**PROGRAM-5**

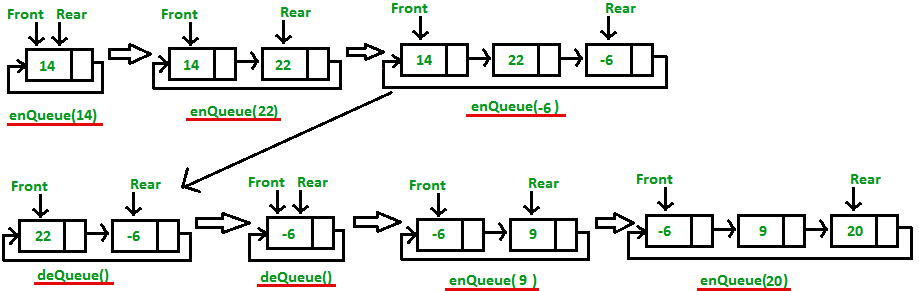
***Q. Design and implement a given type of queue in C (Ordinary queue/Circular queue) using Array implementation and linked list implementation. Also demonstrate the working with suitable inputs. Display appropriate messages in case of exceptions.***

**“Circular Queue Implementation using Linked List”**

* **Theory:**

Because there is no memory waste, using a circular queue is preferable to using a regular queue. Linked lists make it simple to create dynamic memory allocation available. Circular queues are identical to circular linked lists when implemented using linked lists, with the exception that circular queues have two pointers in the front and back, whereas circular linked lists only have one. Let's look at how to use a linked list to construct a circular queue in C programming.

A circular queue is an extended version of a linear queue as it follows the First in First Out principle with the exception that it connects the last node of a queue to its first by forming a circular link. Hence, it is also called a Ring Buffer.



* **Algorithm:**

**Enqueue ():**

Step-1: Create a struct node type node.

Step-2: Insert the given data in the new node data section and NULL in address section.

Step-3: If Queue is empty then initialize front and rear from new node.

Step-4: Queue is not empty then initialize rear next and rear from new node.

Step-5: New node next initialize from front

**Dequeue ():**

Step-1: Check if queue is empty or not.

Step-2: If queue is empty then dequeue is not possible.

Step-3: Else Initialize temp from front.

Step-4: If front is equal to the rear then initialize front and rear from null.

Step-5: Print data of temp and free temp memory.

Step-6: If there is more than one node in Queue then make front next to front then initialize rear next from front.

Step-7: Print temp and free temp.

**Display ():**

Step-1: Check if there is some data in the queue or not.

Step-2: If the queue is empty print “No data in the queue.”

Step-3: Else define a node pointer and initialize it with front.

Step-4: Print data of node pointer until the next of node pointer becomes NULL.

//--------------------------------------------------------------

* **Code:**

#include <stdio.h>

#include<stdlib.h>

struct Node {

int data;

struct Node\* next; //a structure is created

};

struct Node\* front = NULL; // front and rear are set to NULL

struct Node\* rear = NULL;

void enqueue(int x) // enqueue function with x as a paramter

{

struct Node \*ptr; //ptr pointer is declared.

//memory is allocated dynamically for ptr by malloc.

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

ptr->data=x; //x is put to data part of ptr

ptr->next=NULL; //link part of ptr is made NULL

if(rear== NULL)

{

front=rear=ptr; //rear and front are pointed to ptr

rear->next=front; // rear’s next part is connected back to front circularly.

}

else

{

rear->next=ptr;

rear=ptr; //ptr is pointed to rear

rear->next=front; // rear’s next part is connected back to front circularly.

}

printf("value inserted\n"); //value is inserted.

}

//------------------------------------------------------------

void dequeue() //delete function

{

struct Node \*temp= front; //temp pointer is initialized and pointed to front

if((front==NULL)&&(rear==NULL))

{

printf("\nQueue is empty"); //undeflow

}

else if(front==rear)

{

front=rear=NULL;

free(temp); // temp is freed.

}

else

{

front=front->next; // front is traversed looking for elements

rear->next=front; // rear’s next part is connected back to front circularly.

free(temp); //temp is freed using free ().

}

printf("Value deleted\n"); //value deleted.

}

//--------------------------------------------------------------

int peek()

{

if((front==NULL) &&(rear==NULL))

{

printf("\nQueue is empty"); // Underflow

}

else

{

printf("\nThe front element is %d", front->data);

// the dequeued element is printed on the console

}

}

//-----------------------------------------------------------------

void display() //display function

{

struct Node \*ptr= front; //pointer ptr is initilaized

printf("\n The elements in a Queue are : ");

if((front==NULL) && (rear==NULL))

{

printf("Queue is empty"); //empty queue

}

else

{

while(ptr->next!=front) //ptr is traversed along the queue

{

printf("%d\n", ptr->data); //data is printed.

ptr=ptr->next; //ptr is traversed to next node

}

}

}

//-------------------------------------------------------------

int main()

{

enqueue (34);

enqueue (10);

enqueue (23);

display ();

dequeue ();

peek ();

}

//END OF THE PROGRAM

//-----------------------------------------------------------

**“SCREENSHOTS OF OUTPUT”**

