**Ds day6**

1. **Write a program to implement Circular Queue with insertion and deletion operations**.

#include <stdio.h>

#include <stdlib.h>

#define SIZE 5

int front = -1, rear = -1;

int queue[SIZE];

void enqueue(int value) {

if ((front == 0 && rear == SIZE - 1) || (rear == (front - 1) % (SIZE - 1))) {

printf("Queue is full\n");

return;

} else if (front == -1) {

front = rear = 0;

queue[rear] = value;

} else if (rear == SIZE - 1 && front != 0) {

rear = 0;

queue[rear] = value;

} else {

rear++;

queue[rear] = value;

}

}

int dequeue() {

if (front == -1) {

printf("Queue is empty\n");

return -1;

}

int data = queue[front];

queue[front] = -1;

if (front == rear) {

front = rear = -1;

} else if (front == SIZE - 1) {

front = 0;

} else {

front++;

}

return data;

}

void display() {

if (front == -1) {

printf("Queue is empty\n");

return;

}

printf("Elements in Circular Queue are: ");

if (rear >= front) {

for (int i = front; i <= rear; i++) {

printf("%d ", queue[i]);

}

} else {

for (int i = front; i < SIZE; i++) {

printf("%d ", queue[i]);

}

for (int i = 0; i <= rear; i++) {

printf("%d ", queue[i]);

}

}

printf("\n");

}

int main() {

enqueue(1);

enqueue(2);

enqueue(3);

enqueue(4);

enqueue(5);

display();

dequeue();

dequeue();

display();

enqueue(6);

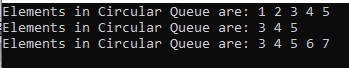
enqueue(7);

display();

return 0;

}

Output:



1. **Write a program to implement Double Ended Queue with insertion and deletion operations.**

#include <stdio.h>

#include <stdlib.h>

#define MAX\_SIZE 5

int deque[MAX\_SIZE];

int front = -1, rear = -1;

void insertFront(int item) {

if ((front == 0 && rear == MAX\_SIZE - 1) || (front == rear + 1)) {

printf("Queue Overflow\n");

return;

}

if (front == -1) {

front = 0;

rear = 0;

} else if (front == 0) {

front = MAX\_SIZE - 1;

} else {

front = front - 1;

}

deque[front] = item;

printf("%d inserted at front\n", item);

}

void insertRear(int item) {

if ((front == 0 && rear == MAX\_SIZE - 1) || (front == rear + 1)) {

printf("Queue Overflow\n");

return;

}

if (rear == -1) {

front = 0;

rear = 0;

} else if (rear == MAX\_SIZE - 1) {

rear = 0;

} else {

rear = rear + 1;

}

deque[rear] = item;

printf("%d inserted at rear\n", item);

}

void deleteFront() {

if (front == -1) {

printf("Queue Underflow\n");

return;

}

printf("%d deleted from front\n", deque[front]);

if (front == rear) {

front = -1;

rear = -1;

} else if (front == MAX\_SIZE - 1) {

front = 0;

} else {

front = front + 1;

}

}

void deleteRear() {

if (rear == -1) {

printf("Queue Underflow\n");

return;

}

printf("%d deleted from rear\n", deque[rear]);

if (front == rear) {

front = -1;

rear = -1;

} else if (rear == 0) {

rear = MAX\_SIZE - 1;

} else {

rear = rear - 1;

}

}

void display() {

int i;

if (front == -1) {

printf("Queue is empty\n");

return;

}

printf("Elements in the Queue are: ");

if (front <= rear) {

for (i = front; i <= rear; i++) {

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

}

} else {

for (i = front; i < MAX\_SIZE; i++) {

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

}

for (i = 0; i <= rear; i++) {

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

}

}

printf("\n");

}

int main() {

insertFront(5);

insertRear(10);

insertRear(15);

display();

deleteFront();

display();

insertFront(20);

insertRear(25);

display();

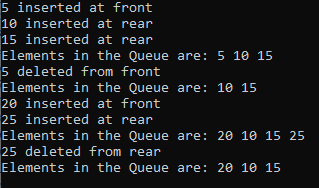
deleteRear();

display();

return 0;

}

Output:



1. **Write a program to implement Priority Queue with insertion and deletion** operations.

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

int priority;

struct Node\* next;

};

struct Node\* createNode(int data, int priority) {

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

newNode->data = data;

newNode->priority = priority;

newNode->next = NULL;

return newNode;

}

void insert(struct Node\*\* head, int data, int priority) {

struct Node\* newNode = createNode(data, priority);

if (\*head == NULL || (\*head)->priority < priority) {

newNode->next = \*head;

\*head = newNode;

} else {

struct Node\* current = \*head;

while (current->next != NULL && current->next->priority >= priority) {

current = current->next;

}

newNode->next = current->next;

current->next = newNode;

}

}

int delete(struct Node\*\* head) {

if (\*head == NULL) {

printf("Queue is empty\n");

return -1;

}

struct Node\* temp = \*head;

int data = temp->data;

\*head = (\*head)->next;

free(temp);

return data;

}

void display(struct Node\* head) {

while (head != NULL) {

printf("Data: %d, Priority: %d\n", head->data, head->priority);

head = head->next;

}

}

int main() {

struct Node\* pq = NULL;

insert(&pq, 10, 2);

insert(&pq, 20, 1);

insert(&pq, 30, 3);

printf("Priority Queue:\n");

display(pq);

printf("Deleted: %d\n", delete(&pq));

printf("Priority Queue after deletion:\n");

display(pq);

return 0;

}

Output:

