**NBKR INSTITUTE OF SCIENCE AND TECHNOLOGY**

**(VIIDYANAGAR)**

**BANK CUSTOMER QUEUE**

**COURSE**:[Data structure]

**BRANCH**: Computer Science Engineering

**SECTION**: C

**YEAR**: 1st

**SEMESTER**: 2nd

**SUBMITTED TO**: Ashok Selva Kumar

**SUBMITTED BY:** S.HARINI

G.NANDHINI

K.RAKSHITHA

ACKNOWLEDGEMENT:

I would like to express my sincere gratitude to [Ashok Selva Kumar] for their valuable guidance ,constant encouragement, and insightful suggestions throughout the development of this project, “customer queue management at bank using linked list”.

Finally ,I acknowledge the resources ,tutorials ,and reference materials that helped deepen my understanding of queue operations and linked list implementation.

ABSTRACT:

In banking environments, efficient customer management is critical to ensure smooth service operations. This project focuses on implementation a customer queue system using a queue data structure to manage customers effectively. Enqueue and dequeue operations are used to add and remove customers from the queue, respectively, ensuring that services are provided on a first-come, first –served basis. the queue is implemented using a linked list to allow the dynamic memory management and efficient handling of customer data without size limitations. This approach ensures optimized performance and flexibility in managing customer flow at the bank.

INTRODUCTION:

\* managing customer efficiently is a crucial task in banking operations.

\*to simulate and practice queue management we developed a program that implements a customer queue system using the c programming language .

\* the system uses a linked list that dynamically manage customers entering and leaving the queue.

OBJECTIVE:

* To implement a queue data structure using linked lists in c.
* To manage the real-time addition (enqueue)and removal (dequeue) of customers.
* To display the queue status at point.
* To understand dynamic memory management (malloc,free)in c
* To simulate real-world queue handling in banks.

SYSTEM REQUIREMENTS:

HARDWARE REQUIREMENTS: PROCESSOR: Intel core higher

RAM: 2GB Minimum (4GB Recommended )

HARD DISK: 500 MB of free space

MONITOR: Any Standard display

KEY BOARD AND MOUSE FOR USER INPUT

SOFTWARE REQUIREMENTS:

OPERATING SYSTEM: windows , linux .

COMPILER:GCC(GNU compiler collection)

TEXT EDITOR OR IDE:

CODE: Blocks/visual studio code/devc++/turboc

OPTIONAL: git for version control.

LITERATURE REVIEW :

Queue management is essential in systems like banks where customers must be served in the order of their arrival. A queue is a linear data structure based on the first- in-first-out(FIFO)principle.

Thus implementation a bank customer queue using linked lists ensures flexibility,efficiency ,and better memory management.

METHODOLOGY:

This project implements a queue data structure using a linked list to manage customer flow at a bank .

Queue concept :follows the first-in-first-out principle, where the first customer added is the first to be served.

Linked list implementation: each customer is represented as a node containing their ID and a pointer to the next node.

Enqueue operation: when a new customer arrives ,a new node is created and added at the rear ofthe queue.

Dequeue operaion : when a customer is served , the node at the front of the queue is removed .

Memory management : dynamic memory allocation ensures there is no predefined size limit unlike array-based queues.

Efficiency: both enqueue and dequeue operations are performed in O(O1)time,ensuring fast customer service even during peak times.

PROJECT DESCRIPTION:

The customer queue management system is designed to efficiently handle customer flow at a bank using a queue implemented through a linked list.

This system eliminates the static size limitation present in array-based queues and ensure faster and flexible management of customer data.

The program is built using c programming language, offering a simple yetpowerful solution for real-time bank operations.

FLOW CHART :

START

DISPLAY MENU

1.ENQUEUE

2.DEQUEUE

3.DISPLAY

4.EXIT

USER SELECTS AN OPTION

OPTION1:ENQUEUE

-INPUT CUSTOMER ID

-CREATE NODE

-ADD NODE TO REAR

OPTION2:DEQUEUE

-CHECK IF QUEUE IS EMPTY

-IF NOT,REMOVE FRONT NODE

OPTION3.DISPLAY QUEUE

-TRAVERSE AND PRINT IDS

OPTION4.EXIT

-TERMINATE PROGRAM

END

ALGORITHM:

\*1.ENQUEUE(adding a customer ):

\*create anew node with customer’s name.

\*if the queue is empty then set front and rear to the new node.

\*else linkthe new node at the end and update rear .

\*2.DEQUEUE(serving the customer)

\*check if the queue is empty .if yes display a message .

\*else, remove the front node and update front node to the next node.

\*if after dequeue the queue become s empty,set rear o NULL.

\*3. DISPLAY QUEUE :

\*traverse from front to rear, printing each customers name.

CODE:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// Node structure

struct Node {

char name[50];

struct Node\* next;

};

// Front and rear pointers

struct Node\* front = NULL;

struct Node\* rear = NULL;

// Enqueue operation

void enqueue(char\* name) {

struct Node\* temp = (struct Node\*)malloc(sizeof(struct Node));

strcpy(temp->name, name);

temp->next = NULL;

if (rear == NULL) {

front = rear = temp;

} else {

rear->next = temp;

rear = temp;

}

printf("Customer '%s' added to the queue.\n", name);

}

// Dequeue operation

void dequeue() {

if (front == NULL) {

printf("Queue is empty. No customers to serve.\n");

return;

}

struct Node\* temp = front;

printf("Customer '%s' served and removed from the queue.\n", temp->name);

front = front->next;

if (front == NULL) {

rear = NULL;

}

free(temp);

}

// Display queue

void displayQueue() {

if (front == NULL) {

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

return;

}

struct Node\* temp = front;

printf("Customers in queue:\n");

while (temp != NULL) {

printf("- %s\n", temp->name);

temp = temp->next;

}

}

int main() {

int choice;

char name[50];

while (1) {

printf("\n--- Bank Customer Queue ---\n");

printf("1. Enqueue customer\n");

printf("2. Dequeue customer\n");

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

printf("4. Exit\n");

printf("Choose an option: ");

scanf("%d", &choice);

getchar(); // to consume newline after scanf

switch (choice) {

case 1:

printf("Enter customer name: ");

fgets(name, sizeof(name), stdin);

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

enqueue(name);

break;

case 2:

dequeue();

break;

case 3:

displayQueue();

break;

case 4:

printf("Exiting...\n");

exit(0);

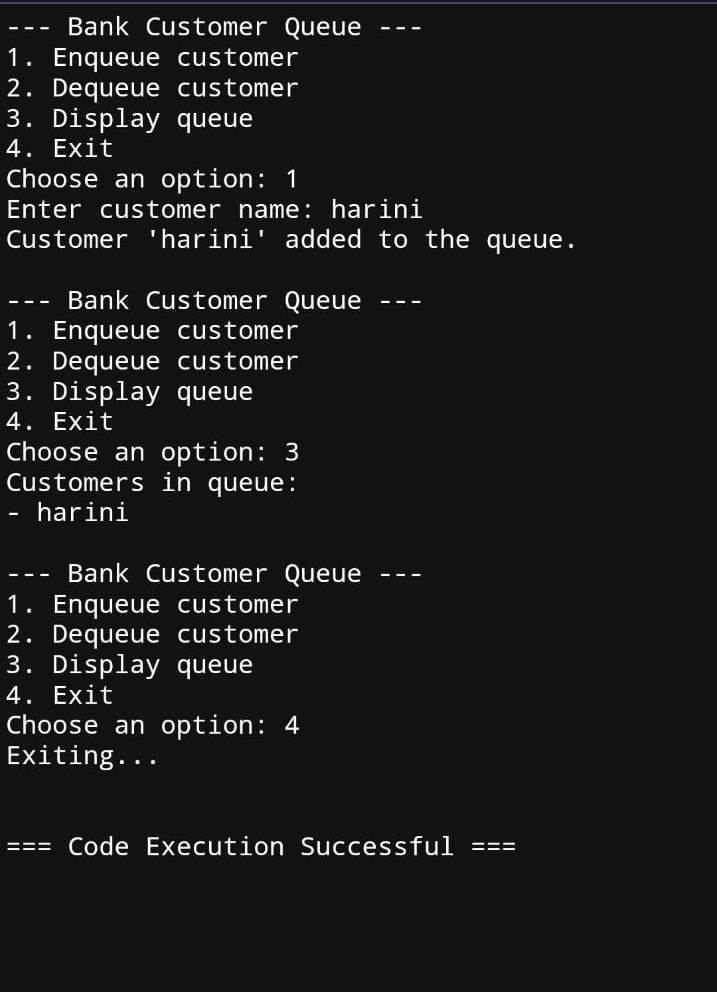
default:

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

}

}

return 0;

}

OUT PUT:

LIMITATIONS:

No priority handling:

All customers are treated equally in the order they arrive .there is no way to prioritise emergency or VIP customer

Manual data entry:

Customer Ids must be entered manually in real banking systems this process is au [8:44 pm, 28/4/2025] harini: Memory Dependence:

Though the linked list allows dynamic size, it is still limited by the system’s available memory. Very large queues could eventually cause memory exhaustion.

No Persistence:

The queue exists only during program execution. Once the program ends, all data is lost; no data is stored for future use.

No Error Handling for Overflow:

While unlikely in linked lists, if memory allocation fails (due to s…

Future Enhancements

Priority Queue System:

Introduce priority levels to serve VIP or emergency customers faster than regular customers.

Data Persistence:

Save the queue data to a file or database so that customer information is not lost when the program closes.

Graphical User Interface (GUI):

Develop a user-friendly interface to allow bank staff to manage the queue easily without needing command-line knowledge.

Memory Allocation Handling:

Add advanced error checking for memory allocation failures to prevent program crashes.

Integration with Token Dispensers:

Automate customer ID generation using a token system similar to real-world banks.

Customer Details Expansion:

Instead of just customer ids ,store more information like name,purpose of visit ,and expected service time .

CONCLUSION:

implementing a customer queue at a bank using a linked list in C allows for efficient management of customers as they arrive and are served. The linked list provides dynamic memory allocation, which makes it flexible in handling varying queue sizes without the limitations of fixed-size arrays. By employing queue operations such as enqueue (adding customers to the queue) and dequeue (serving customers), the system ensures orderly processing of customer requests. The linked list structure also makes it easier to maintain the queue's integrity, handling edge cases such as empty queues or full queues seamlessly. Overall, this approach is both efficient and adaptable for the needs of a banking system, offering scalability and ease of implementation.

REFERENCES:

1. C Programming Language - Brian W. Kernighan and Dennis M. Ritchie (2nd Edition)

This book is a foundational resource for learning C programming and includes details on data structures like linked lists

2. Data Structures and Algorithms in C - Aaron M. Tenenbaum

This book provides an in-depth explanation of various data structures, including linked lists, and how to implement them in C.

3. GeeksforGeeks - Linked List Data Structure

This website has numerous tutorials and code examples for implementing linked lists and queues in C.

4. Programiz - C Programming Linked List

.