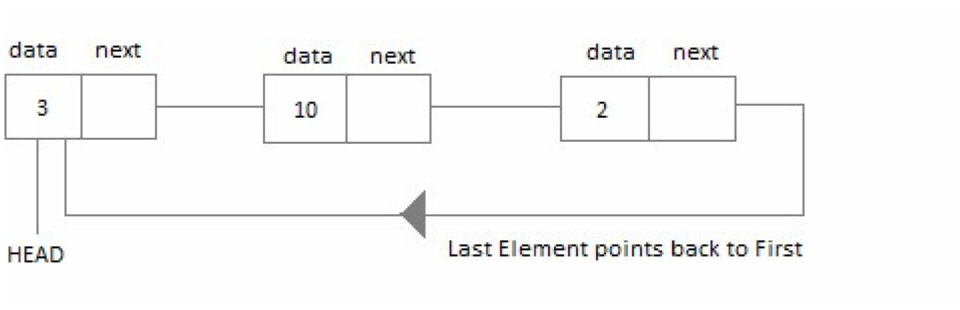
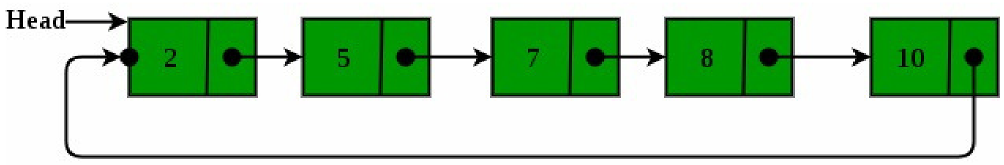
EXPERIMENT NO. 7
IMPLEMENTATION OF CIRCULAR LINKED LIST

Aim: Write A Program to Implement Circular Linked List.

Theory: Circular Linked List is little more complicated linked data structure. In the circular linked list we can insert elements anywhere in the list whereas in the array we cannot insert element anywhere in the list because it is in the contiguous memory. In the circular linked list the previous element stores the address of the next element and the last element stores the address of the starting element. The elements points to each other in a circular way which forms a circular chain. The circular linked list has a dynamic size which means the memory can be allocated when it is required.





Advantages of Circular Linked Lists: 1) Any node can be a starting point. We can traverse the whole list by starting from any point. We just need to stop when the first visited node is visited again. 2) Useful for implementation of queue. Unlike this implementation, we don't need to maintain two pointers for front and rear if we use circular linked list. We can maintain a pointer to the last inserted node and front can always be obtained as next of last. 3) Circular lists are useful in applications to repeatedly go around the list. For example, when multiple applications are running on a PC, it is common for the operating system to put the running applications on a list and then to cycle through them, giving each of them a slice of time to execute, and then making them wait while the CPU is given to another application. It is convenient for the operating system to use a circular list so that when it reaches the end of the list it can cycle around to the front of the list.

Application of Circular Linked List

- The real life application where the circular linked list is used is our Personal Computers, where multiple applications are running. All the running applications are kept in a circular linked list and the OS gives a fixed time slot to all for running. The Operating System keeps on iterating over the linked list until all the applications are completed. Another example can be Multiplayer games. All the

- Players are kept
in a Circular Linked List and the pointer keeps on moving forward as a player's chance ends.
- Circular Linked List can also be used to create Circular Queue. In a Queue we have to keep two pointers, FRONT and REAR in memory all the time, where as in Circular Linked List, only one pointer is required.

FOR APPEND OPERATION:

1. If the Linked List is empty then we simply, add the new Node as the Head of the Linked List.
2. If the Linked List is not empty then we find the last node, and make it' next to the new Node, and make the next of the Newly added Node point to the Head of the List.

FOR INBEGIN OPERATION:

1. The first Node is the Head for any Linked List.
2. When a new Linked List is instantiated, it just has the Head, which is Null.
3. Else, the Head holds the pointer to the fisrt Node of the List.
4. When we want to add any Node at the front, we must make the head point to it.
5. And the Next pointer of the newly added Node, must point to the previous Head, whether it be NULL(in case of new List) or the pointer to the first Node of the List.
6. The previous Head Node is now the second Node of Linked List, because the new Node is added at the front.

FOR DELETE OPERATION:

- If the Node to be deleted is the first node, then simply set the Next pointer of the Head to point to the next element from the Node to be deleted. And update the next pointer of the Last Node as well.
- If the Node is in the middle somewhere, then find the Node before it, and make the Node before it point to the Node next to it.
- If the Node is at the end, then remove it and make the new last node point to the head.

FOR COUNT OPERATION:

```
1. Start
2. Declare C =0
3. while (Q= =NULL)
4. Display “List Does Not Exists”
5. Else (Step 5 To Step 10)
6. Repeat till Step 8 while (Q! =rear)
7. Increment C by 1
8. Q=Q’s Link Section
9. End while
10.Increment c by 1
11.Display Value of C
12..Stop
```

FOR DISPLAY OPERATION:

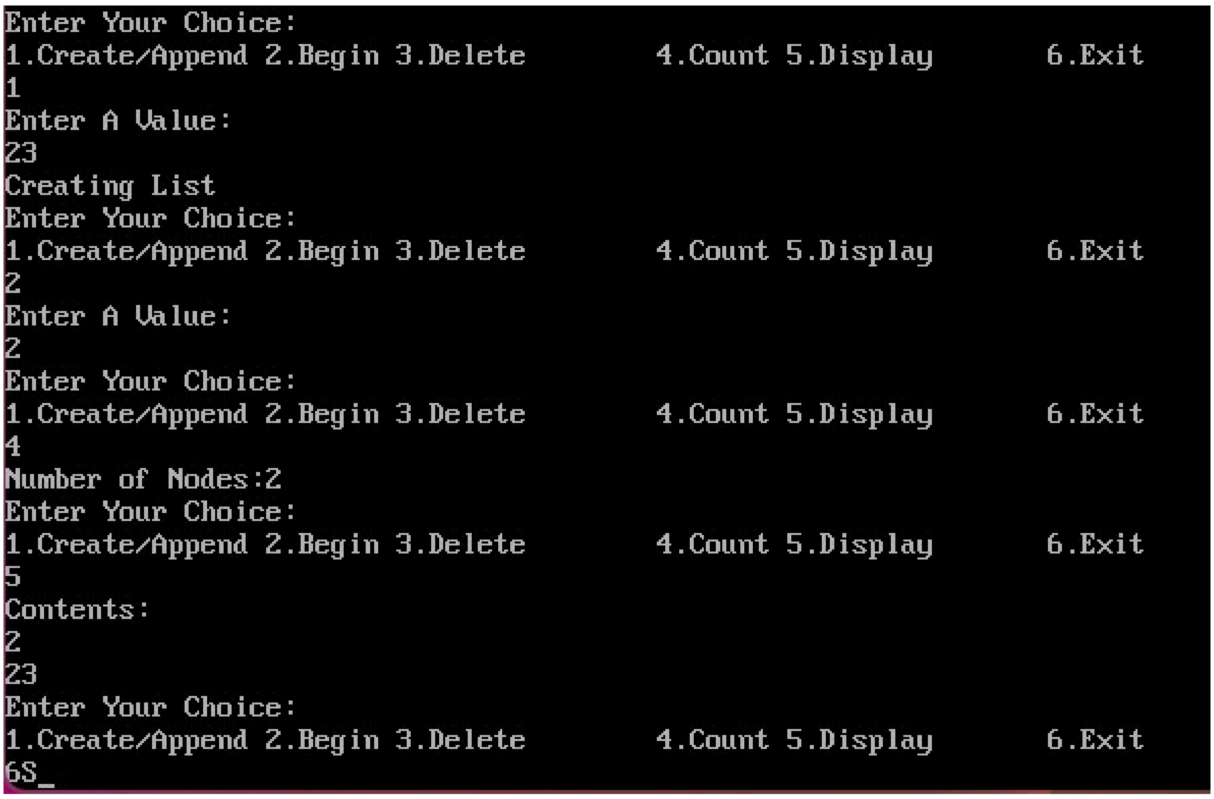
```
1.Start
2. if (Q= =NULL)
3.Display “List Does Not Exists”
4.Else (Step 4 To Step 8)
5.Display “Contents”
6.Move to A New Line
7.Repeat till Step 8 while (Q! =rear)
8.Display Value of (Q’s Data Section)
9.Move to A New Line
10.Q=Q’s Link Section
11.Display Value of(Q’s Data section)
12.Stop
```

Program:

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

struct node
{
   int data;
    struct node *link;
};
struct node *p;

void append(struct node *q, int num)
{
  struct node *temp, *r;
  if (q= NULL)
   {
     printf("Creating List\n");
     temp = (struct node *)malloc(sizeof(struct node));
     temp->data = num;
     temp->link = NULL;
     p = temp;
     q = temp;
   }
   else
   {
     r=q;
     while (r->link != NULL) r = r->link;
     temp = (struct node *)malloc(sizeof(struct node));
     temp->data = num;
     temp->link = NULL;
    }
  }

void inbegin(struct node *q, int num)
{
   struct node *temp;
    if (q = NULL)
    printf("Link List Does Not Exist\n");
```

```
     else
     {
        temp = (struct node *)malloc(sizeof(struct node));
        temp->data = num;
        temp->link = p;
        p = temp;
     }
   }

void delet(struct node *q, int num)
{
  struct node *prev, *curr;
  int found = 0;
  prev = NULL;
  if (q == NULL)
  printf("List does not exist\n");
  else
  {
    for (curr = q; curr != NULL;
    prev = curr, curr = curr->link)
    {
      if (curr->data = num)
      {
        if (prev == NULL)
        {
           p = curr->link;
           found = 1;
        }
        else
        {
           prev->link = curr->link;
           found = 1;
         }
      }
  }
  if (found = 1)
  printf("Node deleted\n");
  else
  printf("Not found\n");
}

void count(struct node *q)
{
  int c;
  c = 0;
  if (q = NULL)
  printf("Link List Does Not Exist\n");
  else
  {
    while (q != NULL)
    {
      c++;
      q = q->link;
    }
    printf("Number Of Nodes: %d\n", c);
   }
}

void display(struct node *q)
{
  if (q = NULL)
  printf("Link List Does Not Exist\n");
  else
  {
     printf("Contents:\n");
     while (q != NULL)
      {
         printf("%d\n", q->data);
         q = q->link;
       }
   }
}

void main()
{
   int n, c;
   P = NULL;
   do
   {
      printf("Enter Your Choice:\n");
      printf("1. Create/Append\t");
      printf("2. Begin\t");
      printf("3. Delete\t");
      printf("4. Count\t");
      printf("5. Display\t");
      printf("6. Exit\n");
      scanf("%d", &c);
    switch (c)
    {
    case 1:
    printf("Enter A Value:\n");
    scanf("%d", &n);
    append(p, n);
    break;
    case 2:
    printf("Enter A Value:\n");
    scanf("%d", &n);
    inbegin(p, n);
    break;
    case 3:
    printf("Enter A Value:\n");
    scanf("%d", &n);
    delet(p, n);
    break;
    case 4:
    count(p);
    break;
    case 5:
    display(p);
     break;
    }
  }while(c!=6);
  getch();
 }
```

Output:



```
Enter Your Choice:
1.Create/Append 2.Begin 3.Delete        4.Count 5.Display       6.Exit
1
Enter A Value:
23
Creating List
Enter Your Choice:
1.Create/Append 2.Begin 3.Delete        4.Count 5.Display       6.Exit
2
Enter A Value:
2
Enter Your Choice:
1.Create/Append 2.Begin 3.Delete        4.Count 5.Display       6.Exit
4
Number of Nodes:2
Enter Your Choice:
1.Create/Append 2.Begin 3.Delete        4.Count 5.Display       6.Exit
5
Contents:
2
23
Enter Your Choice:
1.Create/Append 2.Begin 3.Delete        4.Count 5.Display       6.Exit
6S_
```

Conclusion:

The experiment successfully implements a circular linked list, demonstrating its structure where the last node points to the first, forming a loop. It highlights dynamic memory allocation, allowing easy insertion and deletion at any point. The experiment shows its advantage in queue implementation and traversal starting from any node. This understanding is crucial for applications like round-robin scheduling and resource management.