**EXPERIMENT –5 [Implementation of Queue Using Linked List & Arrays]**

1. Implement various functionalities of Queue using Arrays. For example: insertion, deletion, front element, rear element etc.

**Source code:**

#include <stdio.h>

#include <stdlib.h>

struct queue

{

    int size;

    int f;

    int r;

    int \*arr;

};

void enqueue(struct queue \*ptr, int value)

{

    if (isFull(ptr))

    {

        printf("the queue is full");

    }

    else

    {

        ptr->r = ptr->r + 1;

        ptr->arr[ptr->r] = value;

    }

}

int dequeue(struct queue \*ptr)

{

    int a = -1;

    if (isEmpty(ptr))

    {

        printf("your queue is empty");

    }

    else

    {

     ptr->f = ptr->f + 1;

        a = ptr->arr[ptr->f];

    }

    return a;

}

int isFull(struct queue \*ptr)

{

    if (ptr->r == ptr->size - 1)

    {

        printf("the queue is full");

        return 1;

    }

    return 0;

}

int isEmpty(struct queue \*ptr)

{

    if (ptr->r == ptr->f)

    {

        printf("the queue is empty");

        return 1;

    }

    return 0;

}

int main()

{

    struct queue \*ptr;

    ptr->size = 10;

    ptr->f = ptr->r = -1;

    ptr->arr = (int \*)malloc(ptr->size \* sizeof(int));

    enqueue(ptr, 2); // we write &q because it takes pointer

    enqueue(ptr, 4);

    printf("dequeue element is:%d \n", dequeue(ptr));

    return 0;

}

**Output:**

dequeue element is:2

2. Implement various functionalities of Queue using Linked Lists. Again, you can implement operation given above.

**Source code:**

#include <stdio.h>

#include <stdlib.h>

struct Node \*f = NULL;

struct Node \*r = NULL;

struct Node

{

    int data;

    struct Node \*next;

};

void Traversal(struct Node \*ptr)

{

    printf("Printing the elements of this linked list\n");

    while (ptr != NULL)

    {

        printf("Element: %d\n", ptr->data);

        ptr = ptr->next;

    }

}

void enqueue(int val)

{

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

    if(n==NULL){

        printf("Queue is Full");

    }

    else{

        n->data = val;

        n->next = NULL;

        if(f==NULL){

            f=r=n;

        }

        else{

            r->next = n;

            r=n;

        }

    }

}

int dequeue()

{

    int val = -1;

    struct Node \*ptr = f;

    if(f==NULL){

        printf("Queue is Empty\n");

    }

    else{

        f = f->next;

        val = ptr->data;

        free(ptr);

    }

    return val;

}

int main()

{

    Traversal(f);

    printf("Dequeuing element %d\n", dequeue());

    enqueue(34);

    enqueue(35);

    enqueue(56);

    printf("Dequeuing element %d\n", dequeue());

    Traversal(f);

    return 0;

}

**Output:**

Printing the elements of this linked list

Queue is Empty

Dequeuing element -1

Dequeuing element 34

Printing the elements of this linked list

Element: 35

Element: 56

3. Implement Priority Queue, where every element has a priority associated with it. Perform operations like Insertion and Deletion in a priority queue.

**Source code:**

#include <stdio.h>

#include <stdlib.h>

struct PriorityQueueNode {

    int data;

    int priority;

};

struct PriorityQueue {

    struct PriorityQueueNode\* queue;

    int capacity;

    int size;

};

struct PriorityQueue\* createPriorityQueue(int capacity) {

    struct PriorityQueue\* pq = (struct PriorityQueue\*)malloc(sizeof(struct PriorityQueue));

    pq->queue = (struct PriorityQueueNode\*)malloc(sizeof(struct PriorityQueueNode) \* capacity);

    pq->capacity = capacity;

    pq->size = 0;

    return pq;

}

void swap(struct PriorityQueueNode\* a, struct PriorityQueueNode\* b) {

    struct PriorityQueueNode temp = \*a;

    \*a = \*b;

    \*b = temp;

}

void heapify(struct PriorityQueue\* pq, int i) {

    int largest = i;

    int left = 2 \* i + 1;

    int right = 2 \* i + 2;

    if (left < pq->size && pq->queue[left].priority > pq->queue[largest].priority)

        largest = left;

    if (right < pq->size && pq->queue[right].priority > pq->queue[largest].priority)

        largest = right;

    if (largest != i) {

        swap(&pq->queue[i], &pq->queue[largest]);

        heapify(pq, largest);

    }

}

void insert(struct PriorityQueue\* pq, int data, int priority) {

    if (pq->size == pq->capacity) {

        printf("Priority Queue is full. Cannot insert.\n");

        return;

    }

    struct PriorityQueueNode newNode;

    newNode.data = data;

    newNode.priority = priority;

    int i = pq->size;

    pq->queue[i] = newNode;

    pq->size++;

    while (i > 0 && pq->queue[(i - 1) / 2].priority < pq->queue[i].priority) {

        swap(&pq->queue[i], &pq->queue[(i - 1) / 2]);

        i = (i - 1) / 2;

    }

}

int extractMax(struct PriorityQueue\* pq) {

    if (pq->size == 0) {

        printf("Priority Queue is empty. Cannot extract.\n");

        return -1; // Return a sentinel value to indicate an error

    }

    if (pq->size == 1) {

        pq->size--;

        return pq->queue[0].data;

    }

    int root = pq->queue[0].data;

    pq->queue[0] = pq->queue[pq->size - 1];

    pq->size--;

    heapify(pq, 0);

    return root;

}

int main() {

    struct PriorityQueue\* pq = createPriorityQueue(10);

    insert(pq, 10, 3);

    insert(pq, 20, 2);

    insert(pq, 30, 4);

    insert(pq, 40, 1);

    printf("Highest priority element: %d\n", extractMax(pq));

    printf("Highest priority element: %d\n", extractMax(pq));

    free(pq->queue);

    free(pq);

    return 0;

}

**Output:**

**Highest priority element: 30**

**Highest priority element: 10**

4. Implement Double Ended Queue that supports following operation:

a. insertFront(): Adds an item at the front of Deque

b. insertLast(): Adds an item at the rear of Deque.

c. deleteFront(): Deletes an item from the front of Deque.

d. deleteLast(): Deletes an item from the rear of Deque.

**Source code:**

#include <stdio.h>

#include <stdlib.h>

struct Deque {

    int \*arr;

    int front,rear;

    int size;

};

struct Deque\* createDeque() {

    struct Deque\* deque = (struct Deque\*)malloc(sizeof(struct Deque));

    deque->front = -1;

    deque->rear = -1;

    return deque;

}

int isEmpty(struct Deque\* deque) {

    return (deque->front == -1);

}

int isFull(struct Deque\* deque) {

    return ((deque->front == 0 && deque->rear == deque-> size - 1) || deque->front == deque->rear + 1);

}

void insertFront(struct Deque\* deque, int item) {

    if (isFull(deque)) {

        printf("Deque is full. Cannot insert at the front.\n");

        return;

    }

    if (deque->front == -1) {

        deque->front = 0;

        deque->rear = 0;

    } else if (deque->front == 0) {

        deque->front = deque-> size - 1;

    } else {

        deque->front--;

    }

    deque->arr[deque->front] = item;

}

void insertLast(struct Deque\* deque, int item) {

    if (isFull(deque)) {

        printf("Deque is full. Cannot insert at the rear.\n");

        return;

    }

    if (deque->front == -1) {

        deque->front = 0;

        deque->rear = 0;

    } else if (deque->rear == deque-> size - 1) {

        deque->rear = 0;

    } else {

        deque->rear++;

    }

    deque->arr[deque->rear] = item;

}

void deleteFront(struct Deque\* deque) {

    if (isEmpty(deque)) {

        printf("Deque is empty. Cannot delete from the front.\n");

        return;

    }

    if (deque->front == deque->rear) {

        deque->front = -1;

        deque->rear = -1;

    } else if (deque->front == deque-> size - 1) {

        deque->front = 0;

    } else {

        deque->front++;

    }

}

void deleteLast(struct Deque\* deque) {

    if (isEmpty(deque)) {

        printf("Deque is empty. Cannot delete from the rear.\n");

        return;

    }

    if (deque->front == deque->rear) {

        deque->front = -1;

        deque->rear = -1;

    } else if (deque->rear == 0) {

        deque->rear = deque-> size - 1;

    } else {

        deque->rear--;

    }

}

void display(struct Deque\* deque) {

    if (isEmpty(deque)) {

        printf("Deque is empty.\n");

        return;

    }

    int i;

    if (deque->front <= deque->rear) {

        for (i = deque->front; i <= deque->rear; i++) {

            printf("%d ", deque->arr[i]);

        }

    } else {

        for (i = deque->front; i < deque->size; i++) {

            printf("%d ", deque->arr[i]);

        }

        for (i = 0; i <= deque->rear; i++) {

            printf("%d ", deque->arr[i]);

        }

    }

    printf("\n");

}

int main() {

    struct Deque\* deque = createDeque();

    deque->size=10;

    insertFront(deque, 1);

    insertFront(deque, 2);

    insertLast(deque, 3);

    insertLast(deque, 4);

    printf("Deque: ");

    display(deque);

    deleteFront(deque);

    deleteLast(deque);

    printf("Deque after deleting front and rear elements: ");

    display(deque);

    return 0;

}

**Output:**

Deque: 2 1 3 4

Deque after deleting front and rear elements: 1 3

5. Implement Double Ended Queue that supports following operation:

a. getFront(): Gets the front item from the queue.

b. getRear(): Gets the last item from queue.

c. isEmpty(): Checks whether Deque is empty or not.

d. isFull(): Checks whether Deque is full or not.

**source code:**

#include <stdio.h>

#include <stdbool.h>

#define MAX\_SIZE 100

struct Deque {

    int arr[MAX\_SIZE];

    int front, rear, size;

};

void initializeDeque(struct Deque \*deque) {

    deque->front = -1;

    deque->rear = 0;

    deque->size = 0;

}

bool isFull(struct Deque \*deque) {

    return (deque->size == MAX\_SIZE);

}

bool isEmpty(struct Deque \*deque) {

    return (deque->size == 0);

}

void insertFront(struct Deque \*deque, int data) {

    if (isFull(deque)) {

        printf("Deque is full. Cannot insert.\n");

        return;

    }

    if (deque->front == -1)

        deque->front = 0;

    deque->front = (deque->front - 1 + MAX\_SIZE) % MAX\_SIZE;

    deque->arr[deque->front] = data;

    deque->size++;

}

void insertRear(struct Deque \*deque, int data) {

    if (isFull(deque)) {

        printf("Deque is full. Cannot insert.\n");

        return;

    }

    deque->rear = (deque->rear + 1) % MAX\_SIZE;

    deque->arr[deque->rear] = data;

    if (deque->front == -1)

        deque->front = 0;

    deque->size++;

}

int getFront(struct Deque \*deque) {

    if (isEmpty(deque)) {

        printf("Deque is empty.\n");

        return -1;

    }

    return deque->arr[deque->front];

}

int getRear(struct Deque \*deque) {

    if (isEmpty(deque)) {

        printf("Deque is empty.\n");

        return -1;

    }

    return deque->arr[deque->rear];

}

int main() {

    struct Deque deque;

    initializeDeque(&deque);

    insertRear(&deque, 1);

    insertRear(&deque, 2);

    insertFront(&deque, 0);

    printf("Front: %d\n", getFront(&deque));

    printf("Rear: %d\n", getRear(&deque));

    return 0;

}

**Output:**

Front: 0

Rear: 2