M.Kumarasamy College of Engineering
NAAC Accredited Autonomous Institution
Approved by AICTE & Affiliated to Anna University
ISO 9001:2015 Certified Institution
Thalavapalayam, Karur - 639 113, TAMILNADU.

A Project Report

on

QUEUE BASED HOT POTATO GAME

Submitted in partial fulfilment of requirements for the award of the course

of

CGB1122 – DATA STRUCTURES

Under the guidance of

Mrs. K. MAKANYADEVI M.E.,

Assistant Professor/CSE

Submitted By

SANJAI KUMAR R (927624BAD087)

DEPARTMENT OF FRESHMAN ENGINEERING

M.KUMARASAMY COLLEGE OF ENGINEERING
(Autonomous)

KARUR – 639 113

MAY 2025

M.Kumarasamy College of Engineering
NAAC Accredited Autonomous Institution
Approved by AICTE & Affiliated to Anna University
ISO 9001:2015 Certified Institution
Thalavapalayam, Karur - 639 113, TAMILNADU.

M. KUMARASAMY COLLEGE OF ENGINEERING

(Autonomous Institution affiliated to Anna University, Chennai)

KARUR – 639 113

BONAFIDE CERTIFICATE

Certified that this project report on **"QUEUE BASED HOT POTATO GAME"** is the bonafide work of **SANJAI KUMAR R (927624BAD087)** who carried out the project work during the academic year 2024- 2025 under my supervision.

Signature

Mrs. P. KAYALVIZHI M.E.,

SUPERVISOR,

Department of Computer Science and Engineering,

M. Kumarasamy College of Engineering,

Thalavapalayam, Karur -639 113.

Signature

Dr. K.CHITIRAKALA, M.Sc., M.Phil.,Ph.D.,

HEAD OF THE DEPARTMENT,

Department of Freshman Engineering,

M. Kumarasamy College of Engineering,

Thalavapalayam, Karur -639 113.

Submitted for the End Semester Review held on _____

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

VISION OF THE INSTITUTION

To emerge as a leader among the top institutions in the field of technical education

MISSION OF THE INSTITUTION

- Produce smart technocrats with empirical knowledge who can surmount the global challenges
- Create a diverse, fully-engaged, learner-centric campus environment to provide quality education to the students
- Maintain mutually beneficial partnerships with our alumni, industry, and Professional associations

VISION OF THE DEPARTMENT

To produce competent industry relevant education, skillful research, technical and innovative computer science professionals acquaintance with managerial skills, human and social values.

MISSION OF THE DEPARTMENT

- To impart technical knowledge through innovative teaching, research, and consultancy.
- To develop and to promote student ability thereby to compete globally through excellence in education.
- To facilitate the development of academic-industry Collaboration.
- To produce competent engineers with professional ethics, technical competence and a spirit of innovation and managerial skills.

PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

PEO 1: To acquire technical knowledge and proficiency required for the employment and higher education in the contemporary areas of computer science or management studies.

PEO 2: To apply their competency in design and development of innovative solutions for real-world problems.

PEO 3: To demonstrate leadership qualities with high ethical standards and collaborated with other industries for the socio-economical growth of the country.

PROGRAM OUTCOMES (POs)

Engineering students will be able to:

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES (PSOs)

1. **PSO1:** Ability to apply the analytical and business skills to provide sustainable solutions as an engineer/researcher for the real-time applications using Machine Learning, Internet of Things and Data analytics.
2. **PSO2:** Ability to practice ethical and human values with soft-skills qualities in computer science and business disciplines to emerge as an entrepreneur for the growth and development of the society.

ABSTRACT

The “Queue Based Hot Potato Game” is an engaging simulation project developed using the C programming language, aimed at demonstrating the real-time application of the circular queue data structure . The concept is based on the popular Hot Potato game, where players sit in a circle and pass an imaginary object- the hot potato-until the timer runs out, resulting in the elimination of the players holding it. In this implementation, players are stored in a circular queue, and after a fixed number of passes, the front player is removed, simulating the elimination process. This continues until only one player remains , who is declared the winner. The project effectively illustrates queue operations such as enqueue, dequeue, and circular queue function in a continuous loop without memory overflow . It not only enhances the understanding of data structures but also shows how they can be used in game development, logic building, and problem-solving. This project serves as a useful learning tool for students to connect theoretical knowledge with practical implementation in an enjoyable way.

ABSTRACT WITH POs AND PSOs MAPPING

ABSTRACT	POs MAPPED	PSOs MAPPED
The project "Queue Based Hot Potato Game", uses the concept of circular queues to simulate a popular elimination game. Each player is added to the queue, and the "hot potato" is passed around a fixed number of times. The player holding the potato at the end is removed, and the game continues until one player remains . This project helps in understanding how circular queues work, especially in situations where elements are processed in a circular and repeatative manner. It provides a fun and practical approach to learning queue operations like insertion , deletion , and looping through elements efficiently	**PO1(2)** **PO2(3)** **PO3(2)** **PO4(2)** **PO5(3)** **PO6(1)** **PO7(3)** **PO8(2)** **PO9(3)** **PO10(3)** **PO11(2)** **PO12(2)**	**PSO1(3)** **PSO2(2)**

Note: 1- Low, 2-Medium, 3- High

SUPERVISOR **HEAD OF THE DEPARTMENT**

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Introduction

The Hot Potato Game is a traditional elimination game where players sit in a circle and pass an object—symbolizing a "hot potato"—while a song or timer runs. When the timer stops, the player holding the object is eliminated. This cycle continues until only one player remains, who is declared the winner. This concept can be effectively translated into a computer simulation to understand circular flow, elimination logic, and fair turn-based decision-making.

In this project, the game is simulated using the circular queue data structure in C programming. Each player is represented as a node in the queue, and the passing of the potato is simulated by rotating the queue elements. After a specified number of passes, the player at the front of the queue is removed. This process is repeated until only one player remains. Circular queues allow for smooth rotation of elements without needing to reset the structure, making them ideal for this application.

This implementation helps in understanding fundamental data structure operations such as enqueue, dequeue, and circular iteration. It also introduces modular programming concepts and dynamic memory handling. The Hot Potato simulation serves as an educational tool for mastering queue operations in a fun and practical way, improving both logic building and problem-solving skills.

1.2 Objective

The main objective of this project is to develop a queue-based simulation of the Hot Potato Game, where players are enqueued and eliminated based on the predefined rules of the game. This implementation will demonstrate circular traversal, element deletion, queue manipulation, and random or fixed elimination count logic in a controlled programming environment. The project aims to build a clear understanding of circular queue operations, dynamic memory management, modular

programming in C, and game-based application development using data structures.

1.3 Data Structure Choice

The queue data structure, especially a circular queue, is well-suited for simulating games like Hot Potato where players are rotated continuously. Using a circular queue allows the program to pass the 'hot potato' among players without reaching the end of the list, simulating real-life circular movement. Operations like enqueue (adding a new player), dequeue (removing an eliminated player), and peeking (to see who currently holds the potato) are fundamental. Circular queues also prevent unnecessary memory use and improve efficiency by reusing space in a fixed-size structure.

CHAPTER 2
PROJECT METHODOLOGY

2.1 CIRCULAR QUEUE:

A circular queue is a linear data structure that follows the First In First Out (FIFO) principle, but unlike a simple queue, it connects the last position back to the first position to form a circle. It is particularly useful when memory needs to be reused efficiently without shifting elements. In the context of the "Hot Potato Game," a circular queue is an ideal choice because it allows players to be rotated in a circle until a specific condition is met.

Key features of a circular queue:

1. Efficient Memory Utilization:

In a circular queue, the space of deleted elements can be reused without the need for shifting all elements forward. This is achieved by connecting the rear pointer back to the front when the end of the queue array is reached, thus forming a circle.

2. Continuous Rotation:

Circular queues enable continuous traversal of elements without restarting from the beginning manually. In the Hot Potato Game, this allows players to pass the "potato" in a loop seamlessly until one remains.

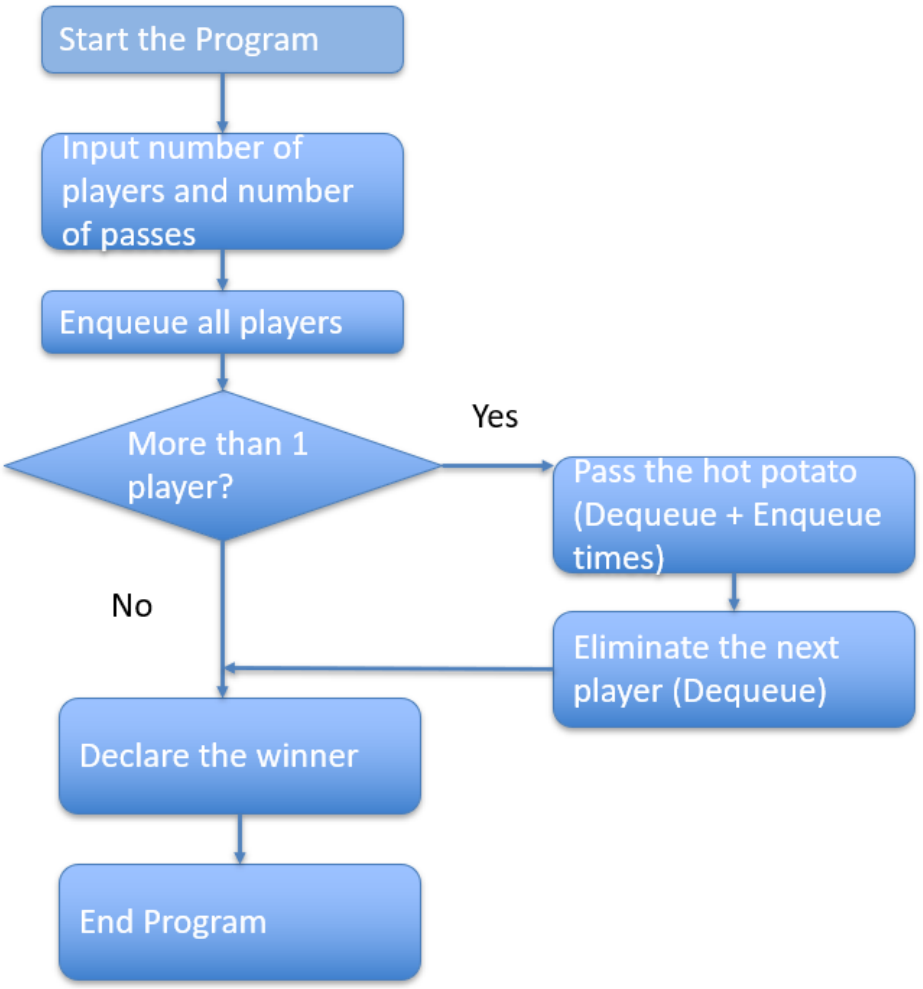
3. FIFO Operations:

A circular queue retains the FIFO behavior, which is ideal for modeling real-world scenarios such as player turns. Elements are added at the rear and removed from the front, ensuring that players are eliminated in the order dictated by the game rules.

4. Implementation Efficiency:

Circular queues offer constant time complexity (O(1)) for enqueue and dequeue operations. This makes the game simulation smooth and fast even with a large number of players.

2.2 Block Diagram



CHAPTER 3
MODULES

The "Queue-Based Hot Potato Game" project is divided into several key modules, each responsible for specific tasks to ensure the smooth execution and simulation of the game. These modules are designed in a modular approach to enhance clarity, maintainability, and reusability.

3.1 Input Module:

This module is responsible for collecting input from the user. The inputs include:

- Total number of players.
- Names or identifiers of each player.
- The number of passes (count) after which a player is eliminated.

The input module ensures all data is correctly captured and validated before proceeding with the game simulation.

3.2 Queue Initialization Module:

This module initializes the circular queue structure required for the game. It:

- Creates an array-based queue with front and rear pointers.
- Allocates memory for storing player names.
- Ensures proper circular linkage for continuous traversal.

3.3 Enqueue Operation Module:

This module handles the insertion of players into the circular queue. It:

- Adds each player to the queue in the order they are input.
- Checks for queue overflow conditions.
- Maintains the circular connection when the rear reaches the end.

3.4 Dequeue Operation Module:

This module removes a player from the queue after the specified number of passes it:

- Checks for queue underflow.
- Updates the front pointer after elimination.
- Returns the eliminated player's name for display.

3.5Game Simulation Module:

This is the core logic module of the game. It:

- Rotates the circular queue according to the user-defined pass count.
- Eliminates the player holding the "hot potato" after the count ends.
- Displays each eliminated player in sequence.
- Continues until only one player remains.

3.6 Output Display Module

This module handles all output to the user. It:

- Displays the names of players as they are eliminated.
- Announces the winner once the game ends.
- May optionally display the queue state after each round (for debugging or demonstration).

CHAPTER 4

RESULTS AND DISCUSSION

4.1 Results

4.1.1 Player Initialization:

```
Enter number of players: 5
Enter name of player 1: sanjai
Enter name of player 2: sabari
Enter name of player 3: yogesh
Enter name of player 4: harish
Enter name of player 5: Kishore
```

4.1.2 : Pass Count Input:

```
Enter number of players: 5
Enter name of player 1: sanjai
Enter name of player 2: sabari
Enter name of player 3: yogesh
Enter name of player 4: harish
Enter name of player 5: kishore
Enter the number of passes before
    elimination: 3
```

4.1.3 Game Progress:

```
Enter number of players: 5
Enter name of player 1: sanjai
Enter name of player 2: sabari
Enter name of player 3: yogesh
Enter name of player 4: harish
Enter name of player 5: kishore
Enter the number of passes before
    elimination: 3

Game starting...
Eliminated: yogesh
Eliminated: sanjai
Eliminated: kishore
Eliminated: sabari
```

4.1.4 Game Result:

```
Enter number of players: 5
Enter name of player 1: sanjai
Enter name of player 2: sabari
Enter name of player 3: yogesh
Enter name of player 4: harish
Enter name of player 5: kishore
Enter the number of passes before
    elimination: 3

Game starting...
Eliminated: yogesh
Eliminated: sanjai
Eliminated: kishore
Eliminated: sabari

The winner is: harish
```

4.2 Discussion:

The implementation of the Queue-Based Hot Potato Game demonstrates how abstract data structures like queues can be effectively used to simulate real-world scenarios. In this game, the queue acts as a circular system where players continuously pass a "hot potato" token, and elimination happens after a fixed number of passes. This simulation offered a valuable opportunity to explore the fundamentals of the circular queue, such as enqueue (insertion), dequeue (removal), and front/rear pointer manipulation, all of which are critical for queue management. During the simulation, players were added into a circular queue structure based on user input. The game then asked for the number of passes before each elimination. The queue rotated that many times, and the player at the front was eliminated. The game loop continued to perform enqueue and dequeue operations dynamically until only one player remained. This entire process not only reinforced the theoretical understanding of queues but also demonstrated the importance of logical control flow, loop design, memory allocation, and program modularity in C. It showcased how real-time applications, such as scheduling systems, multiplayer games, and turn-based algorithms, benefit from circular queue implementations. The project also tested the system's response to different edge cases, such as minimum players, a single pass, or repeated names. Overall, the game functioned accurately across various scenarios, validating the correctness of the implemented logic and data structure.

CHAPTER 5
CONCLUSION

The Queue-Based Hot Potato Game project successfully simulates a real-life elimination game using one of the fundamental data structures—queues. Through this project, we gained hands-on experience with implementing circular queues and applying them in a meaningful and interactive context. The game provided a creative and engaging method to understand how circular logic and queue-based operations work in programming, especially in C.This project enhanced our skills in memory management, modular function design, and algorithmic thinking. Each function—from adding players to eliminating them and declaring the winner—was carefully designed to reflect real-world game behavior. Error handling and user interaction were also included to ensure a smooth experience and prevent unexpected crashes.In conclusion, the Hot Potato Game project served both as an educational and practical application of queue data structures. It laid a strong foundation for implementing more complex data-driven simulations and interactive programs. The experience also underscored the importance of selecting the right data structure for the right problem, which is a key skill in computer science and software development.

REFERENCES

1. Yashavant Kanetkar – Data Structures Through C
2. Ellis Horowitz, Sartaj Sahni – Fundamentals of Data Structures in C
3. Stack Overflow – Circular Queue Implementation Examples
4. GeeksforGeeks – Queue and Circular Queue Tutorials
5. TutorialsPoint – Queue Data Structure in C
6. ScienceDirect – Queue-Based Algorithms in Game Simulation
7. GitHub Repositories – Sample Hot Potato Game Implementations in C

APPENDIX

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
// Structure to represent a contact typedef
struct Contact
{
char name[50];
char phoneNumber[15];
struct Contact* next;
struct Contact* prev;
 } Contact;

// Function to create a new contact Contact* createContact(const char* name,
const char* phoneNumber)
{
 Contact* newContact = (Contact*)malloc(sizeof(Contact));
 if (newContact != NULL)
{
strcpy(newContact->name, name);
strcpy(newContact->phoneNumber, phoneNumber);
newContact->next = NULL;
newContact->prev = NULL;
```

```
}
return newContact;
}

// Function to insert a contact into the phone directory void
insertContact(Contact** head, const char* name, const char* phoneNumber)
{
Contact* newContact = createContact(name, phoneNumber);
if (newContact == NULL)
{
printf("Memory allocation failed.\n");
return;
}
if (*head == NULL)
{ // List is empty
*head = newContact;
}
else
{
 // Insert at the end
Contact* temp = *head;
while (temp->next != NULL)
{
temp = temp->next;
}
temp->next = newContact;
newContact->prev = temp;
}
printf("Contact added successfully.\n");
```

```
}



// Function to display all contacts in the phone directory
 void displayContacts(Contact* head)
 { printf("\nPhone Directory:\n");
  while (head != NULL)
 {
 printf("Name: %s, Phone Number: %s\n", head->name, head->phoneNumber);
head = head->next;
} printf("\n");
 }
 // Function to search for a contact by name
Contact* searchContact(Contact* head, const char* name)
{
  while (head != NULL)
 {
if (strcmp(head->name, name) == 0)
{
return head;
 // Contact found
 }
 head = head->next;
  }
 return NULL;
// Contact not found
 }
```

```
int main()
{
 Contact* phoneDirectory = NULL;
  // Menu-driven phone directory application
int choice;
  char name[50];
 char phoneNumber[15];
 do {
```

```
    printf("\nPhone Directory Application\n");
    printf("1. Add Contact\n");
    printf("2. Display Contacts\n");
    printf("3. Search Contact\n");
    printf("4. Delete Contact\n");
    printf("5. Edit Contact\n");
    printf("6. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            printf("Enter contact name: ");
            scanf("%s", name);
            printf("Enter phone number: ");
            scanf("%s", phoneNumber);
            insertContact(&phoneDirectory, name, phoneNumber);
            break;
        case 2:
            displayContacts(phoneDirectory);
            break;
        case 3:
            printf("Enter the name to search: ");
scanf("%s", name);
```

```
                Contact* result = searchContact(phoneDirectory,
name);
            if (result != NULL) {
                printf("Contact found - Name: %s, Phone Number:
%s\n", result->name, result->phoneNumber);
            } else {
                printf("Contact not found.\n");
            }
            break;
        case 4:
            printf("Enter the name to delete: ");
            scanf("%s", name);
            deleteContact(&phoneDirectory, name);
            break;
        case 5:
            printf("Enter the name to edit: ");
            scanf("%s", name);
              Contact* editResult = searchContact(phoneDirectory,
name);
            if (editResult != NULL) {
                editContact(editResult);
            } else {
                printf("Contact not found.\n");
            }
  break;
```

```
            case 6:
                // Exit the program
                break;
            default:
                printf("Invalid choice. Please enter a valid option.\n");
        }
    } while (choice != 6);

    // Free allocated memory before exiting
    while (phoneDirectory != NULL) {
        Contact* temp = phoneDirectory;
        phoneDirectory = phoneDirectory->next;
        free(temp);
    }

    return 0;
}
```