3) WAP Implement
doubly link list with primitive operations

a) Create a doubly linked list.
b) Insert a new node to the left of the node.
c) Delete the node based on a specific value
d) Display the contents of the list

```
#include <stdio.h>

#include <malloc.h>



struct NODE

{

   struct NODE *Llink;

   int data;

   struct NODE *Rlink;

};

typedef struct NODE node;

node *start = NULL;

void main()

{

   int ch;

   while (1)

   {

      printf("\n1.Create LL\t2.Insert\t3.Delete\t4.Display\t5.Exit\n");

      printf("Enter your choice:\n");

      scanf("%d", &ch);

      switch (ch)

      {
      case 1:
          create_DLL();
          break;

      case 2:
          Insert_pos();
          break;
      case 3:
          Delete_pos();
          break;
      case 4:
          display();
          break;
      case 5:
          exit(0);
          break;
      default:
          printf("Invalid choice\n");
      }
   }
}
void create_DLL()
{
   int c;
   node *new, *curr;
    start = (node *)malloc(sizeof(node));
    printf("Enter element\n");
    scanf("%d", &start->data);
    start->Llink = NULL;
    curr = start;
    while (1)
    {
        printf("Do you want to add another element(Y/N)\n");
        scanf("%d", &c);
        if (c == 1)
        {
            new = (node *)malloc(sizeof(node));
            printf("Enter element\n");
            scanf("%d", &new->data);
            curr->Rlink = new;
            new->Llink = curr;
            curr = new;
        }
        else
        {
            curr->Rlink = NULL;
            break;
        }
    }
}
```

```
void Insert_pos()

{

   node *new, *temp;

   int pos;

   new = (node *)malloc(sizeof(node));

   printf("Enter Element\n");

   scanf("%d", &new->data);

   printf("Enter Position\n");

   scanf("%d", &pos);

   if (pos == 1)

   {

      new->Rlink = start;

      start->Llink = new;

      new->Llink = NULL;

      start=new;

      return;

   }

   temp = start;

   int i = 1;

   while (i < (pos - 1) && temp != NULL)

   {

      temp = temp->Rlink;

      i++;

   }

   if (i == (pos - 1))
```

```
    {
        new->Llink = temp;
        temp->Rlink->Llink = new;
        new->Rlink = temp->Rlink;
        temp->Rlink = new;
        return;
    }
    else if (temp == NULL)
    {
        printf("Invalid position");
    }
}
void Delete_pos()
{
    node *temp, *curr, *next;
    int el;
    if (start == NULL)
    {
        printf("Linked List is empty\n");
        return;
    }
    printf("Enter element to be deleted: ");
    scanf("%d", &el);

    if (start->data == el)
```

```
{
    temp = start;
    if (start->Rlink == NULL)
    {
        start = NULL;
    }
    else
    {
        start = start->Rlink;
    }
    printf("Deleted Element is %d", temp->data);
}
curr = start;
next = start->Rlink;

while (curr->data != el && next->Rlink != NULL)
{
    curr = next;
    next = next->Rlink;
}
if (curr->data == el)
{
    curr->Llink->Rlink = next;
    next->Llink = curr->Llink;
    printf("Deleted element is :%d\n", curr->data);
```

```
        free(curr);
    }
    else if (next->data == el)
    {
        curr->Rlink = NULL;
        printf("\nDeleted element is :%d\n ", next->data);
        free(next);
    }
    else
    {
        printf("\nElement Not Found\n");
    }
}
void display()
{
    node *temp;
    if (start == NULL)
    {
        printf("Linked List is empty\n");
        return;
    }
    printf("Elements are:\n");
    temp = start;
    while (temp != NULL)
    {
        printf("%d\t", temp->data);

        temp = temp->Rlink;

    }

}
```