("Failed to load the menu\n");
        return 0;
    }

    initialize(&order);                  //Initialize the order with 0 products.
```

```
 print_menu(menu, size_menu);              //Display the menu on the screen.
    while (1) {
        int id = getInteger("\nEnter Product ID to add (0 to finish): ");
//Read product ID from user.

        if (id == 0)
            break;

        int idx = find_product_index(menu, size_menu, id);      //Find the index
of the product in the menu array.
        if (idx == -1) {
            printf("Invalid ID!\n");
            continue;
        }

        int qty = getInteger("Enter quantity: ");      //Get quantity from user.

        if (qty <= 0) {
            printf("Invalid quantity!\n");
            continue;
        }

        add_product(&order, id, qty);                //Add the product to the
order using ID and quantity.
        printf("Product added!\n");
    }
    print_bill(&order, menu, size_menu);             //Print the final bill for
the order.
    return 0;
}
```

==========

menu.c

```
#include <stdio.h>
#include <string.h>
#include "menu.h"

int update_menu(const char *filename, Product menu[]) {      //Function to get
the menu from menu.txt file
    FILE *fptr = fopen(filename, "r");
    if (!fptr) {
        printf("Error in opening the file %s\n", filename);
        return 0;
    }

    int  count= 0;
   // The loop below reads each line from the file and puts them in the menu
array of structure.
    while (fscanf(fptr, "%d %s %f",
                  &menu[count].id,
                  menu[count].name,
                  &menu[count].price) == 3) {
        count++;
    }

    fclose(fptr);
    return count;
}

int find_product_index(Product menu[], int size_menu, int id) {   //Function to
return the index of a product in the menu based on its ID
    for (int i = 0; i < size_menu; i++) {
        if (menu[i].id == id)
```

```
            return i;
    }
    return -1;
}

void print_menu(Product menu[], int size_menu) {     //Function to display the
menu on screen
    printf("\n========= RESTAURANT MENU =========\n");
    printf("ID\tProduct\t\tPrice\n");
    printf("-----------------------------------------------------\n");

    for (int i = 0; i < size_menu; i++) {
        printf("%d\t%-15s\t%.2f\n", menu[i].id, menu[i].name, menu[i].price);
    }
    printf("-----------------------------------------------------\n");
}
```

order.c

```
#include <stdio.h>
#include "order.h"

void initialize(Order *o) {    //Function to initialize the number of products
in the order to zero
    o->count = 0;
}

int add_product(Order *o, int id, int qty) {   // Function to add an product in
the order
    if (o->count >= MAX_ORDER_PRODUCTS)
        return -1;

    o->productId[o->count] = id;
    o->qty[o->count] = qty;
    o->count++;

    return 0;
}

void print_bill(Order *o,Product menu[], int size_menu) {    //Function to print
the bill of the order
    float total = 0;

    printf("\n============= ORDER SUMMARY =============\n");
    printf("Product\t\tQty\tPrice\tSubtotal\n");
    printf("---------------------------------------------------------------\n");
   // The loop below iterates through the the products in order(structure) and
calculates the subtotal.
    for (int i = 0; i < o->count; i++) {
        int idx = find_product_index(menu, size_menu, o->productId[i]);
        float price = menu[idx].price;
        float sub = price * o->qty[i];

        total += sub;

        printf("%-15s\t%d\t%.2f\t%.2f\n",
               menu[idx].name,
               o->qty[i],
               price,
               sub);
    }

    float gst = total * 0.05;
    float final = total + gst;
```

```
    printf("-----------------------------------------------------------------\n");
    printf("Total:\t\t\t\t%.2f\n", total);
    printf("GST (5%%):\t\t\t%.2f\n", gst);
    printf("Final Amount:\t\t\t%.2f\n", final);
    printf("=========================================\n");
}
```

==========

util.c

```
#include <stdio.h>
#include "util.h"

int getInteger(const char *prompt) {       //Function to read an integer from
user with a prompt message
    int a;
    printf("%s", prompt);
    scanf("%d", &a);
    return a;
}
```

==========

Header files

==========

menu.h

```
#ifndef MENU_H
#define MENU_H

#define MAX_PRODUCTS 50       //Maximum number of products in the menu

typedef struct {
    int id;                  //Stores Product ID
    char name[50];           //Stores Name of Product
    float price;             //Stores Price of Product
} Product;

int update_menu(const char *filename, Product menu[]);
void print_menu(Product menu[], int n);
int find_product_index(Product menu[], int size_menu, int id);

#endif
```

==========

order.h

```
#ifndef ORDER_H
#define ORDER_H

#include "menu.h"

#define MAX_ORDER_PRODUCTS 20        //Maximum number of products in one single
order

typedef struct {
    int productId[MAX_ORDER_PRODUCTS];  //Array to store product IDs
    int qty[MAX_ORDER_PRODUCTS];     //Array to store quantities of every
product
    int count;                     //Number of products in the order
} Order;

void initialize(Order *o);
int add_product(Order *o, int id, int qty);
void print_bill(Order *o, Product menu[], int size_menu);
```

--

```
#endif
```

```
util.h

#ifndef UTIL_H
#define UTIL_H

int getInteger(const char *prompt);

#endif
```

```
# PROJECT-RESTAURANT_MANAGEMENT_SYSTEM
```

◘ Features

Loads menu items from an external file (menu.txt)
Displays menu in a formatted layout
Allows users to enter product ID and quantity
Validates input and product IDs
Adds items to cart using structured data
Calculates:
Item-wise subtotal
Total bill
GST at 5%
Final payable amount
Generates a clean and readable invoice

◘ Technologies & Concepts Used

C Programming
Structures
Multi-file programming (.c + .h)
File handling (fopen, fscanf, fclose)
Modular functions
Arrays & loops
Formatted output (printf)

◘ How to Run the Project

1. Compile the Code
Use GCC or any C compiler:

```
gcc src/main.c src/menu.c src/order.c src/util.c -I include -o main
```

2. Run the Executable

```
./main
```

3. Ensure menu.txt Exists
Example format:
<pre>
menu.txt
1 PIZZA 120
2 FRIES 75
3 BURGER 80
4 COKE 40
5 PASTA 140
6 DABELI 35
7 BIRYANI 110
</pre>

◘ Sample Output (Invoice)

<pre>
========== ORDER SUMMARY ==========
Product   Qty   Price   Subtotal

PIZZA   1   120.00   120.00
BURGER   1   80.00   80.00

```
Total:                              200.00
GST (5%):                            10.00
Final Amount:                       210.00
=========================================
</pre>
```