N.B.K.R Institute of Science & Technology

Vidyanagar-Tirupati Dt

ATM Transaction Queue System

**Course: Data Structures**

**Department: Computer Science**

**Year: I**

**Semester: II**

**Submitted to:** **Ashok Selva Kumar**

Submitted by:

M. Hasini (24KB1A05Z8)

M. Mokshitha (24KB1A05AC)

P. Srilakshmi (24KB1A05CP)

P. Poojitha (24KB1A05CV)

**Acknowledgement:**

I would like to express my sincere gratitude to all those who supported and guided me in completing this project on the **ATM Transaction Queue System**.

I am especially thankful to my **Ashok Selva Kumar Sir** for his valuable guidance, feedback, and encouragement throughout the development of this project. Their insights helped me understand the concepts of data structures and programming more effectively.

I also extend my sincere thanks to the **institution**, and all the faculty members who provided the necessary resources and a learning environment for developing this project.

I also appreciate the support of my classmates, friends, and family for their continuous encouragement and motivation.

This project has helped me strengthen my understanding of C programming, particularly in implementing data structures like linked lists and queues, and applying them to real-life scenarios.

**Abstract**

The **ATM Transaction Queue System** is a console-based application developed in the C programming language to simulate real-world ATM operations using queue data structures. The system manages a queue of customers waiting to withdraw money, ensuring transactions are processed in a First-In-First-Out (FIFO) manner. Each customer entry includes a name and a withdrawal amount, with input validation to ensure amounts are in acceptable denominations (multiples of 100 and not less than 100). The system provides three main functionalities: adding a customer to the queue, processing withdrawals for the next customer in line, and displaying the current queue status. This project demonstrates core concepts of dynamic memory allocation, linked lists, input handling, and basic error checking, offering a simplified yet functional model of real ATM queue management.

**Introduction:**

The ATM Transaction Queue System is a C program designed to simulate the process of handling customer transactions in a queue. It uses a linked list to manage a line of customers, ensuring that each customer is served in the order they arrive (First-In-First-Out). The system allows users to add new customers, process transactions one at a time, and view the current queue. It also includes basic input validation and demonstrates key programming concepts such as structures, pointers, dynamic memory allocation, and queue operations in c.

**Objective:**

The main objective of this project is to develop a console-based ATM Transaction Queue System using the C programming language. The system aims to:

* Simulate the queuing process of customers at an ATM.
* Ensure customers are served in the order they arrive (FIFO).
* Allow users to add, process, and view customers in the queue.
* Implement basic input validation to ensure proper system operation.
* Demonstrate the use of linked lists, structures, pointers, and dynamic memory allocation in managing real-time queue operations

**System Requirements:**

**Software Requirements:**

Code: Blocks IDE or Turbo C++ Compile

Windows/Linux Operating System (or online C compiler)

**Hardware Requirements:**

Minimum 2 GB RAM

Intel Core i3 Processor or higher

100 MB of disk space for storing files

**Literature Review**

Queue systems are commonly used in many real-life situations such as banks, hospitals, and customer service centers. A queue works on the First-In-First-Out (FIFO) method, where the first person to enter is the first to be served. In computer programming, queues can be created using data structures like arrays or linked lists.

In ATM systems, managing customers in a proper order is important to ensure smooth service. Many basic queue programs have been developed to simulate how customers are added, served, and removed from the line. This project uses a linked list to create a dynamic queue, which allows adding and removing customers easily. The concepts used in this program are commonly taught in data structures and programming courses.

**Methodology:**

The main steps followed in building the system are:

1. First we have defined the structure of the customer.
2. Then implement the Queue operations like **enqueue(),dequeue().**
3. Before adding a customer, we have to check if the entered amount meets the required conditions or not.
4. Then we will give the menu with various options.
5. We will allocate the memory dynamically using **malloc(),free().**

This method ensures that customers are handled in the correct order and that system resources are managed efficiently.

**Project description:**

**Problem statement:**

The task is to develop a system that simulates ATM transaction processing using a queue. The system should allow customers to enter their name and withdrawal amount, validate the amount, and process customers in the order they are added to the queue.

**Proposed Solution:**

The system will use a linked list-based queue to manage customers, where each customer’s name and withdrawal amount are stored. The program will validate withdrawal amounts and process customers in FIFO order.

**Key Features:**

1. Customer Queue Management.
2. Withdrawal Validation.
3. Scale and manage customer data efficiently.
4. Interactive Menu.
5. Customer details viewing.
6. Queue display.
7. Process transaction.

**Flowchart:**

**Algorithm:**

Step 1: Initialize an empty queue.

Step 2: Display menu:

1. Add customer
2. Process Transaction
3. Display Queue
4. Exit

Step 3: Based on the option:

* Add customer:
* Input: ID, Name, with drawl Amount.
* Create a node and add it to the rear of the queue.
* Process transaction:
* If queue is not empty:
* Retrieve customer at front.
* Simulate with drawl with output message.
* Remove customer from queue.
* Else: Show empty queue message.
* Display Queue:
* Traverse from front to rear and display all customers.
* Step 4: Repeat until the user exits.

**Program Code:**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#define NAME\_LENGTH 100

// Structure for a customer node

typedef struct Customer {

char name [NAME\_LENGTH];

int amount;

struct Customer\* next;

} Customer;

// Queue front and rear

Customer\* front = NULL;

Customer\* rear = NULL;

// Enqueue: Add a customer to the queue

void addCustomer(char name [], int amount) {

Customer\* new Customer = (Customer\*)malloc(sizeof(Customer));

Strcpy(newCustomer->name, name);

newCustomer->amount = amount;

newCustomer->next = NULL;

if (rear == NULL) {

front = rear = newCustomer;

} else {

rear->next = newCustomer;

rear = newCustomer;

}

if(amount<100 || (amount%100!=0)){

printf("Invalid Amount");

} else {

printf("Customer %s added with withdrawal amount %d.\n", name, amount);

}

}

// Dequeue: Process the first customer

void processCustomer() {

if (front == NULL) {

printf("Queue is empty. No customer to process.\n");

return;

}

printf("Processing customer: %s\n", front->name);

printf("Withdrawal Amount: %d\n", front->amount);

// Simulate successful transaction

Customer\* temp = front;

front = front->next;

// If queue becomes empty

if (front == NULL)

rear = NULL;

printf("Transaction successful for %s. Removing from queue.\n", temp->name);

free(temp);

}

// Display queue

void displayQueue() {

if (front == NULL) {

printf("Queue is empty.\n");

return;

}

Customer\* temp = front;

printf("\n--- ATM Queue ---\n");

while (temp != NULL) {

printf("Customer: %s | Amount: %d\n", temp->name, temp->amount);

temp = temp->next;

}

printf("------------------\n");

}

// Menu

int main() {

int choice, amount;

char name[NAME\_LENGTH];

while (1) {

printf("\nATM Transaction Queue System\n");

printf("1. Add Customer\n");

printf("2. Process Withdrawal\n");

printf("3. Display Queue\n");

printf("4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

getchar(); // To consume newline left by scanf

switch (choice) {

case 1:

printf("Enter customer name: ");

fgets(name, NAME\_LENGTH, stdin);

name[strcspn(name, "\n")] = '\0'; // Remove newline

printf("Enter withdrawal amount: ");

scanf("%d", &amount);

addCustomer(name, amount);

break;

case 2:

processCustomer();

break;

case 3:

displayQueue();

break;

case 4:

printf("Exiting ATM System. Goodbye!\n");

return 0;

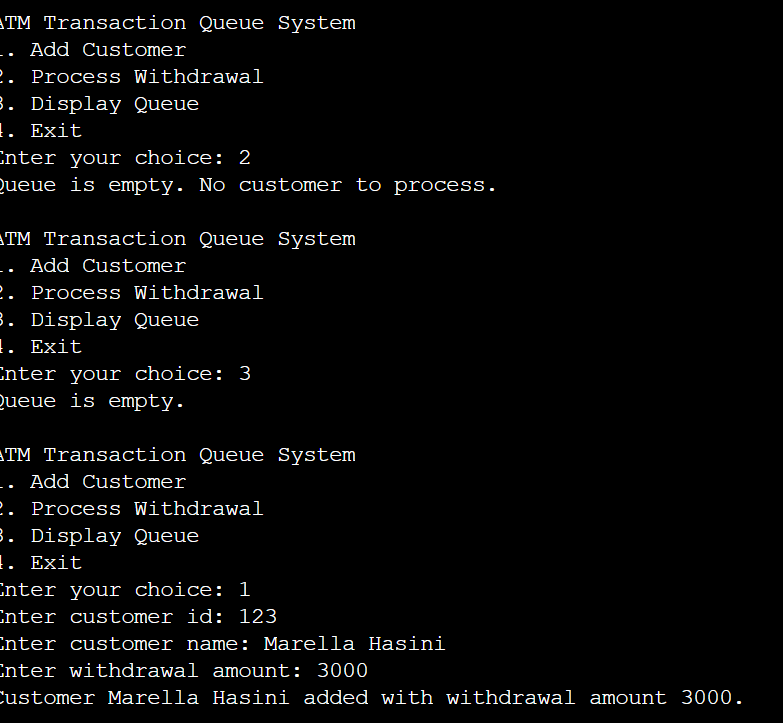
default:

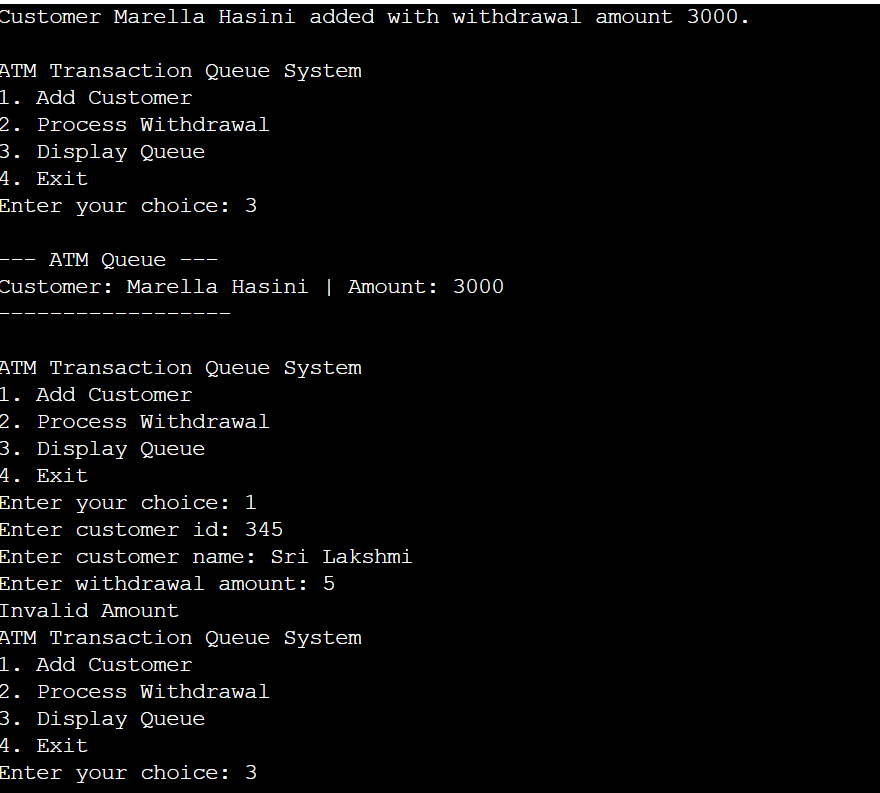
printf("Invalid choice. Please try again.\n");

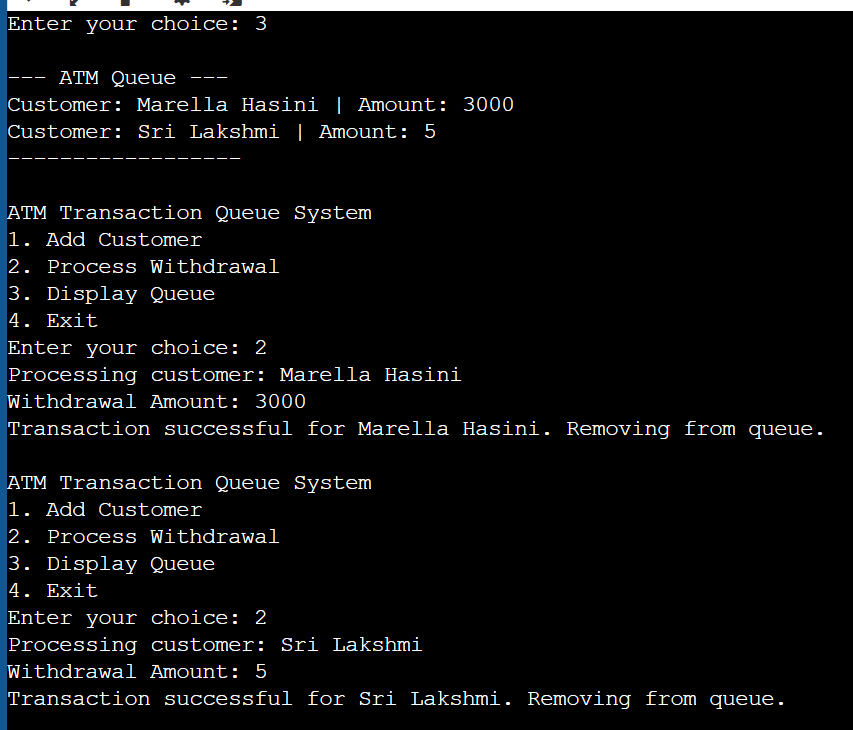
}

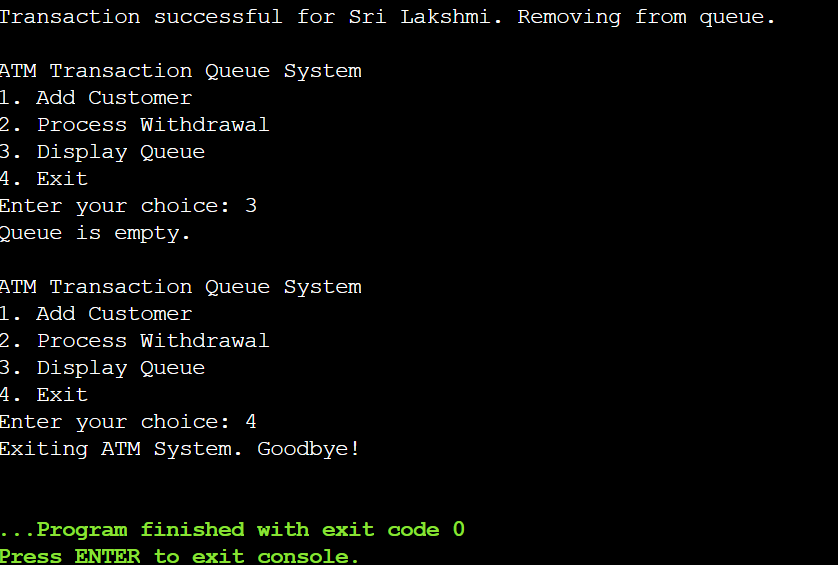
}

return 0;

**Output:**

****





**Testing and Validation:**

**🔹 Test Case 1: Add Valid Customer**

**Input:**

Choice: 1

Enter customer id: 1

Enter customer name: Alice

Enter withdrawal amount: 100

**Output:**

Customer Alice added with withdrawal amount 100.

✅ **Status**: Passed

**🔹 Test Case 2: Add Another Valid Customer**

**Input:**

Choice: 1

Enter customer id: 2

Enter customer name: Bob

Enter withdrawal amount: 500

**Output:**

Customer Bob added with withdrawal amount 500.

✅ **Status**: Passed

**🔹 Test Case 3: Add Customer with Amount < 100**

**Input:**

Choice: 1

Enter customer id: 3

Enter customer name: Charlie

Enter withdrawal amount: 50

**Output:**

Invalid Amount

✅ **Status**: Passed

**🔹 Test Case 4: Add Customer with Amount Not a Multiple of 100**

**Input:**

Choice: 1

Enter customer id: 4

Enter customer name: David

Enter withdrawal amount: 350

**Output:**

Invalid Amount

✅ **Status**: Passed

**🔹 Test Case 5: Display Queue**

**Input:**

Choice: 3

**Output:**

--- ATM Queue ---

Customer: Alice | Amount: 100

Customer: Bob | Amount: 500

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

✅ **Status**: Passed

**🔹 Test Case 6: Process First Customer**

**Input:**

Choice: 2

**Output:**

Processing customer: Alice

Withdrawal Amount: 100

Transaction successful for Alice. Removing from queue.

✅ **Status**: Passed

**🔹 Test Case 7: Process Second Customer**

**Input:**

Choice: 2

**Output:**

Processing customer: Bob

Withdrawal Amount: 500

Transaction successful for Bob. Removing from queue.

✅ **Status**: Passed

**🔹 Test Case 8: Process When Queue is Empty**

**Input:**

Choice: 2

**Output:**

Queue is empty. No customer to process.

✅ **Status**: Passed

**🔹 Test Case 9: Exit**

**Input:**

Choice: 4

**Output:**

**E**xiting ATM System. Goodbye!

✅ **Status**: Passed

**Limitations:**

1. No Duplicate Customer ID Check.
2. Manual Memory Management.
3. Limited Error Handling.
4. No Transaction History.
5. No Built-in String Handling.
6. Fixed Amount Validation (Only multiples of 100).
7. No Account Balance Checking.
8. No User Authentication Mechanism.
9. No Support for Concurrent Transactions.
10. No Built-in Data Structures.

**Future Enhancements:**

1. Customer ID Validation.
2. Flexible Withdrawal Amounts.
3. Account Balance Management.
4. Transaction Logging.
5. User Authentication.
6. Dynamic Queue Size.
7. Enhanced Error Handling.
8. Concurrent Processing (Multi-threading).
9. Graphical User Interface (GUI).
10. Network Integration

.

**Conclusion:**

In conclusion, the ATM Transaction Queue System developed in C is a simple yet effective approach to managing customer transactions in an ATM environment. The system allows customers to be added to a queue, processed for withdrawals, and provides basic functionality for transaction management. While the code successfully demonstrates queue handling and basic withdrawal processing, there are some limitations such as lack of account balance management, no support for multi-threading, and insufficient error handling.

Future improvements could focus on adding features like customer ID validation, account balance management, user authentication, and a graphical user interface (GUI) for better user interaction. Additionally, enhancements such as transaction logging, multi-threading for concurrent customer processing, and more flexible withdrawal options could further enhance the system's real-world applicability.

Overall, the ATM Transaction Queue System serves as a foundation for building more advanced and secure ATM systems, and with future improvements, it can evolve into a fully functional, user-friendly, and scalable solution for real-world banking scenarios.

**References:**

1.Algorithms and Data Structures: The Basic Toolbox by Kurt Mehlhorn and Peter Sanders

2. C Data Structures and Algorithms by Alfred V. Aho, Jeffrey D. Ullman, and John E. Hopcroft

3. Problem Solving with Algorithms and Data Structures" by Brad Miller and David Ranum

4. Introduction to Algorithms by Thomas HCormen, Charles ELeiserson, Ronald L. Rivest, and Clifford Stein

5. Algorithms in C, Parts 1-5 (Bundle): Fundamentals, Data Structures, Sorting, Searching, and Graph Algorithms" by Robert Sedgewick