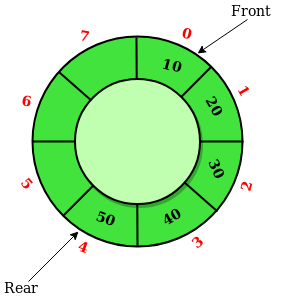
**Objective-**

Write a program to perform enqueue,dequeue & traversing operation in an Circular queue using array.

**Description –**

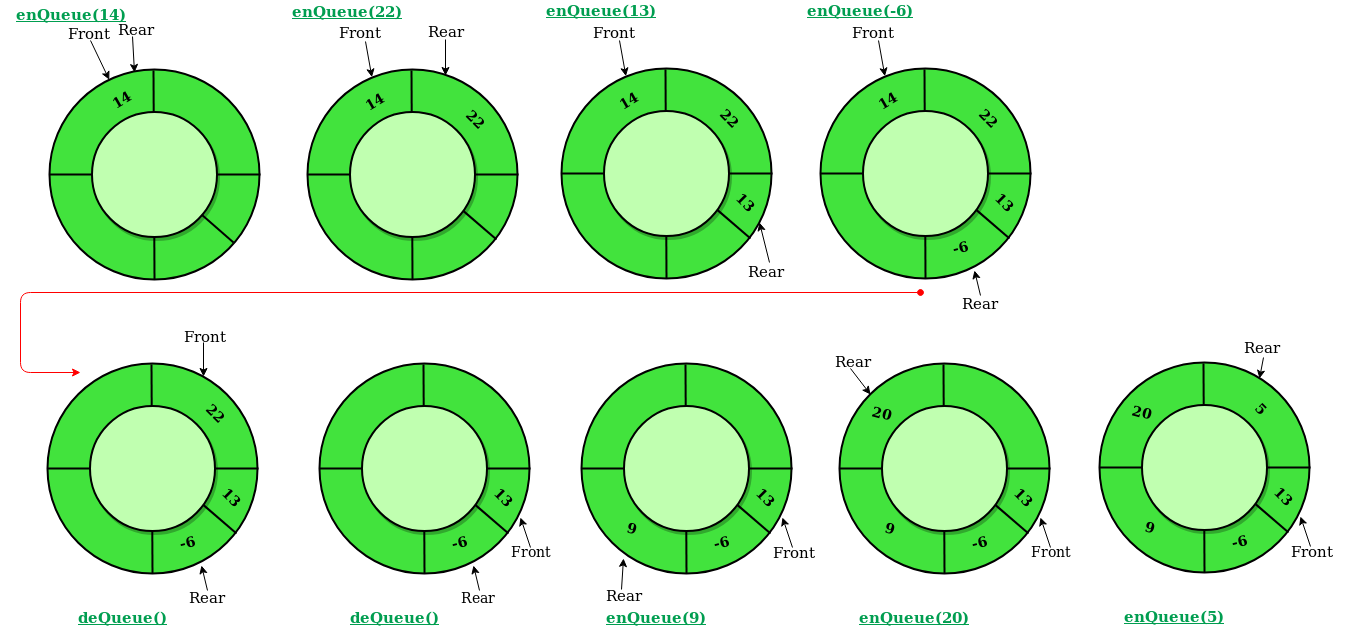
1. **Circular queue**

Circular Queue is a linear data structure in which the operations are performed based on FIFO (First In First Out) principle and the last position is connected back to the first position to make a circle. It is also called **‘Ring Buffer’**.



In a normal Queue, we can insert elements until queue becomes full. But once queue becomes full, we can not insert the next element even if there is a space in front of queue.

A circular queue solved the limitations of the normal queue. Thus making it a better pick than the normal queue. It also follows the first come first serve algorithm.



Operations on Circular Queue:

* **Front:** Get the front item from queue.
* **Rear:** Get the last item from queue.
* **enQueue(value)**This function is used to insert an element into the circular queue. In a circular queue, the new element is always inserted at Rear position.

**Steps:**

* 1. Check whether queue is Full – Check ((rear == SIZE-1 && front == 0) || (rear == front-1)).
  2. If it is full then display Queue is full. If queue is not full then, check if (rear == SIZE – 1 && front != 0) if it is true then set rear=0 and insert element.
* **deQueue()** This function is used to delete an element from the circular queue. In a circular queue, the element is always deleted from front position.

**Steps:**

* 1. Check whether queue is Empty means check (front==-1).
  2. If it is empty then display Queue is empty. If queue is not empty then step 3
  3. Check if (front==rear) if it is true then set front=rear= -1 else check if (front==size-1), if it is true then set front=0 and return the element.
* **Traversing():** this function is used to display the elements in the circular queue.

**Steps:**

**1.**check whether circular queue is empty.(front==-1)

**2.** if it is empty , then display “queue is empty”and terminate the function .

**3.**if it is not empty,then define an integer variable ‘i’ and set ‘i=front’.

**4.** check whether ‘front<=rear’,if it is ‘TRUE’ ,then display’queue[i]’ value and increment ‘i’ value by one (i++) .repeat the same until ‘i<=rear’becomes ‘FALSE’.

**5.**if ‘front<=rear’is FALSE ,then display ‘queue[i]’ value and increment ‘I’value by one (i++). Repeat the same until’i<=size-1’becomes FALSE.

**6.** set i to 0.

**7.** again display ‘cqueue[i]’ value and increment I value by one(i++) . repeat the same until ‘i<=rear’ becomes FALSE.

**Time Complexity:** Time complexity of enQueue(), deQueue() operation is O(1) as there is no loop in any of the operation.

1. Switch statement

Switch case statements are a substitute for long if statements that compare a variable to several integral values

* The switch statement is a multiway branch statement .it provides an esay way to dispatch execution to different parts of code based on the value of the expression.
* Switch is a case control statement that allows a value to change control of execution.

Syntax:

Switch(n)

{

Case 1: //code to be executed if n=1;

break;

Case 2: //code to be executed if n=2;

break;

default://code to be executed if n does’t match any cases

}

Program

#include<stdio.h> //header file

#include<conio.h>

#define SIZE 5

void enQueue(int); //function declaration

void deQueue(); //function declaration

void traverse(); //function declaration

int cQueue[SIZE], front = -1, rear = -1; //global declaration

void main()

{

int choice, value;

clrscr();

do{ //do-while loop

printf("\n operation of circular queue");

printf("1. Insert\n2. Delete\n3. traversing\n4. Exit\n");

printf("Enter your choice: ");

scanf("%d",&choice);

switch(choice) //switch case

{

case 1: printf("\nEnter the value to be insert: ");

scanf("%d",&value);

enQueue(value); //function calling

break;

case 2: deQueue(); //function calling

break;

case 3: traverse(); //function calling

break;

case 4: break;

default: printf("\nPlease select the correct choice!!!\n");

}

}while(choice!=4);

}

void enQueue(int value) //function definition

{

if((front == 0 && rear == SIZE - 1) || (front == rear+1)) //if-statement

printf("\nCircular Queue is Full! insertion is not possible!\n");

else //else statement

{

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

rear = -1;

cQueue[++rear] = value;

printf("\nInsertion Success!\n");

if(front == -1)

front = 0;

}

}

void deQueue() ////function definition

{

if(front == -1 && rear == -1)

printf("\nCircular Queue is Empty! Deletion is not possible!\n");

else{

printf("\nDeleted element : %d\n",cQueue[front++]);

if(front == SIZE)

front = 0;

if(front-1 == rear)

front = rear = -1;

}

}

void traverse() //function definition

{

if(front == -1)

printf("\nCircular Queue is Empty!\n");

else{

int i = front;

printf("\nCircular Queue Elements are :\n");

if(front <= rear){

while(i <= rear)

printf("%d\t",cQueue[i++]);

}

else{

while(i <= SIZE - 1)

printf("%d\t", cQueue[i++]);

i = 0;

while(i <= rear)

printf("%d\t",cQueue[i++]);

}

}

}

Output:

