Circular Queue (LAB-7)

```
#include <stdio.h>
#define size 5
int queue[size];
int front = -1;
int rear = -1;

void enqueue (int a)
{
   if ((front == rear + 1) || (front == 0 && rear == size - 1))
   {
      printf ("Queue overflow \n");
      return;
   }
```

```
else
{
    if (front == -1)
        front = 0;
    rear = (rear + 1) % size;
    queue[rear] = a;
}
}
void dequeue()
{
  if (front == -1)
  { printf("Queue Empty \n");
  }
  else
  {
    int a = queue[front];
    if (front == rear)
    {
      front = -1;
      rear = -1;
    }
    else
    {
      front = (front+1) % size;
    }
    printf("Deleted element = %d \n", a);
    return (a);
  }
}
void display()
{
    if (front == -1)
    {
```

```
        printf("Queue Empty\n");
        return;
    }
    else
    {
        int i;
        printf("\n Front = %d", front);
        printf("\n Items = ");
        for(i=front ; i!=rear ; i=(i+1)%size)
        {
            printf("%d", queue[i]);
        }
        printf("%d", queue[i]);
        printf("\n Rear = %d", rear);
    }
}
void main()
{
    int choice;
    int a;
    while(1)
    {
        printf("\n 1. Insert \n 2. Delete \n 3. Display \n Choice: ");
        scanf("%d", &choice);
        switch(choice)
        {
            case 1: printf("Enter an element: ");
                    scanf("%d", &a);
                    enqueue(a);
                    display();
                    break;
            case 2: dequeue();
                    display();
                    break;
            case 3: display();
                    break;
        }
    }
}
```

Output:

1. Insert
2. Delete
3. Display
Choice : 1
Enter an element : 2

Front = 0
Items = 2
Rear = 0

1. Insert
2. Delete
3. Display
Choice : 1
Enter an element : 3

Front = 0
Items = 23
Rear = 1

1. Insert
2. Delete
3. Display
Choice : 1
Enter an element : 4

Front = 0
Items = 234
Rear = 2

1. Insert
2. Delete
3. Display
Choice : 2
Deleted element = 2

Front = 1
Items = 34
Rear = 2

LAB-4 (Insertion of element at every pos in a Linked List)

```
#include <stdio.h>
#include <stdlib.h>

struct Node
{
  int data;
  struct Node * next;
};

Struct Node *createNode (int value)
{
  struct Node *newNode = (struct Node*) malloc (sizeof (struct Node));
  if (newNode == NULL)
  {
    printf ("Memory allocation failed \n");
    exit (1);
  }
  newNode → data = value;
  newNode → next = NULL;
  return newNode;
};

void displayList (struct Node *head)
{
  if (head == NULL)
  {
    printf ("List is empty \n");
    return;
  }
  struct Node * temp = head;
  printf ("Linked list : \n");
  while (temp != NULL)
  {
    printf ("%d → ", temp → data);
    temp = temp → next;
  }
```

```
        printf("NULL\n");
    }
struct Node *insertatbeginning(struct Node *head, int value)
{
    struct Node *newNode = createNode(value);
    newNode->next = head;
    return newNode;
};
void insertatend(struct Node *head, int value)
{
    struct Node *newNode = createNode(value);
    struct Node *temp = head;
    while (temp->next != NULL)
    {
        temp = temp->next;
    }
    temp->next = newNode;
}
void insertatposition(struct Node *head, int position, int value
{
    struct Node *newNode = createNode(value);
    struct Node *temp = head;
    int count = 1;
    while (temp != NULL && count < position - 1)
    {
        temp = temp->next;
        count++;
    }
    if (temp == NULL)
    {
        printf("Position out of range\n");
        free(newNode);
    return;
```

```
    newNode->next = temp->next;
    temp->next = newNode;
}

int main()
{
    struct Node *head = NULL;

    head = createNode(1);
    head->next = createNode(2);
    head->next->next = createNode(3);

    printf("Initial:\n");
    displayList(head);

    head = insertAtBeginning(head, 0);
    printf("After insertion at the beginning:\n");
    displayList(head);

    insertAtPosition(head, 3, 10);
    printf("After insertion at position 3:\n");
    displayList(head);

    ~~insertAtPosition(2)~~
    insertAtEnd(head, 20);
    printf("After insertion at the end:\n");
    displayList(head);

    return 0;
}
```

Output:

Initial:

Linked List:

1 → 2 → 3 → NULL

After insertion at the beginning:

Linked List:

0 → 1 → 2 → 3 → NULL

After insertion at position 3:

Linked list:

0 → 1 → 10 → 2 → 3 → NULL

After insertion at the end:

Linked list:

0 → 1 → 10 → 2 → 3 → 20 → NULL.

Executed
11/1/24