**Batch: E-2**

**Roll No.: 16010123325**

**Experiment / assignment / tutorial No. 5**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

**Title:** Implementation of Queue operations (Static and Dynamic implementation)- Queue, circular queue, priority queue, and deque

**Objective:** To implement Basic Operations of Queues

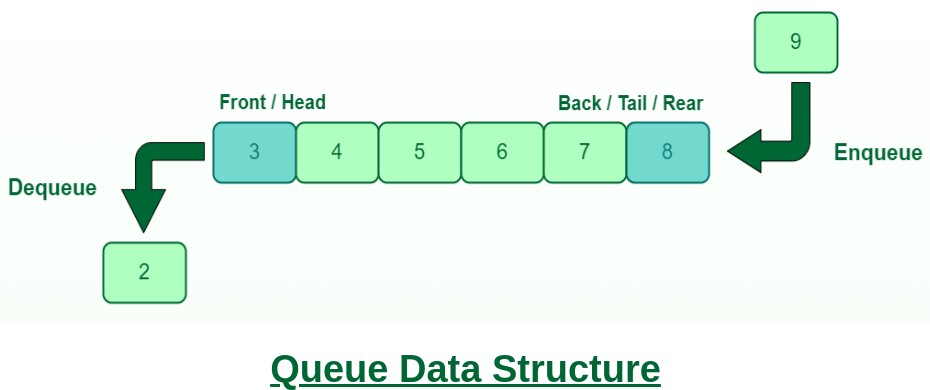
# Expected Outcome of Experiment:

|  |  |
| --- | --- |
| **CO** | **Outcome** |
| **2** | Apply linear and non-linear data structure in application development. |

**Books/ Journals/ Websites referred:**

# Introduction:

*(diagram of queue)*



# Program source code:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_NAME 100

typedef struct Node {

    char name[MAX\_NAME];

    struct Node\* next;

} Node;

typedef struct Queue {

    Node\* head;

    Node\* tail;

} Queue;

Node\* createNode(char\* name) {

    Node\* node = malloc(sizeof(Node));

    if (!node) {

        printf("Memory Error");

        return NULL;

    }

    strcpy(node->name, name);

    node->next = NULL;

    return node;

}

void enqueue(Queue\* q, char\* name) {

    Node\* node = createNode(name);

    if (q->tail == NULL) {

        q->head = node;

        q->tail = node;

    } else {

        q->tail->next = node;

        q->tail = node;

    }

    q->tail->next = q->head;

}

void dequeue(Queue\* q) {

    if (q->head == NULL) {

        printf("Queue is empty\n");

        return;

    }

    Node\* temp = q->head;

    if (q->head == q->tail) {

        q->head = NULL;

        q->tail = NULL;

    } else {

        q->head = q->head->next;

        q->tail->next = q->head;

    }

    free(temp);

}

int isEmpty(Queue\* q) {

    return (q->head == NULL);

}

void printQueue(Queue\* q) {

    Node\* temp = q->head;

    if (temp == NULL) {

        printf("Queue is empty\n");

        return;

    }

    do {

        printf("%s ", temp->name);

        temp = temp->next;

    } while (temp != q->head);

    printf("\n");

}

void josephus(Queue\* q, int m) {

    Node\* p = q->head;

    Node\* prev = q->head;

    while (p->next != p) {

        int cnt = 1;

        while (cnt != m) {

            prev = p;

            p = p->next;

            cnt++;

        }

        prev->next = p->next;

        p = prev->next;

    }

    printf("The Last person Standing is %s\n", p->name);

}

int main() {

    int n, m;

    printf("Enter the number of people: ");

    scanf("%d", &n);

    printf("Enter the position to kill: ");

    scanf("%d", &m);

    Queue\* q = malloc(sizeof(Queue));

    q->head = NULL;

    q->tail = NULL;

    for (int i = 1; i <= n; ++i) {

        char name[MAX\_NAME];

        printf("Enter Name %d:\n ", i);

        scanf("%s", name);

        enqueue(q, name);

    }

    printf("Initial Queue: ");

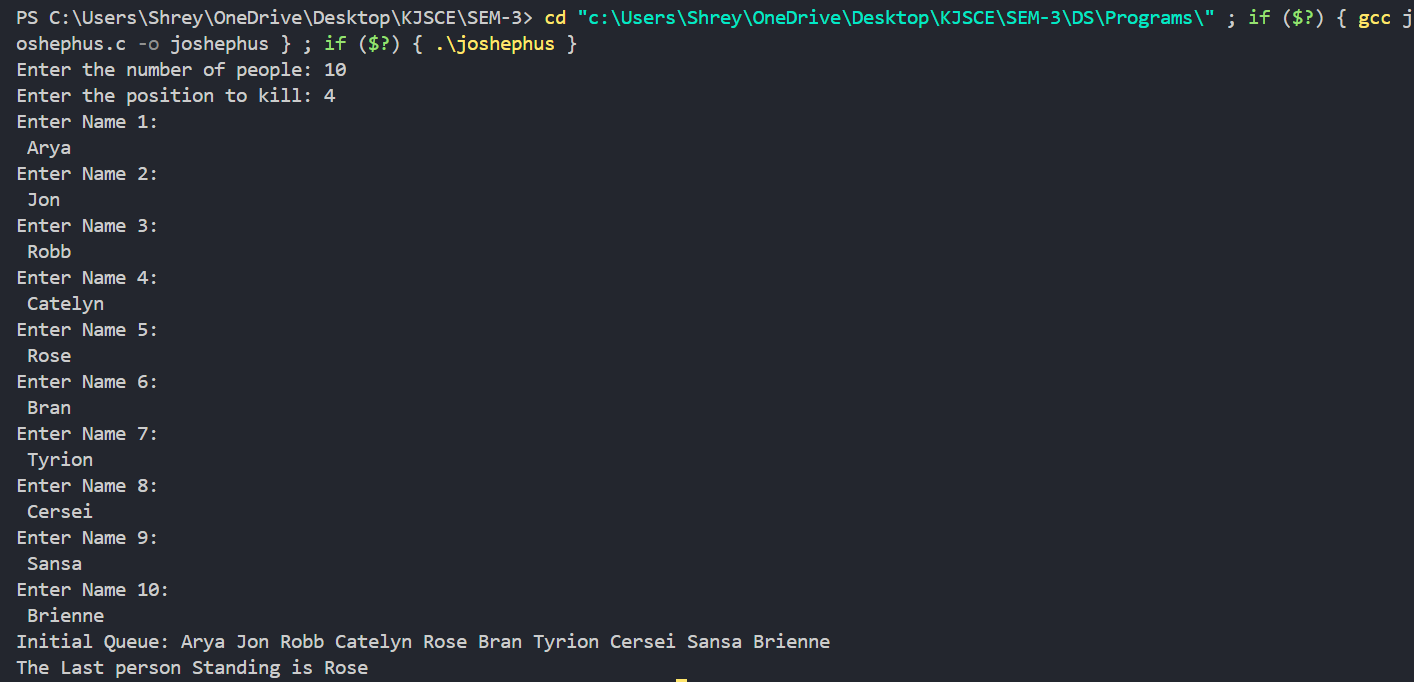
    printQueue(q);

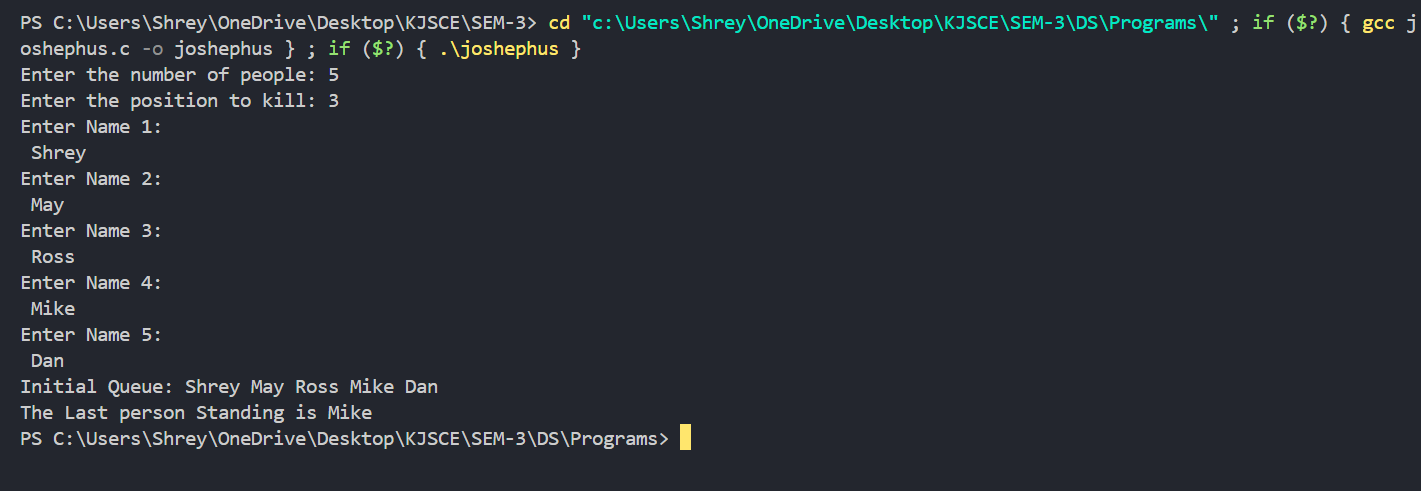
    josephus(q, m);

    return 0;

}

**Output Screenshots:**

****

****

# Conclusion:-

The above program highlights implementation of Max Priority Queue in C using arrays, and showcases its various functions like enqueue, dequeue.

# Post lab questions:

1. Implement Queue using 2 stacks. Show the working of Enqueue, Dequeue, display queue operations.

Code:

#include <stdio.h>

#include <stdlib.h>

#define MAX 100

typedef struct {

    int d[MAX];

    int t;

} S;

typedef struct {

    S s1;

    S s2;

} Q;

void initS(S\* s) {

    s->t = -1;

}

int isFull(S\* s) {

    return s->t == MAX - 1;

}

int isEmpty(S\* s) {

    return s->t == -1;

}

void push(S\* s, int x) {

    if (!isFull(s)) {

        s->d[++(s->t)] = x;

    } else {

        printf("Stack is full.\n");

    }

}

int pop(S\* s) {

    if (!isEmpty(s)) {

        return s->d[(s->t)--];

    }

    printf("Stack is empty.\n");

    return -1;

}

void initQ(Q\* q) {

    initS(&q->s1);

    initS(&q->s2);

}

void enq(Q\* q, int x) {

    push(&q->s1, x);

}

int deq(Q\* q) {

    if (isEmpty(&q->s2)) {

        while (!isEmpty(&q->s1)) {

            push(&q->s2, pop(&q->s1));

        }

    }

    return pop(&q->s2);

}

void dispQ(Q\* q) {

    if (isEmpty(&q->s2)) {

        while (!isEmpty(&q->s1)) {

            push(&q->s2, pop(&q->s1));

        }

    }

    if (isEmpty(&q->s2)) {

        printf("Queue is empty.\n");

        return;

    }

    printf("Queue elements: ");

    for (int i = q->s2.t; i >= 0; i--) {

        printf("%d ", q->s2.d[i]);

    }

    printf("\n");

}

int main() {

    Q q;

    initQ(&q);

    enq(&q, 10);

    enq(&q, 20);

    enq(&q, 30);

    dispQ(&q);

    printf("Dequeued: %d\n", deq(&q));

    dispQ(&q);

    enq(&q, 40);

    dispQ(&q);

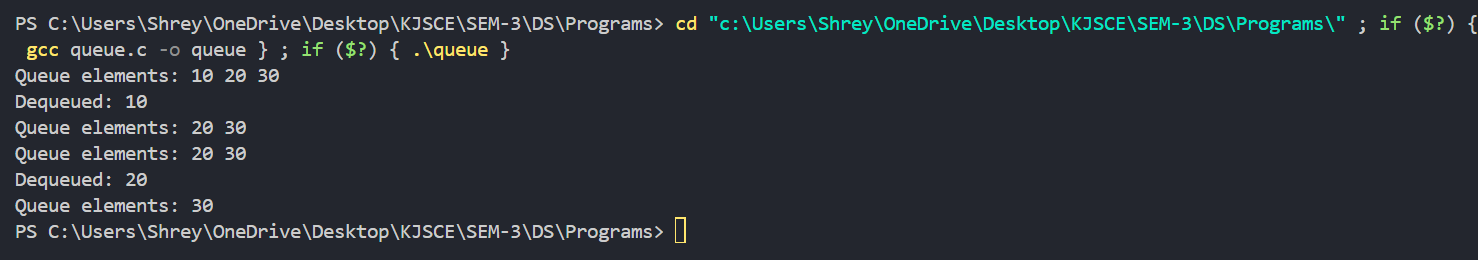
    printf("Dequeued: %d\n", deq(&q));

    dispQ(&q);

    return 0;

}

Output:



1. Discuss how different types of queues can be used in real-world applications, such as job scheduling, CPU task management, and customer service systems.

# Job Scheduling

**Simple Queues**: Used in first-come, first-served (FCFS) scheduling, where jobs are processed in the order they arrive.

# CPU Task Management

**Multi-Level Queues**: Manage different types of tasks (e.g., foreground vs. background) using various scheduling algorithms.

# Customer Service Systems

**Priority Queues**: Handle urgent calls or requests first, such as in emergency services or tech support.