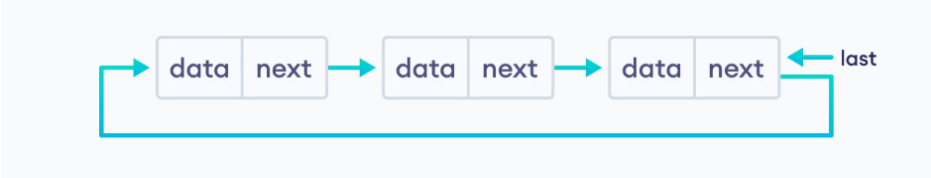
Practical – 7
Implementation of Circular and Doubly Linked List

Circular Linked list

In a circular Singly linked list, the last node of the list contains a pointer to the first node of the list. We can have circular singly linked list as well as circular doubly linked list.

We traverse a circular singly linked list until we reach the same node where we started. The circular singly liked list has no beginning and no ending. There is no null value present in the next part of any of the nodes.

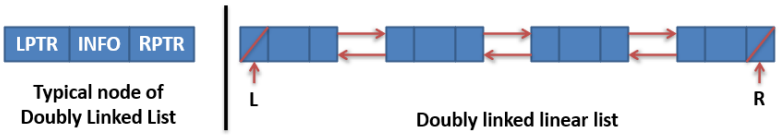



```
typedef struct Node
{
	int info;
	struct Node *next;
}node;

node *first=NULL,*last=NULL,*temp;
void create()
{
	node *newnode;
	newnode=(node*)malloc(sizeof(node));
	printf("\nEnter the node value : ");
	scanf("%d",&newnode->info);
	newnode->next=NULL;
	if(last==NULL)
		first=last=newnode;
	else
	{
		last->next=newnode;
		last=newnode;
	}
	last->next=first;
}

void del()
{
	temp=first;
	if(first==NULL)
	printf("\nUnderflow :");
	else
	{
	if(first==last)
	{	printf("\n%d",first->info);
		first=last=NULL;
	}
	else
	{		printf("\n%d",first->info);
		first=first->next;
```

```
            last->next=first;
      }
       temp->next=NULL;
      free(temp);
      }
}
```

Doubly linked list

Doubly linked list is a complex type of linked list in which a node contains a pointer to the previous as well as the next node in the sequence. Therefore, in a doubly linked list, a node consists of three parts: node data, pointer to the next node in sequence (next pointer) , pointer to the previous node (previous pointer).




```
struct                                                          node
{  int data;
    struct node *prev, *next;
};
struct node* head = NULL;
struct node* tail = NULL;
void insertFront()
{
      int val;
      struct node* temp;
      temp=(struct node*)malloc(sizeof(struct node));
      printf(" Enter value= ");
      scanf("%d",&val);
      temp->data=val;
      temp->prev=NULL;
      temp->next=head;
      if(head!=NULL)
            head->prev=temp;
      head=temp;

}
void insertPosition()
{
      int val,pos,i=1;
      struct node *temp, *newnode;
      newnode=malloc(sizeof(struct node));
      newnode->next=NULL;
      newnode->prev=NULL;
      printf("Enter position= ");
      scanf("%d",&pos);
      printf("Enter value= ");
      scanf("%d",&val);
      newnode->data=val;
      temp=head;
      if(head==NULL)
      {
            head=newnode;
            newnode->prev=NULL;
            newnode->next=NULL;
      }
```

```
	else if(pos==1)
	{
		newnode->next=head;
		newnode->next->prev=newnode;
		newnode->prev=NULL;
		head=newnode;
	}
	else
	{
		while(i<pos-1)
		{
		    temp = temp->next;
		    i++;
		}
		newnode->next=temp->next;
		newnode->prev=temp;
		temp->next=newnode;
		temp->next->prev=newnode;
	}
}
void deleteFirst()
{	struct node* temp;
	if(head==NULL)
	{	printf("List is empty\n");
	}
	else
	{	temp=head;
		head=head->next;
		if(head!=NULL)
		{
			head->prev=NULL;
		}
		free(temp);
	}
}
void deleteEnd()
{	struct node* temp;
	if(head==NULL)
	{	printf("List is empty\n");
	}
	temp=head;
	while(temp->next!=NULL)
	{	temp=temp->next;
	}
	if(head->next==NULL)
	{	head=NULL;
	}
	else
	{	temp->prev->next=NULL;
		free(temp);
	}
}
void deletePosition()
{	int pos,i=1;
	struct node *temp, *position;
	temp=head;
	if(head==NULL)
	{
		printf("List is empty\n");
	}
	else
```

```
        {       printf("Enter position= ");
                scanf("%d",&pos);
                if(pos==1)
                {     position=head;
                    head=head->next;
                    if(head!=NULL)
                    {
                      head->prev=NULL;
                    }
                    free(position);
                    return;
                 }
                 while(i<pos-1)
                 {
                       temp=temp->next;
                       i++;
                 }
                 position=temp->next;
                 if(position->next!=NULL)
                 {
                       position->next->prev=temp;
                 }
                 temp->next=position->next;
                 free(position);
        }
}
```

Exercise

1. Write a program to implement Enqueue and Dequeue operations of circular queue using circular link list.
2. Write program for all operations of doubly linked list.
   a. Inserting Node – as First Node, at specific location, as Last Node
   b. Deleting Node – at First, at Last, specific node
   c. Display List