Supermarket Billing System (Using Stacks)

# INDEX

**1. Introduction**

**2. Objectives**

**3. Tools & Technologies Used**

**4. System Requirements**

i) Hardware Requirements

ii) Software Requirements

**5. System Design & Implementation**

5.1 Functional Components

5.2 Data Structure Used

5.3 Algorithm

**6. Code Implementation**

**7. Testing & Validation**

**8. Results & Discussion**

**9. Future Enhancements**

**10. Conclusion**

**11. References**

**1.Introduction**

This project presents a simple yet effective Supermarket Billing System developed using C programming and the stack data structure. The system facilitates item entry, calculates total price dynamically, and generates a final bill. It is a console-based application mainly designed for educational purposes to demonstrate stack operations.

# 2. Objectives

- To implement a billing system using Stacks.

- To apply LIFO (Last-In-First-Out) principles in a real-world application.

- To practice and understand data structure implementation in C.

- To generate a user-friendly and efficient billing interface.

# 3. Tools & Technologies Used

- Programming Language: C

- Development Environment: Gdb compiler, Code::Blocks / Turbo C / VS Code

- Data Structure: Stack (Array Implementation)

# 4. System Requirements

i) Hardware Requirements

- Processor: Intel i3 or higher

- RAM: 2 GB or more

- Hard Disk: Minimum 100 MB of free space

- Input Device: Keyboard

ii) Software Requirements

- Operating System: Windows/Linux

- IDE: Code::Blocks or VS Code

- Compiler: GCC (MinGW on Windows)

# 5. System Design & Implementation

5.1 Functional Components

- Add item to the bill

- Calculate total price of each item

- Print all items in reverse order (LIFO)

- Display grand total

5.2 Data Structure Used

- Stack: Implemented using arrays to store item name, quantity, price, and total price.

5.3 Algorithm

1. Initialize top = -1

2. When user selects Add Item:

- Increment top

- Input item name, quantity, and price

- Calculate total and store all values at position top

3. When user selects Print Bill:

- Loop through stack from top to 0

- Print each item and calculate grand total

- Decrement top in each iteration (simulate pop)

# 6. Code Implementation

#include <stdio.h>

#include <string.h>

#define MAX 100

int main() {

char item\_name[MAX][50];

int quantity[MAX];

float price[MAX];

float total\_price[MAX];

int top = -1;

int choice;

float grand\_total = 0;

printf("=== Supermarket Billing System (Using Stack) ===\n");

while (1) {

printf("\n1. Add Item\n2. Print Bill and Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

if (choice == 1) {

if (top >= MAX - 1) {

printf("Stack Overflow! Cannot add more items.\n");

continue;

}

top++;

printf("Enter item name: ");

scanf("%s", item\_name[top]);

printf("Enter quantity: ");

scanf("%d", &quantity[top]);

printf("Enter price per item: ");

scanf("%f", &price[top]);

total\_price[top] = quantity[top] \* price[top];

printf("Item added: %s | Qty: %d | Price: %.2f | Total: %.2f\n",

item\_name[top], quantity[top], price[top], total\_price[top]);

}

else if (choice == 2) {

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

printf("%-20s %-10s %-10s %-10s\n", "Item", "Qty", "Price", "Total");

while (top >= 0) {

printf("%-20s %-10d %-10.2f %-10.2f\n",

item\_name[top], quantity[top], price[top], total\_price[top]);

grand\_total += total\_price[top];

top--;

}

printf("\nGrand Total: %.2f\n", grand\_total);

printf("Thank you for shopping!\n");

break;

}

else {

printf("Invalid choice. Try again.\n");

}

}

return 0;

}

# 7. Testing & Validation

The system was tested with various combinations of item entries. Edge cases like:

- Adding maximum items (stack overflow)

- Zero quantity or negative price entries

- Exiting without any item added

All were tested to ensure correct behavior. The LIFO structure correctly reversed item order at billing.

# 8. Results & Discussion

The application worked as intended. Items were successfully added and stored in the stack. Final billing displayed items in reverse order (latest item first). Grand total was accurately calculated and printed.

# 9. Future Enhancements

- Introduce file handling for saving bills.

- Add GUI using C graphics or integrate with web frontend.

- Implement discount and tax calculations.

- Allow item deletion or editing before final billing.

- Use linked list stack implementation for dynamic memory use.

# 10. Conclusion

This project demonstrates how basic data structures like stacks can be used to model real-world applications. Through this implementation, learners can understand stack operations more effectively and appreciate their practical utility in simple billing systems.

# 11. References

- Let Us C by Yashavant Kanetkar

- Data Structures Using C by Reema Thareja

- www.geeksforgeeks.org

- Lecture notes and online tutorials