QUEUE & CIRCULAR QUEUE

QUEUE

- A Queue is a linear data structure that stores a collection of elements.
- The queue operates on first in first out (FIFO) algorithm.

QUEUE OPERATIONS

- INSERTION/ENQUEUE
- DELETION/DEQUEUE
- WHETHER EMPTY
- WHETHER FULL
- DISPLAY
- TWO POINTERS OR POSITION INDICATORS
  - FRONT/ HEAD
  - REAR/TAIL/END

IMPLEMENTING QUEUE USING ARRAYS

```
#include <stdio.h>
#define SIZE 5
void enQueue(int);
void deQueue();
void display();
int items[SIZE], front = -1, rear = -1;
```

IMPLEMENTING QUEUE-INSERTION

```
void enQueue(int value)
{
        if (rear == SIZE - 1)
                printf("\nQueue is Full!!");
        else
        {
                if (front == -1)
                        front = 0;
                rear++;
                items[rear] = value;
                printf("\nInserted -> %d", value);
        }
}
```

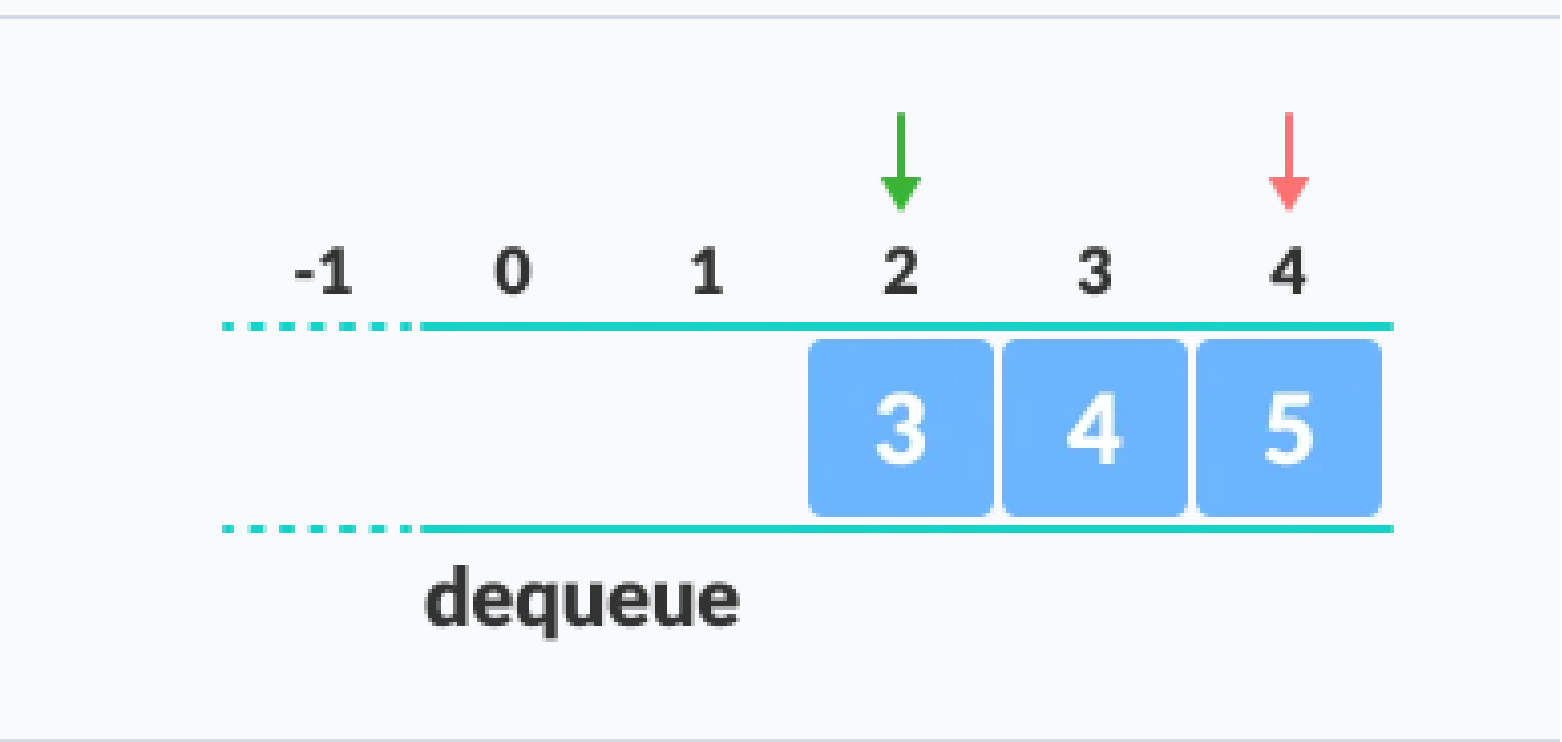
IMPLEMENTING QUEUE-DELETION

```
void deQueue()
{
        if (front == -1)
                printf("\nQueue is Empty!!");
        else
        {
                printf("\nDeleted : %d", items[front]);
                front++;
                if (front > rear)
                        front = rear = -1;
        }
}
```

IMPLEMENTING QUEUE-DISPLAY

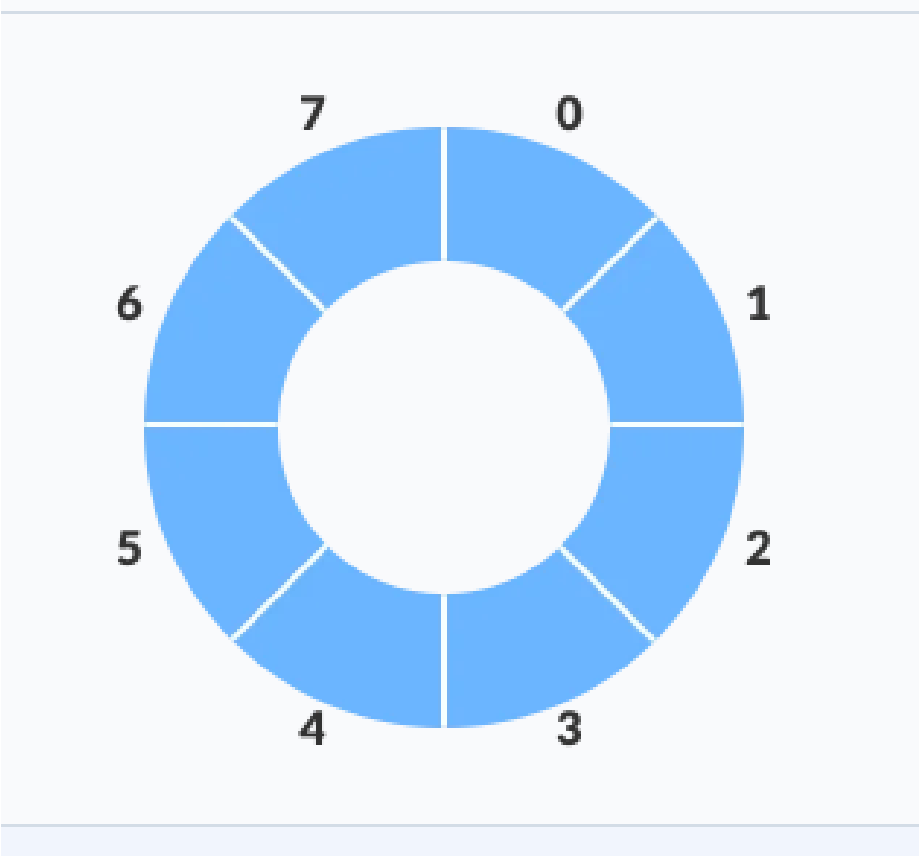
```
void display()
{
        if (rear == -1)
                printf("\nQueue is Empty!!!");
        else
        {
                int i;
                printf("\nQueue elements are:\n");
                for (i = front; i <= rear; i++)
                        printf("%d ", items[i]);
        }
        printf("\n");
}
```

LIMITATION OF QUEUE



CIRCULAR QUEUE

- A circular queue is the extended version of a regular queue where the last element is connected to the first element. Thus forming a circle-like structure.



IMPLEMENTING CIRCULAR QUEUE-FULL QUEUE

```
int isFull()
{
        if ((front == rear + 1) || (front == 0 && rear == SIZE - 1))
                return 1;
        return 0;
}
```

IMPLEMENTING CIRCULAR QUEUE-EMPTY QUEUE

```
int isEmpty()
{
        if (front == -1)
                return 1;
        return 0;
 }
```

IMPLEMENTING CIRCULAR QUEUE-INSERTION

```
void enQueue(int element)
{
        if (isFull())
                printf("\n Queue is full!! \n");
        else
        {
                if (front == -1)
                front = 0;
        rear = (rear + 1) % SIZE;
        items[rear] = element;
        printf("\n Inserted -> %d", element);
        }
}
```

IMPLEMENTING CIRCULAR QUEUE-DELETION

```
int deQueue()
{
        int element;
        if (isEmpty())
        {
                printf("\n Queue is empty !! \n");
                return (-1);
        }
        else
        {
                element = items[front];
                if (front == rear) { front = -1; rear = -1; }
                else { front = (front + 1) % SIZE; }
                printf("\n Deleted element -> %d \n", element);
                return (element);
        }
}
```

IMPLEMENTING CIRCULAR QUEUE-DISPLAY

```
void display()
{
        int i;
        if (isEmpty())
                printf(" \n Empty Queue\n");
        else {
                printf("\n Front -> %d ", front);
                printf("\n Items -> ");
                for (i = front; i != rear; i = (i + 1) % SIZE)
                        { printf("%d ", items[i]); }
                printf("%d ", items[i]);
                printf("\n Rear -> %d \n", rear);
         }
}
```

QUEUE IMPLEMENTATION USING LINKED LIST

```
struct node {
    int data;
    struct node * next;
};

struct node * front = NULL;
struct node * rear = NULL;
```

QUEUE IMPLEMENTATION USING LINKED LIST INSERTION

```
void enqueue(int value) {
    struct node * ptr;
    ptr = (struct node * ) malloc(sizeof(struct node));
    ptr -> data = value;
    ptr -> next = NULL;
    if ((front == NULL) && (rear == NULL)) {
        front = rear = ptr;
    } else {
        rear -> next = ptr;
        rear = ptr;
    }
    printf("Node is Inserted\n\n");
}
```

QUEUE IMPLEMENTATION USING LINKED LIST DELETION

```
int dequeue() {
    if (front == NULL) {
        printf("\nEmpty Queue\n");
        return -1;
    } else {
        struct node * temp = front;
        int temp_data = front -> data;
        front = front -> next;
        free(temp);
        return temp_data;
    }
}
```

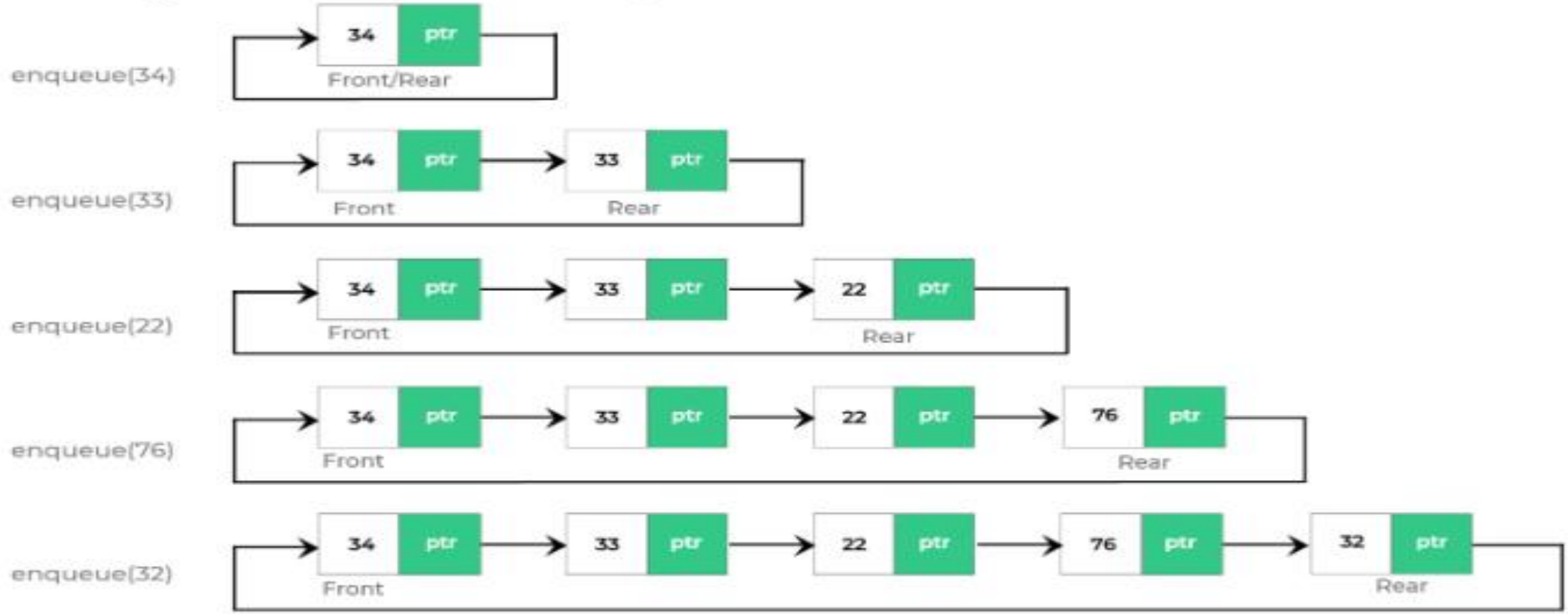
QUEUE IMPLEMENTATION USING LINKED LIST

DISPLAY

```
void display() {
    struct node * temp;
    if ((front == NULL) && (rear == NULL)) {
        printf("\nQueue is Empty\n");
    } else {
        printf("The queue is \n");
        temp = front;
        while (temp) {
            printf("%d--->", temp -> data);
            temp = temp -> next;
        }
        printf("NULL\n\n");
    }
}
```

CIRCULAR QUEUE USING LINKED LIST

Adding the elements into Queue



Removing the elements from Queue