Project Report

on

PLANENT SIMULATOR

Submitted to:

Manav Rachna International Institute of Research & Studies,

Faridabad (Haryana)

In Partial Fulfilment of

BACHELOR OF COMPUTER APPLICATIONS (BCA)

Submitted by:

Leesha Arora

(25/SCA/MCAN/061)

Rakshit Sharma

(25/SCA/MCAN/060)

Faiza

(25/SCA/MCAN/058)

Aman Kumar Akhil

(25/SCA/MCAN/059)

SUBMITED TO:

Dr. Sachin Sharma(Professor)

SCHOOL OF COMPUTER APPLICATIONS (SCA)

Manav Rachna International Institute of Research & Studies

Sector-43, Aravalli Hills

Faridabad – 121001

INTRODUCTION

This project demonstrates how planets move through zodiac signs using a queue-based data structure. Each planet is enqueued and assigned a sign, then moved one by one to simulate cosmic transitions.

The **Queue-Driven Planet Movement Simulation** is a mini-project based on Data Structures that demonstrates how planets move across zodiac signs using **queue operations**. Instead of performing astronomical calculations, the system simplifies planetary motion by placing each planet in a queue representing its current zodiac sign. As time progresses, the program **dequeues** a planet from its present sign and **enqueues** it into the next sign, creating a flow similar to real movement.

This project helps in understanding how queues manage information in a **First-In-First-Out (FIFO)** manner and how they can be used to model real-life sequential systems. It strengthens concepts of **C programming**, **linear queues**, **circular queues**, and structured logical processing.

PROBLEM STATEMENT

Understanding real planetary movement requires complex mathematics and astrophysical formulas, which may be difficult at the beginner level. Therefore, there is a need for a simplified representation that students can easily comprehend.

The problem focuses on creating a model where planets move from one zodiac sign to another using a structured approach. Since queues naturally maintain order, they are ideal for representing the sequence of movement.

The challenge is to design a system where this movement happens automatically through queue operations, ensuring every planet transitions smoothly and accurately across signs.

OBJECTIVE

- To build a simulation that demonstrates how planets move through a circular path using FIFO principles.
- To teach the application of queue data structures in organizing dynamic processes.
- To help students understand event scheduling and step-by-step transitions using C programming.
- To create a system that is simple yet realistic enough to represent continuous motion.
- To provide scope for expanding the project into interactive visual models using front-end technologies.
- To develop logical thinking and improve the ability to map real-world processes through programming concepts.

SCOPE OF PROJECT

The scope covers both the theoretical understanding of queues and their practical implementation. It allows students to work on:

- Console-based simulation (C language)
- Visual-based simulation (HTML, CSS, and JavaScript)
- Understanding cyclical transitions using circular queues
- Extending the program to include delays, animations, or interactive controls
- Using linked queues for dynamically sized planet lists
  The project can be further enhanced to include real-time data or performance-based comparisons, making it suitable for advanced coursework as well.

METHODOLOGY

The working process of the system is structured in the following steps:

1. **Initialization** – All zodiac signs and planets are assigned queues or nodes.
2. **Planet Assigning** – Each planet is placed into a zodiac queue as its starting point.
3. **Simulation Start** – During each cycle, planets shift using dequeue and enqueue operations.
4. **Circular Transition** – After reaching the last zodiac sign, the planet moves back to the first (circular queue concept).
5. **Display** – The system prints updated positions after each movement.
   This creates a smooth flow resembling real planetary cycles, making it easy to observe changes at every iteration.

SYSTEM DESIGN

6.1 Data Structures Used

The primary data structure used is the **Queue**, which ensures orderly progression of elements. The queue can be implemented using:

- **Arrays** (simple linear queue)
- **Linked Lists** (dynamic queue)
- **Circular Queue** (ideal for continuous movement)

6.2 System Components

- **Planet Nodes**: Stores planet names or IDs
- **Zodiac Queues**: Each sign holds the planets currently in that sign
- **Movement Controller**: Handles transitions between signs
- **Display Function**: Shows the current arrangement of the solar system

CODE OF THE PROJECT

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX 12


typedef struct {

char signs[MAX][20];

int front, rear;

} Queue;


void initQueue(Queue *q) {

q->front = -1;

q->rear = -1;


char zodiac[MAX][20] = {

"Aries", "Taurus", "Gemini", "Cancer", "Leo", "Virgo",

"Libra", "Scorpio", "Sagittarius", "Capricorn", "Aquarius", "Pisces"

};


for(int i = 0; i < MAX; i++)

strcpy(q->signs[i], zodiac[i]);


q->front = 0;

q->rear = MAX - 1;

}


char* movePlanet(Queue *q, int steps) {

for(int i = 0; i < steps; i++) {
```

```
    char first[20];
    strcpy(first, q->signs[q->front]);

    for(int j = 0; j < MAX - 1; j++)
    strcpy(q->signs[j], q->signs[j+1]);

    strcpy(q->signs[q->rear], first);
    }

    return q->signs[q->front];
    }

    int main() {
    Queue zodiacQueue;
    initQueue(&zodiacQueue);

    int steps;
    printf("Enter number of steps to move planet: ");
    scanf("%d", &steps);

    char* result = movePlanet(&zodiacQueue, steps);
    printf("Planet moved to: %s\n", result);

    return 0;
}
```

WORKING

The Queue-Driven Planet Movement Simulation works by treating each zodiac sign as a queue. Every planet is placed in its current zodiac queue. During each step of the simulation, the planet at the front of each queue is dequeued (removed) and then enqueued (added) to the next zodiac sign. Since zodiac signs follow a circular order, the last sign connects back to the first. This process repeats for every cycle, showing how planets move one by one across all zodiac signs.

OUTPUT

Output 1: Initial State

- This output shows the **starting arrangement of the planets** before any movement begins.
- All planets (Sun, Moon, Mars, Venus, Mercury, Jupiter, Saturn) are placed in their **original zodiac sign positions**.
- This represents **Step 0**, meaning no rotation or shifting has been applied yet.
- It visually displays how the queue initially stores the planets in order.

Purpose:

To show the *base position* of planets before simulation starts.

Output 2: After 2 Steps

- This output shows how the planets look **after rotating the queue by 2 steps**.
- Each step means the **front planet moves to the back**, simulating movement across zodiac signs.
- After 2 steps, the order of planets is changed compared to the initial state.
- This demonstrates how the **Queue Data Structure** works to model movement.

Purpose:

To show that your simulator correctly performs rotations and updates the planet positions after a number of steps entered by the user.

Planet Movement Simulator

Enter steps: 2 Simulate

Planet: Mercury

Gemini Cancer Leo II Virgo Libra Scorpiaarius Capricorn Aquarus Pisces Aries Taurus

Planet Movement Simulator

Enter steps: 4 Simulate

Planet: Mercury

Taurus Gemini Cancer Leo Virgo Scorpiarius Capricorn Aquarius Pisces Aries Aries

CONCLUSION

The **Queue-Driven Planet Movement Simulation** successfully demonstrates how queue data structures can be used to model continuous and ordered processes. By representing each zodiac sign as a queue and moving planets using enqueue and dequeue operations, the project simplifies complex planetary motion into an easy-to-understand logical flow. This approach not only strengthens the understanding of **FIFO principles** but also shows how real-world systems can be simulated using basic data structure concepts. Overall, the project provides a clear, structured, and educational way to observe cyclic movement, making it a valuable learning tool for students studying C programming and Data Structures.