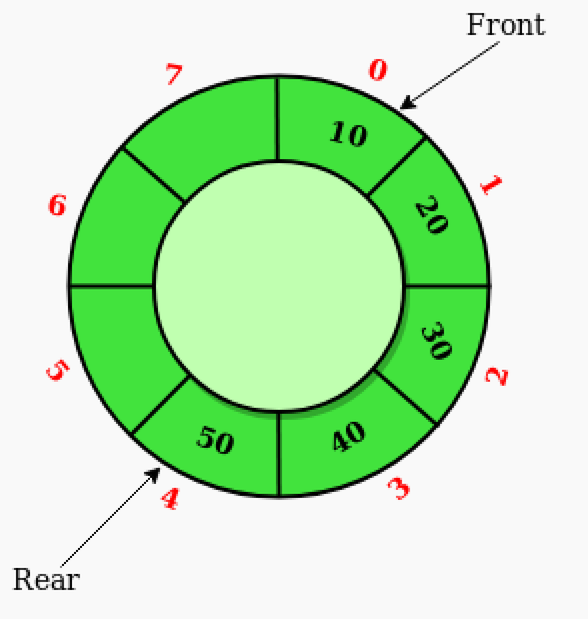
Circular Queue using Array

A Circular Queue is an extended version of a normal queue where the last element of the queue is connected to the first element of the queue forming a circle.

The operations are performed based on FIFO (First In First Out) principle. 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.

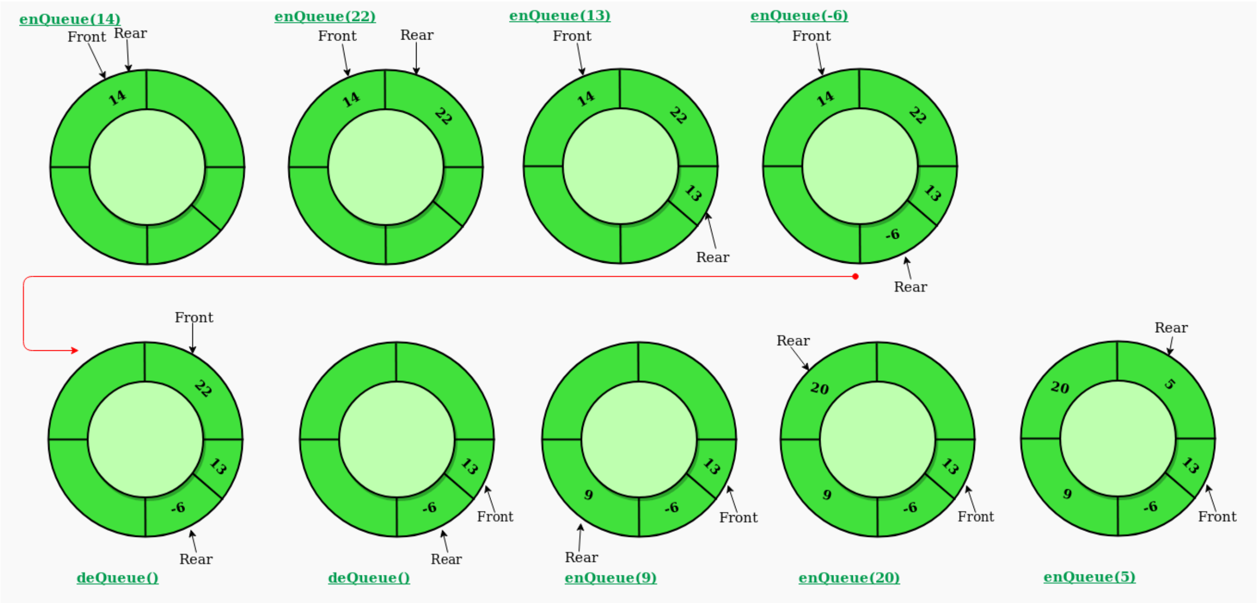
Operations on Circular Queue:

- **Front:** Get the front item from the queue.
- **Rear:** Get the last item from the 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 the rear position.
    - Check whether the queue is full – [i.e., the rear end is in just before the front end in a circular manner].
    - If it is full then display Queue is full.
        - If the queue is not full then, insert an element at the end of the queue.
- **deQueue()** This function is used to delete an element from the circular queue. In a circular queue, the element is always deleted from the front position.
    - Check whether the queue is Empty.
    - If it is empty then display Queue is empty.
        - If the queue is not empty, then get the last element and remove it from the queue.

Illustration of Circular Queue Operations:

Follow the below image for a better understanding of the enqueue and dequeue operations.



Working of Circular queue operations

Implement Circular Queue using Array:

1. Initialize an array queue of size **n**, where n is the maximum number of elements that the queue can hold.
2. Initialize two variables front and rear to -1.
3. **Enqueue:** To enqueue an element **x** into the queue, do the following:
    - Increment rear by 1.
        - If **rear** is equal to n, set **rear** to 0.
    - If **front** is -1, set **front** to 0.
    - Set queue[rear] to x.
4. **Dequeue:** To dequeue an element from the queue, do the following:
    - Check if the queue is empty by checking if **front** is -1.

    - If it is, return an error message indicating that the queue is empty.
- Set **x** to queue[front].
- If **front** is equal to **rear**, set **front** and **rear** to -1.
- Otherwise, increment **front** by 1 and if **front** is equal to n, set **front** to 0.
- Return x.

Below is the implementation of the above idea:

```
// C Program to implement the circular queue in c using arrays
#include <stdio.h>

// Define the maximum size of the queue
#define MAX_SIZE 5

    // Declare the queue array and front, rear variables
    int queue[MAX_SIZE];
int front = -1, rear = -1;
```

```
// Function to check if the queue is full
int isFull()
{
    // If the next position is the front, the queue is full
    return (rear + 1) % MAX_SIZE == front;
}

// Function to check if the queue is empty
int isEmpty()
{
    // If the front hasn't been set, the queue is empty
    return front == -1;
}

// Function to enqueue (insert) an element
void enqueue(int data)
{
    // If the queue is full, print an error message and
    // return
    if (isFull()) {
        printf("Queue overflow\n");
        return;
    }
    // If the queue is empty, set the front to the first
    // position
    if (front == -1) {
        front = 0;
    }
    // Add the data to the queue and move the rear pointer
    rear = (rear + 1) % MAX_SIZE;
    queue[rear] = data;
    printf("Element %d inserted\n", data);
}

// Function to dequeue (remove) an element
int dequeue()
{
    // If the queue is empty, print an error message and
    // return -1
    if (isEmpty()) {
```

```c
        printf("Queue underflow\n");
        return -1;
    }
    // Get the data from the front of the queue
    int data = queue[front];
    // If the front and rear pointers are at the same
    // position, reset them
    if (front == rear) {
        front = rear = -1;
    }
    else {
        // Otherwise, move the front pointer to the next
        // position
        front = (front + 1) % MAX_SIZE;
    }
    // Return the dequeued data
    return data;
}

// Function to display the queue elements
```

```c
void display()
{
    // If the queue is empty, print a message and return
    if (isEmpty()) {
        printf("Queue is empty\n");
        return;
    }
    // Print the elements in the queue
    printf("Queue elements: ");
    int i = front;
    while (i != rear) {
        printf("%d ", queue[i]);
        i = (i + 1) % MAX_SIZE;
    }
    // Print the last element
    printf("%d\n", queue[rear]);
}


// Main function
int main()
```

```
{
    // Enqueue some elements
    enqueue(10);
    enqueue(20);
    enqueue(30);
    // Display the queue
    display();

    // Dequeue an element and print it
    printf("Dequeued element: %d\n", dequeue());
    // Display the queue again
    display();

    // End of main function
    return 0;
}
```

Output

```
Element 10 inserted
Element 20 inserted

Element 30 inserted

Queue elements: 10 20 30

Dequeued element: 10

Queue elements: 20 30
```

```cpp
// C or C++ program for insertion and
// deletion in Circular Queue
#include<bits/stdc++.h>
using namespace std;

class Queue
{
    // Initialize front and rear
    int rear, front;

   // Circular Queue
    int size;
    int *arr;
public:
    Queue(int s)
    {
       front = rear = -1;
       size = s;
       arr = new int[s];
    }

   void enQueue(int value);
    int deQueue();
    void displayQueue();
};


/* Function to create Circular queue */
void Queue::enQueue(int value)
{
    if ((front == 0 && rear == size-1) ||
             ((rear+1) % size == front))
    {
        printf("\nQueue is Full");
        return;
    }

   else if (front == -1) /* Insert First Element */
    {
        front = rear = 0;
        arr[rear] = value;
    }

   else if (rear == size-1 && front != 0)
    {
        rear = 0;
        arr[rear] = value;
    }

   else
    {
        rear++;
        arr[rear] = value;
    }
}

// Function to delete element from Circular Queue
int Queue::deQueue()
{
    if (front == -1)
    {
        printf("\nQueue is Empty");
        return INT_MIN;
    }

   int data = arr[front];
    arr[front] = -1;
    if (front == rear)
    {
        front = -1;
        rear = -1;
    }
    else if (front == size-1)
        front = 0;
    else
        front++;

   return data;
}

// Function displaying the elements
// of Circular Queue
void Queue::displayQueue()
{
    if (front == -1)
    {
        printf("\nQueue is Empty");
        return;
    }
    printf("\nElements in Circular Queue are: ");
    if (rear >= front)
    {
        for (int i = front; i <= rear; i++)
            printf("%d ",arr[i]);
    }
    else
    {
        for (int i = front; i < size; i++)
            printf("%d ", arr[i]);

      for (int i = 0; i <= rear; i++)
            printf("%d ", arr[i]);
    }
}

/* Driver of the program */
int main()
```

```
{
    Queue q(5);

   // Inserting elements in Circular Queue
    q.enQueue(14);
    q.enQueue(22);
    q.enQueue(13);
    q.enQueue(-6);

   // Display elements present in Circular Queue
    q.displayQueue();

   // Deleting elements from Circular Queue
    printf("\nDeleted value = %d", q.deQueue());
    printf("\nDeleted value = %d", q.deQueue());

   q.displayQueue();

   q.enQueue(9);
    q.enQueue(20);
    q.enQueue(5);

   q.displayQueue();

   q.enQueue(20);
    return 0;
}
```

Output

```
Elements in Circular Queue are: 14 22 13 -6

Deleted value = 14

Deleted value = 22

Elements in Circular Queue are: 13 -6

Elements in Circular Queue are: 13 -6 9 20 5
```

```
Queue is Full
```