CHANDIGARH UNIVERSITY
Discover. Learn. Empower.

UNIVERSITY INSTITUTE OF COMPUTING

Project Report

ON

Supermarket Billing System (using Stacks)

Program Name: BCA

Subject Name/Code: Data Structures Lab(24CAP-152)

Submitted by:

Name: Gurjot Singh, Ayush Kumar, Mohita

UID: 24BCA10298, 24BCA10292, 24BCA10307

Section: 24BCA6

Group: A

Submitted to:

Name: Mr. Dharam Pal Singh

Designation: Associate Professor

Contents

1. **Introduction** 3
2. **Objectives** 3
3. **Tools & Technologies Used** 3
4. **System Requirements** 3
   4.1 Hardware Requirements 3
   4.2 Software Requirements 3
5. **System Design & Implementation** 4
   5.1 Functional Components 4
   5.2 Data Structure Used 4
   5.3 Algorithm 4
6. **Code Implementation** 4
7. **Testing & Validation** 8
8. **Results & Discussion** 8
9. **Future Enhancements** 8
10. **Conclusion** 8
11. **References** 8

1. Introduction

The Supermarket Billing System is a simple console-based application that simulates the process of adding items to a cart, calculating totals, and printing bills. This mini project focuses on implementing the billing mechanism using the **stack data structure**, following a Last-In-First-Out (LIFO) approach.

---

2. Objectives

- To implement a billing system using stacks.
- To simulate real-time addition and removal of items from a cart.
- To understand stack operations in C through a practical application.
- To develop problem-solving skills with data structures.

---

3. Tools & Technologies Used

- Programming Language: C
- Compiler: GCC / Turbo C
- Data Structure: Stack (Array implementation)

---

4. System Requirements

4.1 Hardware Requirements

- Processor: Intel i3 or above
- RAM: 2 GB or more
- Storage: 100 MB free space
- Display: Any standard monitor

4.2 Software Requirements

- Operating System: Windows/Linux/macOS
- Compiler: GCC / Turbo C
- Editor: VS Code

---

5. System Design & Implementation

5.1 Functional Components

- **Add Item:** Pushes item into the stack.
- **Remove Item:** Pops the last added item.
- **Display Cart:** Traverses the stack to display items.
- **Total Amount:** Calculates the sum of all item prices.
- **Exit:** Terminates the program.

5.2 Data Structure Used

- **Stack** is used to simulate the cart.
- Implemented using arrays.
- Each item includes name, quantity, and price.

5.3 Algorithm

1. Initialize top = -1 (empty cart).
2. For push (add item):
    a. If top >= MAX - 1, cart is full.
    b. Else increment top and store item.
3. For pop (remove item):
    a. If top == -1, cart is empty.
    b. Else return item at top and decrement.
4. For display: loop from top to 0.
5. For total: sum of (quantity * price) of each item.

---

6. Code Implementation

```
#include <stdio.h>
#include <string.h>
#define MAX 100
// Stack pointer
int top = -1;
// Item structure
struct Item {
    char name[50];
    int quantity;
    float price;
};
```

```
// Stack of items
struct Item stack[MAX];
// Push function to add item to cart
void push(struct Item item) {
    if (top >= MAX - 1) {
        printf("Cart is full!\n");
        return;
    }
    stack[++top] = item;
    printf("Item added to cart successfully!\n");
}
// Pop function to remove last added item
struct Item pop() {
    if (top == -1) {
        printf("Cart is empty!\n");
        struct Item empty = {"", 0, 0.0};
        return empty;
    }
    printf("Item removed from cart successfully!\n");
    return stack[top--];
}
// Display all items in the cart
void displayCart() {
    if (top == -1) {
        printf("Cart is empty!\n");
        return;
    }
    printf("\nItems in your cart:\n");
    printf("----------------------------------------
\n");
    printf("%-20s %-10s %-10s\n", "Item Name", "Qty",
"Price");
    printf("----------------------------------------
\n");
    for (int i = top; i >= 0; i--) {
        printf("%-20s %-10d %-10.2f\n", stack[i].name,
stack[i].quantity, stack[i].price);
    }
```

```
    printf("----------------------------------------
\n");
}
// Calculate and print total amount
void calculateTotal() {
    float total = 0.0;
    if (top == -1) {
        printf("Cart is empty!\n");
        return;
    }
    for (int i = 0; i <= top; i++) {
        total += stack[i].quantity * stack[i].price;
    }
    printf("Total Amount: ₹ %.2f\n", total);
}
int main() {
    int choice;
    struct Item item;
    do {
        printf("\n=====  Supermarket  Billing  System
=====\n");
        printf("1. Add Item\n");
        printf("2. Remove Last Item\n");
        printf("3. Display Cart\n");
        printf("4. Calculate Total\n");
        printf("5. Exit\n");
        printf("Enter your choice (1-5): ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter item name: ");
                scanf(" %[^\n]", item.name);  // Reads
string with spaces
                printf("Enter quantity: ");
                scanf("%d", &item.quantity);
                printf("Enter price per item: ");
                scanf("%f", &item.price);
                push(item);
```

```
                break;
            case 2:
                pop();
                break;
            case 3:
                displayCart();
                break;
            case 4:
                calculateTotal();
                break;
            case 5:
                printf("Thank you for shopping with
us!\n");
                break;
            default:
                printf("Invalid choice, please select
again.\n");
        }
    } while (choice != 5);
    return 0;
}
```

7. Testing & Validation

- The system was tested by adding and removing multiple items.
- Edge cases such as popping from an empty stack and pushing to a full stack were handled.
- Validation included verifying total cost and correct item sequence.

---

8. Results & Discussion

- The stack-based billing system performed as expected.
- Items were stored and removed in LIFO order.
- Stack allowed for easy rollback of the last added item, which simulates removing an item from the cart.
- Limitations include lack of persistent storage and no GUI.

---

9. Future Enhancements

- Add file handling to save and retrieve bills.
- Implement a user-friendly graphical interface.
- Use dynamic stacks (linked list) to allow unlimited items.
- Add barcode scanning simulation.
- Include customer and admin login.

---

10. Conclusion

The Supermarket Billing System successfully demonstrates the application of the stack data structure in a real-life scenario. It provides a clear understanding of stack operations and their use in managing dynamic data such as a shopping cart.

---

11. References

- GeeksforGeeks – Stack in C
- TutorialsPoint – C Programming Basics
- Course Lecture Notes