LINKED LIST

INTRODUCTION

So far, data structures such as stacks, queues have been understood by representing and implementing them as arrays.

Sequential allocation such as arrays are easy to understand, use them and access elements using the index.

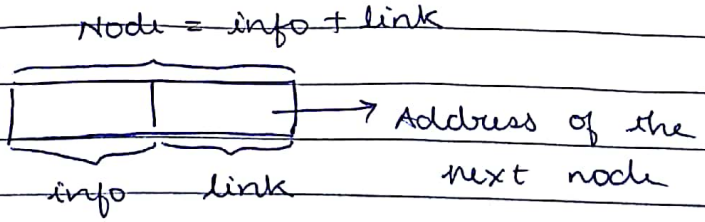
However, there are some disadvantages such as the fixed size available; insufficient continuous memory available and insertion and deletion of elements from any part of the array becomes tedious.

The above said disadvantages can be overcome by using LINKED LIST.

DEFINITION

A LINKED LIST is a data structure which is a collection of zero or more nodes where each node has some information.

The pictorial representation of the node is as follows.

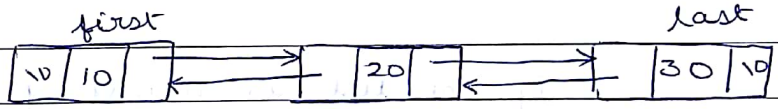


NOTE: Given address of a node, we can obtain address of subsequent nodes using link fields. Thus adjacency between the nodes is maintained by means of links.

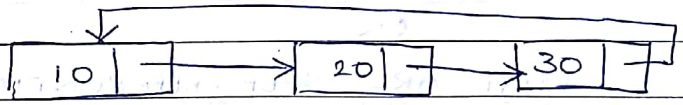
TYPES OF LINKED LISTS
1. Singly linked list
2. Doubly linked list
3. Circular singly LL
4. Circular doubly LL

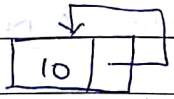
SINGLY LINKED LIST :-

DOUBLY LINKED LIST

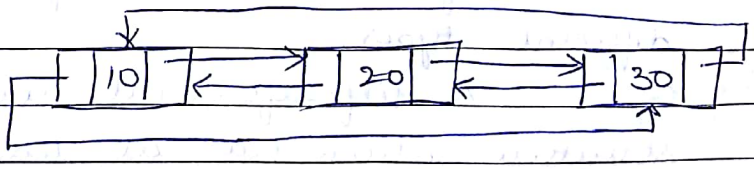


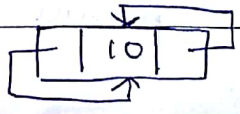
CIRCULAR SLL





CIRCULAR DLL



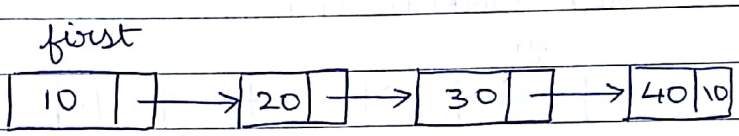


SINGLY LINKED LISTS AND CHAINS

A singly linked list is a collection of zero or more nodes where each node has two or more fields and only one link field which contains the address of the next node.

A chain is a singly linked list with zero or more nodes.

PICTORIAL REPRESENTATION OF SLL



'first' is a pointer pointing to first node in the list.

If the list is empty first == NULL.

REPRESENTING CHAINS / LINKS IN C

OR

WHAT ARE SELF-REFERENTIAL STRUCTURES ?

OR

HOW TO DEFINE A NODE IN C.

A structure is a collection of one or more fields which are of same or different types.

A self-referential structure is a structure which has at least one field which is a pointer to same structure.

For example, consider the following structure definition

```
struct node
{
    int info;
    struct node * link;
};
```

DECLARING A VARIABLE

```
typedef struct node * NODE;
```

i.e. struct node * = NODE

```
NODE first;
```

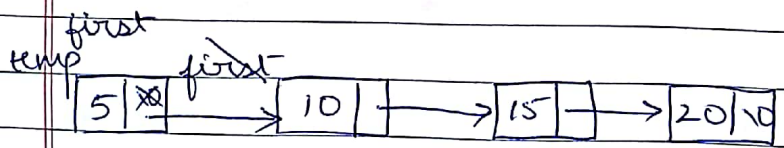
when list is empty first = NULL;

// FUCTION TO INSERT A NODE AT THE BEGINI
// OF THE LIST [SLL]

```
void insertfront()
{
    NODE temp;
    temp = (NODE) malloc (sizeof (struct node));
    printf("Enter the data\n");
    scanf("%d", &temp->info);
    temp->link = NULL;

    if(first == NULL)        //List empty.
        first = temp;

    else
    {
        temp->link = first;
        first = temp;
    }
}
```

first = NULL

temp

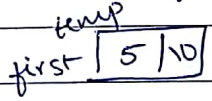


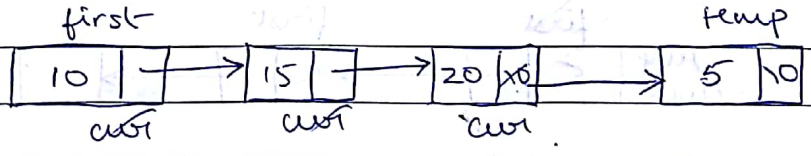
classmate
Date
Page

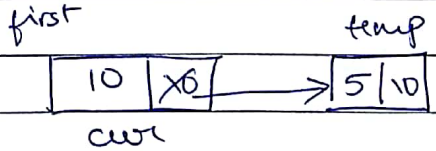
FUNCTION TO INSERT A NODE AT THE END OF SLL

```
void insertend()
{
    NODE temp, cur;
    temp = (NODE) malloc (sizeof (struct node));
    printf ("Enter the data \n");
    scanf ("%d", &temp->info);
    temp->link = NULL;

    if (first == NULL)
        first = temp;

    else
    {
        cur = first;
        while (cur->link != NULL)
            cur = cur->link;
        cur->link = temp;
    }
}
```

first=NULL



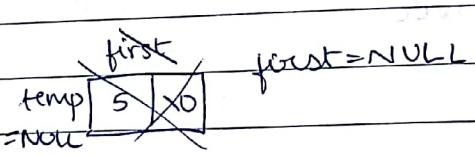


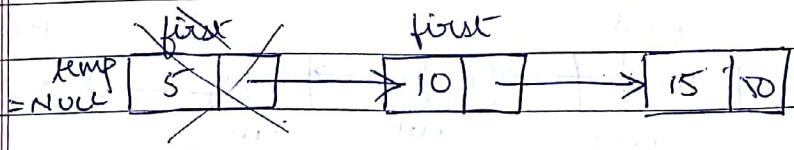


// FUNCTION TO DELETE A NODE AT THE
// BEGINING OF THE LIST [SLL]

```
void deletefront()
{
    NODE temp;
    if(first == NULL)
        printf("List is empty\n");

    else
    {
        temp = first;
        first = first -> link;
        printf("%d is deleted", temp->info);
        free(temp);
        temp = NULL;
    }
}
```

first = NULL

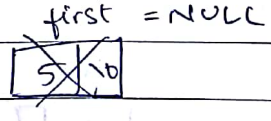


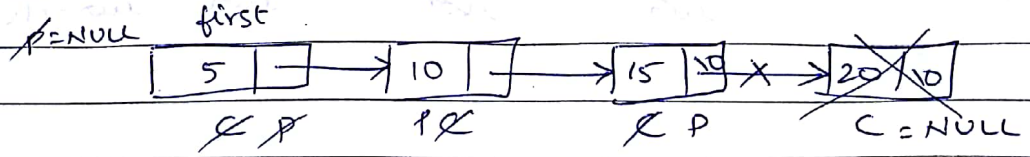


// FUNCTION TO DELETE A NODE AT THE END OF SLL

```
void delatend()
{
    NODE prev, cur;
    if (first == NULL)
        printf ("List is empty\n");

    else if (first->link == NULL)
    {
        printf ("%d is deleted", first->info);
        free (first);
        first = NULL;
    }

    else
    {
        prev = NULL;
        cur = first;
        while (cur->link != NULL)
        {       prev = cur;
                cur = cur->link;
        }
        prev->link = NULL;
        printf ("%d is deleted\n", cur->info);
        free (cur);
        cur = NULL;
    }
}
```

first = NULL

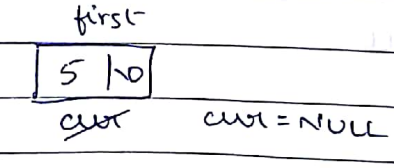


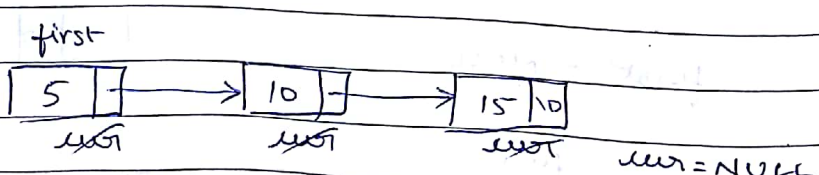


//FUNCTION TO DISPLAY SLL

```
void display()
{
   NODE cur;
   if(first==NULL)
       printf("List is empty\n");

   else
   {
       printf("\n List elements are \n");
       cur= first;
       while (cur!=NULL);
       {
           printf("%d\t", cur->info);
           cur = cur->link;
       }
   }
}
```

first=NULL





// FUNCTION FOR LINEAR SEARCH IN SLL

```
void linearsearch()
{
    NODE cur;
    int key, loc;
    printf("Enter the key to be searched\n");
    scanf("%d", &key);

    if(first == NULL)
      printf("List is empty\n");

    else
    {
      cur = first;
      loc = 1;
      while(cur != NULL)
      {
          if(cur->info == key)
          {
            printf("%d is found at %d loc",
                    cur->info, loc);
            break;
          }
          else
          {
            cur = cur->link;
            loc++;
          }
      }
    }
}
```

/* WRITE A PROGRAM IN C TO ILLUSTRATE THE OPERATIONS OF STACK USING SLL */

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

struct node
{
   int info;
   struct node *link;
};
typedef struct node * NODE;

NODE top = NULL;
```

/* Write the function to insert a node at the bigining of the SLL, refer Pg 6 */

/* For pop function, write the function to delete a node at the bigining of a SLL, refer Pg 8 */

/* To display the stack, write the function to ~~dilete~~ display a SLL, refer Pg 10 */

```
void main()
{
    int ch;

    for(;;)
    {
        printf("\n1.Push\n2.Pop\n3.Display\n4.Exit\n");
        printf("Enter your choice\n");
        scanf("%d", &ch);

        switch(ch)
        {
            case 1: push();
                    break;

            case 2: pop();
                    break;

            case 3: display();
                    break;

            case 4: exit(0);

            default: printf("Invalid choice\n");
        }
    }
}
```

NOTE: Use appropriate terms for the functions like push, pop and to refer to the list we can use top as the variable.

/* PROGRAM TO ILLUSTRATE OPERATIONS OF QUEUE USING SINGLY LINKED LIST */

// Include header files

// Add the global declaration of structure, // typedef statment and initialization of fir

/* For the insert function, write function to add a node at the end of the SLL, refer Pg 7 */

// * For delete function, write ~~first~~ function to delete a node at the begining of the list, refer Pg 8 */

/* For display function refer Pg 10 */

/* The main fuction should have the menu*

/* FUCTION TO COUNT THE NUMBER OF NODES IN A SLL */

```
void countnodes()
{
    NODE cur;
    int count;
    if(first == NULL)
        printf("List is empty\n");

    else
    {
        cur = first;
        count = 0;
        while(cur != NULL)
        {
            cur = cur -> link;
            count++;
        }
        printf("The list has %d nodes", count);
    }
}
```

```
//FUNCTION TO INSERT A NEW NODE AT A
//SPECIFIED LOCATION IN THE LIST [SLL]

void insertloc()
{
    int loc, cnt;
    NODE temp, cur;
    printf("Enter the location\n");
    scanf("%d", &loc);

    temp = (NODE)malloc(sizeof(struct node));
    printf("Enter the data\n");
    scanf("%d", &temp->info);
    temp->link = NULL;

    if(first == NULL)
    {
        if(loc == 1)
        {       first = temp;
            temp
        }

        else
            printf("Invalid location\n");

    }

    else if(loc == 1)
    {
        temp -> link = first;
        first = temp;
    }
```

```
else
{
    cur = first;
    cntr = 1;

    while (cur != NULL)
    {
        if (cntr == loc - 1)
        {
            temp -> link = cur -> link;
            cur -> link = temp;
            break;
        }
        cur = cur -> link;
        cntr++;
    }
    if (cur == NULL)
        printf ("Invalid location\n");
}
}
```

classmate
Date
Page

// FUNCTION TO DELETE A NODE WHOSE INFO/ // VALUE IS SPECIFIED IN A SLL

```
void delutiKey()
{
    int key, flag = 0;
    NODE temp, prev, cur;
    printf("Enter the key\n");
    scanf("%d", &key);

    if(first == NULL)
        printf("List is empty\n");

    else if (first -> info == key)
    {
        temp = first;
        first = first -> link;
        printf("%d is deletd", temp -> info);
        free(temp);
        temp = NULL;
    }

    else
    {
        cur = first -> link;
        prev = first;
        while(cur != NULL)
        {
            if(cur -> info == key)
            {
                printf("%d is deletd", cur -> info)
                prev -> link = cur -> link;
                free(cur);
                cur = NULL; flag = 1;
            }
```

```
        prev = cur;
        cur = cur->link;
    }
    if (flag == 0)
        printf("Key Not found\n");
}
}
```

CIRCULAR SINGLY LINKED LIST

Circular singly linked list is a data structure with zero or more nodes. Each node has at least two fields, one to save the information and another, a link to point to the next node.

The last node in the circular singly linked list would point to the first node in the list.

Representing each node :-

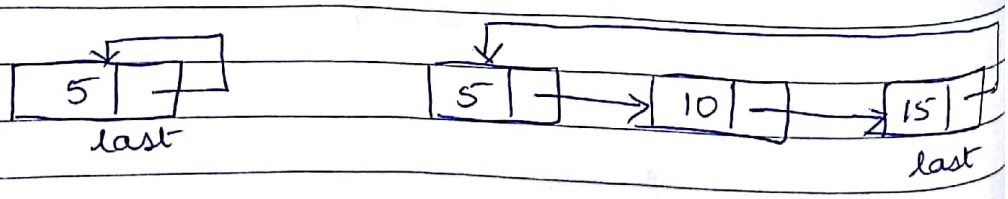
```
struct node
{
    int info;
    struct node * link;
};
```

A pointer last would be used to refer to the list. The pointer last points to the last node in a CSLL.

When list is empty last = NULL

declaring variable :-

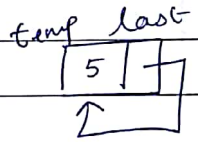
```
typedef struct node * NODE;
NODE last = NULL;
```

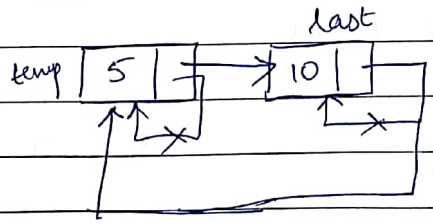


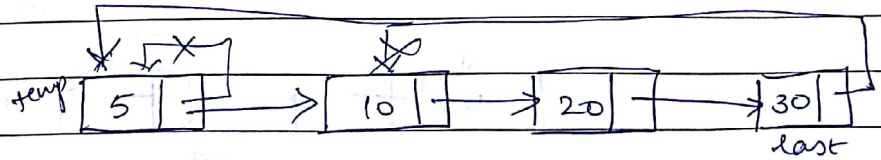
// FUNCTION TO INSERT A NODE TO A CIRCULAR SLL

```
void insertnode()
{
    NODE temp;
    temp = (NODE) malloc(sizeof(struct node));
    printf("\nEnter the data\n");
    scanf("%d", &temp->info);
    temp->link = temp;

    if(last == NULL)  //List empty
        last = temp;

    else
    {
        temp->link = last->link;
        last->link = temp;
    }
}
```

last = NULL



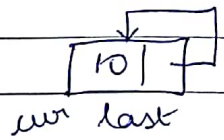


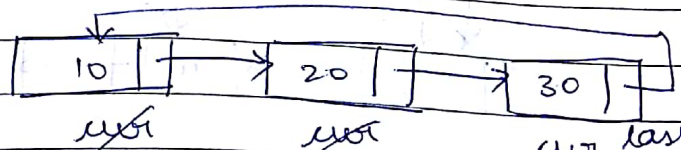


// FUNCTION TO DISPLAY CSLL

```
void display()
{

    if(last == NULL)
    {   printf("List is empty\n");
        return;
    }

    cur = last -> link;
    printf("List elements are\n");

    while(cur != last)
    {
        printf("%d\t", cur -> info);
        cur = cur -> link;
    }
    printf("%d\n", cur -> info); // print last node

}
```

last = NULL





DOUBLY LINKED LIST

Doubly linked list is a data structure with zero or more nodes. Each node has at least three fields, one to save the information and two other to point to the previous node and the next node.

NOTE 1: Two pointers will be maintained for DLL to refer to the first node and last node in the list.

NOTE 2: In a DLL traversal can be done in two directions.

Representing each node :-

```
struct node
{
    int info;
    struct node *llink;
    struct node *rlink;
};
```

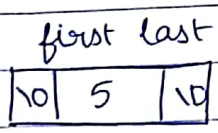
declaring a variable :-

```
typedef struct node * NODE;
NODE first = NULL;
NODE last = NULL;
```

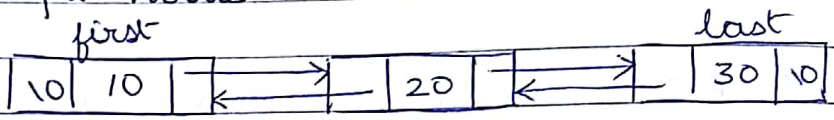
If list is empty :-

first = last = NULL;

List with one node :-



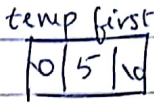
Multiple nodes

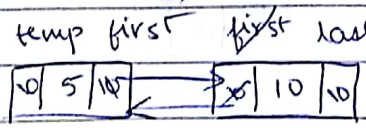


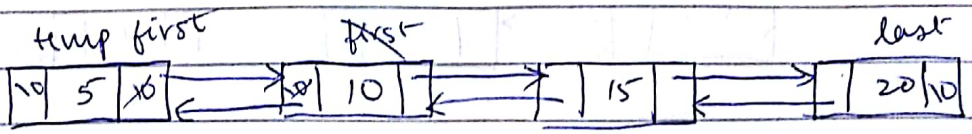
// FUNCTION TO INSERT A NODE AT THE
// BEGINING OF DLL

```
void insertfront()
{
   NODE temp;
   temp = (NODE) malloc (sizeof(struct node));
   printf("Enter the data\n");
    scanf("%d", &temp->info);
   temp->llink = NULL;
   temp->rlink = NULL;
    if (first == NULL)
    {     first = last = temp;
          return;
     }

   temp->rlink = first;
   first->llink = temp;
    first = temp;

}
```

first = last = NULL



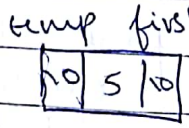


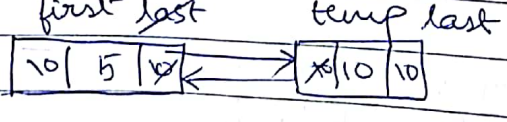


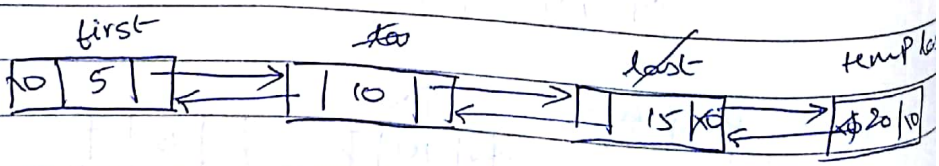
// FUNCTION TO INSERT A NODE AT THE
// END OF THE LIST [DLL]

```
void insertend()
{

    NODE temp;
    temp = (NODE) malloc(sizeof(struct node));
    printf("Enter the data\n");
    scanf("%d", &temp->info);
    temp->rlink = temp->llink = NULL;

    if (first == NULL)
    {
        first = last = temp;
        return;
    }

    last->rlink = temp;
    temp->llink = last;
    last = temp;

}
```

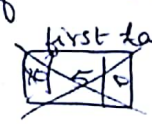
first = last = NULL







first = last = NULL → list empty

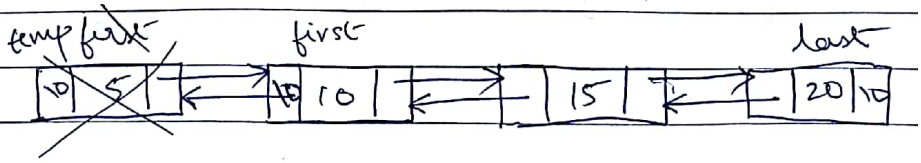


first = last = NULL

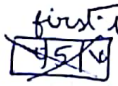
classmate Date Page

// FUNCTION TO DELETE A NODE AT THE
// BEGINING OF THE LIST

```
void delfront()
{
  NODE temp;

  if (first == NULL)
  {
     printf("List is empty\n");
     return;
  }

  else if (first -> rlink == NULL)
  {
     printf("%d is deleted\n", first->info);
     free(first);
     first = last = NULL;
     return;
  }

  else
  {
     temp = first;
     first = first -> rlink; first -> llink = NULL;
     printf("%d is deleted\n", temp -> info);
     free(temp);
     temp = NULL;
  }

}
```



first = last = NULL



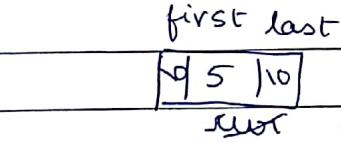
// FUNCTION TO DELETE A NODE FROM T
// END OF DLL.

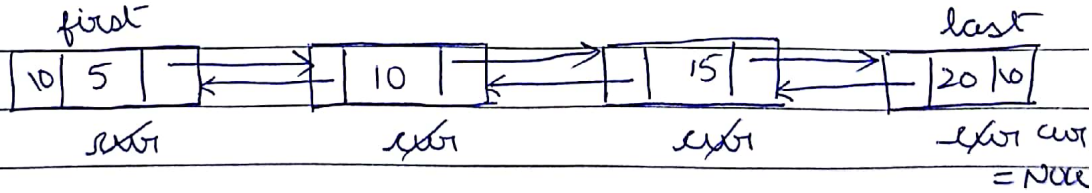
```
void delete_end()
{
    NODE temp;

    if(first == NULL)
    {
        printf("List is empty\n");
        return;
    }

    temp = last;
    printf

    else if(first->rlink == NULL)
    {
        printf("%d is deleted\n", first->info);
        free(first);
        first = last = NULL;
        return;
    }

    else
    {
        temp = last;
        last = last->llink;
        last->rlink = NULL;
        printf("%d is deleted\n", temp->info);
        free(temp); temp = NULL;
    }

}
```

// FUNCTION TO DISPLAY DLL

```
void display()
{
    NODE cur;

    if (first == NULL)
    {
        printf("List is empty\n");
        return;
    }

    printf("List elements are\n");
    cur = first;
    while (cur != NULL)
    {
        printf("%d\t", cur->info)
        cur = cur->rlink;
    }

}
```

first = last = NULL





// FUNCTION TO COUNT NO. OF NODES IN DLL

```
void countNodes()
{
  NODE cur;
  int count;
  if(first == NULL)
  {
     printf (" List is empty \n");
     return;
  }
    count = 0;
   cur = first;
   while(cur != NULL)
   {
      count ++;
      cur = cur -> rlink;
   }
   printf (" The DLL has %d no. of nodes \n", cou
}
```

classmate
Date
Page

//FUNCTION TO SEARCH FOR A KEY IN DLL

```
void linearsearch()
{
    int key, loc = 01;
    NODE cur;
    if (first == NULL)
    {
        printf("List is empty \n");
        return;
    }

    printf("Enter the key \n");
    scanf("%d", &key);
    cur = first;

    while (cur != NULL)
    {
        if (cur -> info == key)
        {
            printf("Key found at %d location", loc);
            return;
        }
        cur = cur -> rlink;
        loc++;
    }
    printf("Key not found \n");
}
```

```
//FUNCTION TO DELETE A NODE WHOSE INFORMATION
//IS SPECIFIED.

void delinfo()
{  NODE cur, prev, nxt;
   int key, flag=0;
   printf("Enter the key\n");
   scanf("%d", &key);
   if (first ==NULL)
   {
     printf("List is empty\n");
     return;
   }

   if(first -> rlink ==NULL)
   {
       if (key == first -> info)
       {
          printf("%d is deleted", first -> info);
          free(first);
          first= last = NULL; return;
       }
   }


   if (key == first -> info)
   {
      printf("%d is deleted", first -> info;
      cur= first;
      first = first -> rlink;
      first -> llink = NULL;
      free(cur);
      cur = NULL;
      return;
   }
```

```
if(key == last -> info)
{
    printf("%d is deleted", last -> info);
    cur = last;
    last = last -> llink;
    last -> rlink = NULL;
    free(cur);
    cur = NULL;
    return;
}

cur = first -> rlink;

while(cur != last)
{
    if(cur -> info == key)
    {
        prev = cur -> llink;
        next = cur -> rlink;
        printf("%d is deleted", cur -> info);
        prev -> rlink = Next;
        Next -> llink = prev;
        free(cur);
        cur = NULL;
        flag = 1;
    }
    cur = cur -> rlink;
}

if(flag == 0)
    printf("key not found\n");
}
```

// FUNCTION TO INSERT A NODE BEFORE A KEY

```
void insertbeforekey()
{
    NODE temp;
    int key;

    if(first == NULL)
    {
        printf("List is empty");
        return;
    }
    temp = (NODE) malloc(sizeof(struct node));
    printf("Enter the data to be inserted\n");
    scanf("%d", &temp->info);
    temp->rlink = temp->llink = NULL;

    printf("Enter the key");
    scanf("%d", &key);

    if(key == first->info)
    {
        temp->rlink = first;
        first->llink = temp;
        first = temp;
        return;
    }

    cur = first->rlink;

    while(cur != NULL)
    {
        if(cur->info == key)
        {
```

```
            prev = cur -> llink;
            prev -> rlink = temp;
            temp -> llink = prev;
            temp -> rlink = cur;
            cur -> llink = temp;
            return;
        }
        cur = cur -> rlink;
    }
    printf("Key not found\n");
}
```

// FUNCTION TO ENTER A NODE AT A SPECIFIC LOCATION IN DLL

```
void insertLoc()
{
    NODE temp, cur, prev
    int loc, count = 0

    temp = (NODE) malloc(sizeof(struct node));
    printf("Enter data\n");
    scanf("%d", &temp->info);
    temp->rlink = temp->llink = NULL;

    printf("Enter the location\n");
    scanf("%d", &loc);

    cur = first;

    while(cur != NULL)
    {
        count++;
        cur = cur->rlink;
    }

    if(count == 0 && loc == 1)
    {
        first = last = temp;
        return;
    }
```

```
if(loc > 0 && loc < (count/2))
{

   ~~cur = f~~
   if(loc == 1)
   {
      temp -> rlink = first;
      first -> llink = temp;
      first = temp;
      return;
   }

else
{
   cur = first -> rlink;
   i = 2;

   while (i != loc)
   {
      cur = cur -> rlink;
      ~~i++~~;
   }
   prev = cur -> llink;
   prev -> rlink = temp;
   temp -> llink = prev;
   temp -> rlink = cur;
   cur -> llink = temp;
   return;
}

}
```

P.T.O.

```
if(loc >= count/2 && loc < count+2
{

    if(loc = count+1)
    {
        last -> rlink = temp;
        temp -> llink = last;
        last = temp;
        return;
    }

    cur = last;
    i = count;

    while(i != loc)
    {
        i--;
        cur = cur -> rlink;
    }

    prev = cur -> llink;
    prev -> rlink = temp;
    temp -> llink = prev;
    temp -> rlink = cur;
    cur -> llink = temp;
    return;

}

printf(" Invalid location \n");
}
```

OPERATIONS OF STACK USING DLL

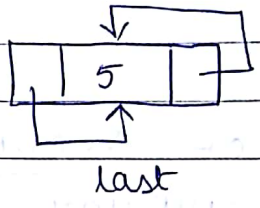
push = insertfront
pop = deletefront
display

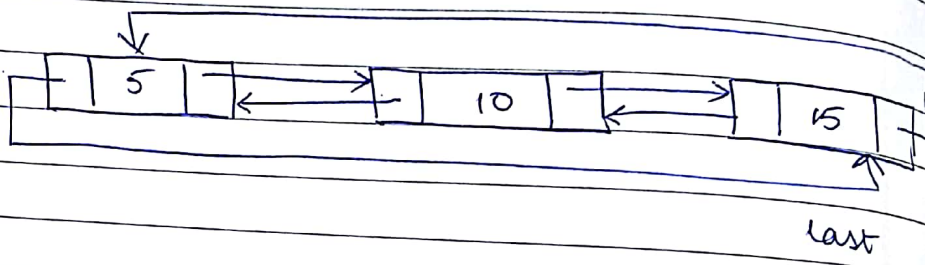
OPERATIONS OF QUEUE USING DLL

insert = insertrear
delete = deletefront
display

CIRCULAR DOUBLY LINKED LIST

last = NULL






```
struct node
{
    int info;
    struct node *llink;
    struct node *rlink;
};

typedef struct node * NODE;
NODE last = NULL;
```

// FUNCTION TO INSERT A NEW NODE IN
// A CIRCULAR DOUBLY LINKED LIST

```
void insert()
{
    NODE temp,nxt;

    temp = (NODE) malloc( sizeof (struct node));
    printf ("Enter the data\n");
    scanf("%d", &temp->info);
    temp->llink = temp->rlink = temp;

    if(last == NULL)
    {
        last = temp;
        return;
    }

    nxt = last->rlink;
    temp->rlink = nxt;
    nxt->llink = temp;
    last->rlink = temp;
    temp->llink = last;
    return;
}
```

// FUNCTION TO DISPLAY A CIRCULAR DLL

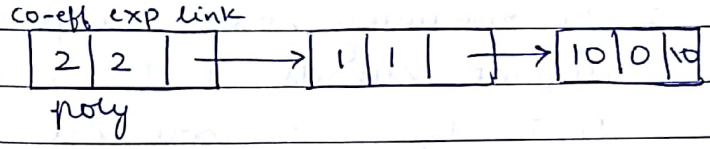
```
void display()
{
    NODE cur;

    if(first == NULL)
    {
        printf("List is empty\n");
        return;
    }
    printf("List elements are\n");
    cur = last -> rlink;
    while(cur != last)
    {
        printf("%d\t", cur->info);
        cur = cur -> rlink;
    }
    printf("%d", cur->info);
}
```

APPLICATIONS OF LINKED LISTS

- Implementing operations of stacks and queues using dynamic memory allocation.

- Linked lists can be used to represent polynomials.

  Ex: $2x^2 + x + 10$



- Linked lists can be used to represent trees.

  Ex: