AIM:-A C program to demonstrate linked list based implementation of queue

Algorithm:-

Insert

Step 1: If maxsize-1=rear then this means the queue is full.

Step 2: But if rear<maxsize means that we can store an element in an array.

Step 3: So increment the rear value by 1 and then insert an element at the rear index.

Delete

Step 4: If front=-1 or front>rear then no element is available to delete.

Step 5: else delete front index element.

Step 6: if rear=front then set-1 to both front and rear.

Step 7: else we increment front.

Display

Step 8: first check whether the queue is not empty.

Step 9: if empty we display that the queue is empty we simply return from the function and not execute further inside the function.

Step 10: else print all elements from front to rear.

Program:-

#include <stdio.h>

#include <stdlib.h>

// A linked list (LL) node to store a queue entry

struct QNode {

int key;

struct QNode\* next;

};

// The queue, front stores the front node of LL and rear

// stores the last node of LL

struct Queue {

struct QNode \*front, \*rear;

};

// A utility function to create a new linked list node.

struct QNode\* newNode(int k)

{

struct QNode\* temp

= (struct QNode\*)malloc(sizeof(struct QNode));

temp->key = k;

temp->next = NULL;

return temp;

}

// A utility function to create an empty queue

struct Queue\* createQueue()

{

struct Queue\* q

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

q->front = q->rear = NULL;

return q;

}

// The function to add a key k to q

void enQueue(struct Queue\* q, int k)

{

// Create a new LL node

struct QNode\* temp = newNode(k);

// If queue is empty, then new node is front and rear

// both

if (q->rear == NULL) {

q->front = q->rear = temp;

return;

}

// Add the new node at the end of queue and change rear

q->rear->next = temp;

q->rear = temp;

}

// Function to remove a key from given queue q

void deQueue(struct Queue\* q)

{

// If queue is empty, return NULL.

if (q->front == NULL)

return;

// Store previous front and move front one node ahead

struct QNode\* temp = q->front;

q->front = q->front->next;

// If front becomes NULL, then change rear also as NULL

if (q->front == NULL)

q->rear = NULL;

free(temp);

}

// Driver Program to test above functions

int main()

{

struct Queue\* q = createQueue();

enQueue(q, 10);

enQueue(q, 20);

deQueue(q);

deQueue(q);

enQueue(q, 30);

enQueue(q, 40);

enQueue(q, 50);

deQueue(q);

printf("Queue Front : %d \n", q->front->key);

printf("Queue Rear : %d", q->rear->key);

return 0;

}

void push(int ch)

{

top++;

s[top]=ch;

}

int pop()

{

int ch;

ch=s[top];

top=top-1;

return(ch);

}

Screenshot of the output:-

[data-structures-2/23\_02\_2023\_1NT21IS102\_PROGRAM 7B.docx at main · Niharika-PS/data-structures-2 (github.com)](https://github.com/Niharika-PS/data-structures-2/blob/main/23_02_2023_1NT21IS102_PROGRAM%207B.docx)