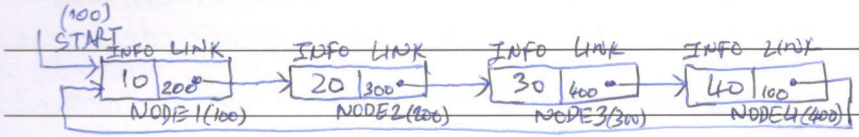
## Circular linked list (CLL)

It is a type of LL in which the link field of last node holds the address of starting node of the LL.

### Representation of CLL:




### Operations on CLL:

1) create 2) Traverse 3) Insertion 4) Deletion
5) Merging 6) Searching 7) Display.

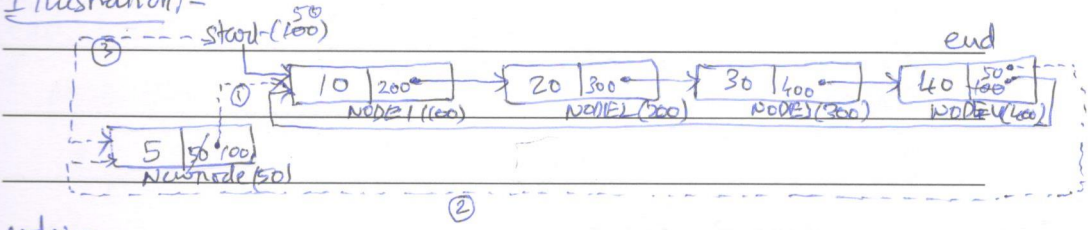
### Insertion in CLL:-

**case 1: Insert a node at the beginning of CLL:-**

Algorithm:-

step 1: start
2: create a newnode
3: store the data into the 'info' field of newnode
4: If the list is empty, then make start = end = newnode
5: otherwise make the 'link' field of the newnode to point to the start node and make the 'link' field of the end to point to the newnode. Then make start to the newnode.
6: stop.

Illustration:-



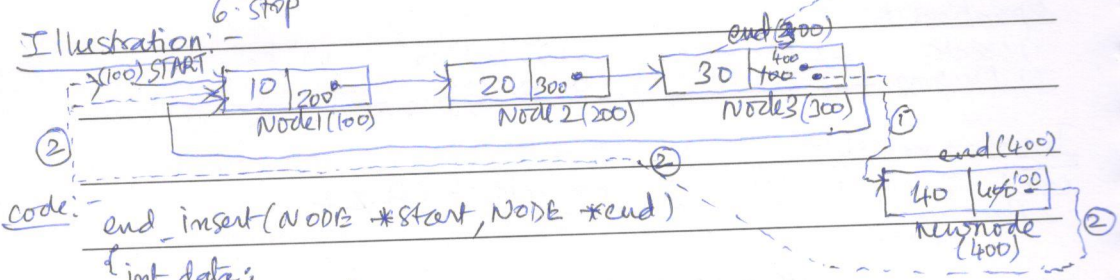

code:-

```
Begin insert(NODE *start, NODE *end)
{
NODE *newnode;
int data;
newnode = creat_node();
newnode -> info = data;
newnode -> link = newnode;
if(start == NULL)
{ start = end = newnode;
}
else
{
  newnode -> link = start;
  end -> link = newnode;
  start = newnode;
}
}
```

case-2: Insert a node at the end of CLL:-

Algorithm:- Step 1: Start
2: Create a newnode
3: Store the data into 'info' field of newnode
4: If the list is empty then make start = end = newnode
5: otherwise make the 'link' field of the endnode to the newnode and make the 'link' field of the newnode to point to 'start' node. Then make 'end' to newnode at end.
6. Stop

Illustration:-



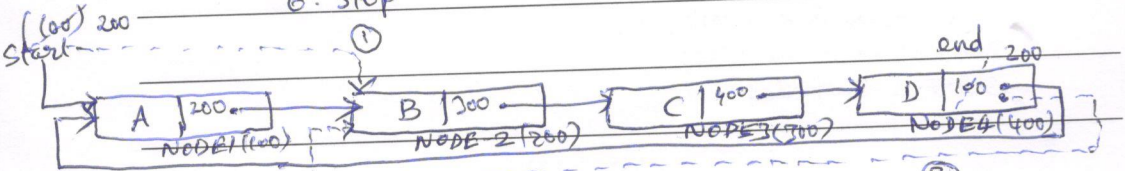

Code:-

```
end_insert(NODE *start, NODE *end)
{int data;
NODE *newnode;
newnode = create_node();
newnode->info = data;
newnode->link = newnode;
if(start == NULL)
{ start = end = newnode;
}
else
  {
    end->link = newnode;
    newnode->link = start
    end = newnode
}}
```

Deletion of a node in CLL:-

case 1: Deletion of a node at the beginning:-

Algorithm:- Step 1: Start
2: If the list is empty print "Deletion is not possible"
3: If start = end then free the 'start node' and then make start = end = NULL
4: Make the auxilary pointer to point to the start node and make the start to point to the next node in the list. Then set the 'link' field of the end node to point to wherever the 'start' is pointing to.
5 Delete the node pointed by auxilary variable
6. Stop



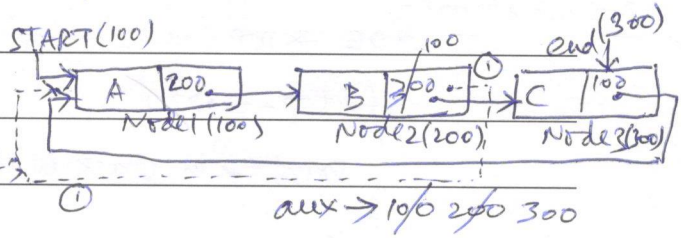
```
Begin_delete (NODE *end)
{
  NODE *aux;
  If(start == NULL)
    { printf("In list is empty");
    }
  else
    {
      aux = start;
     start = start->link
     end->link=start;
     free(aux);
    }
}
```

Case 2: Deletion of a node at the end of CLL

Algorithem:

Step 1: Start
2: If the list is empty then display "list is empty"
3: Suppose if start = end then make start = end = NULL;
4: Otherwise traverse the list till the end but one node and make its link field to point back to the start node
5. delete the endnode using free()
6. Stop

```
end_delete (NODE *start, NODE *end)
{
  NODE *aux;
  If(start == NULL)
    { printf("In The list is empty");
  else
    { if (start == end)
        { free(start);
          start = end = NULL;
        }
      else
        {
          aux=start;
          while (aux->link != end)
          { aux = aux->link;
          }
          aux->lik = start;
          free(aux);
          end = aux;
          notion(start);
        }
    }
}
```



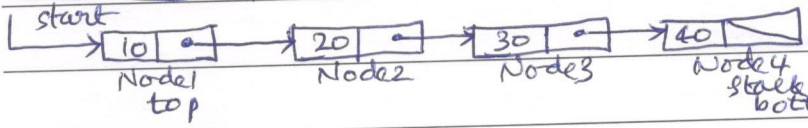
## Linked representation of stack:-

In this method the last node acts as bottom of the stack and front/starting node acts as beginning of stack/top of the stack.

If we want to push the element into stack, we need to attach the newnode to the front end of the list only.

while to delete an element, we can delete from the front end/starting node and next node --- etc.

## Representation of stack using linked list:-




## Push operation:-

```
NODE *push(NODE *top)
{
  NODE *newnode=create node();
  if(newnode==NULL)
  {
    printf("In stack is overflow");
  }
  else
  {
    printf("In Enter the data");
    scanf("%d", &newnode->info);
    newnode->link=top;
    top=newnode;
  }
}
```

not required as it is dynamic

Initial values:

```
NODE *top=NULL;
NODE *create node()
{
  NODE *newnode=
  (NODE*)malloc(sizeof
  (NODE));
  return(newnode);
}
```

## Pop operation:-

```
NODE *POP(NODE *top)
{
  if(top==NULL)
  {
    printf("In stack is underflow/empty");
  }
  else
  {
    printf("In The deleted item is %d", top->info);
    top=top->link;
  }
}
```

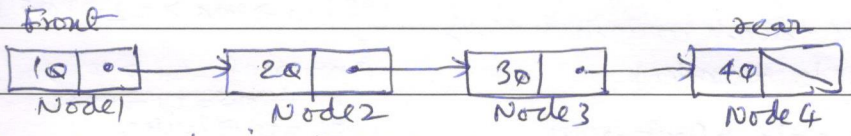
## Display operation:-

```
void display(NODE *top)
{
  if(top==NULL)
    printf("In Stack is empty");
  else
  {
    printf("The stack elements are");
    while(top)
    {
      printf("In %d", top->link);
      top=top->link;
    }
  }
```

## Linked list representation of Queue:-

In this linked list of queue the end of the list acts like rear-end and the beginning of the linked list acts as a front-end of the queue.

## Representation of LL Queue:-




## Deletion of a node in LL Queue:-

```
NODE *deletion(NODE *front)
{
  if(front==NULL)
  {
    printf("In Queue is empty");
  }
  else
  {
    printf("In The deleted element is %d", front->info);
    front = front->link;
  }
}
```

## Insertion of a node into LL Queue:-

```
insert(NODE *front, NODE *rear)
{
  NODE *newnode = create_node();
  printf("In Enter data");
  scanf("%d", &newnode->info);
  if(*front==NULL)
  {
    *front = newnode;
    *rear = newnode;
    newnode->link = NULL;
  }
  else
  {
    newnode->link = (*rear)->link;
    (*rear)->link = newnode;
    *rear = newnode;
  }
}
```

## Initial values

```
NODE *front, *rear;
*front = *rear = NULL;
```

```
NODE *create_node()
{
  return ((NODE*) malloc
    (sizeof(NODE)));
}
```

```
void display(NODE *front)
{
  if(front==NULL)
    printf("In Queue is empty");
  else
    while(front)
    {
      printf("%d\t", front->info);
      front = front->link;
    }
}
```

// Polynomial Addition

```
struct node
{
  int coef;
  int pow;
  struct node *link;
} *P1, *P2, *P3;
#include<stdio.h, conio.h, math.h>
struct node *create()
{
  struct node *first=NULL,
       *last, *nn;
  char ch;
  do
  {
    nn=(struct node*)malloc(sizeof
        (struct node));
    printf("\n Enter term coeff and
        Power ");
    scanf("%d%d", &nn->coef,
        &nn->pow);
    nn->link = NULL;
    if(first==NULL)
      first=last=nn;
    else
    {
      last->link=nn;
      last=nn;
    }
    printf("\n Do you want to
        Continue Y/n ");
    fflush(stdin);
    ch=getchar();
  }
  while(ch!='n');
  return(first);
}
void display(struct node *t)
{
  while(t!=NULL)
  {
    if(t->coef<0)
      printf("-");
    else
      printf("+");
    printf("%dx^%d",abs(t->coef)
        ,t->pow);
    t=t->link;
  }
}
```

```
void add(struct node *t1, node *t2)
{
  struct node *nn, *t;
  while(t1!=NULL && t2!=NULL)
  {
    nn=(struct node*) malloc(sizeof
        (struct node));
    nn->link=NULL;
    if(t1->pow > t2->pow)
    {
      nn->coef=t1->coef;
      nn->pow=t1->pow;
      t1=t1->link;
    }
    else if(t2->pow > t1->pow)
    {
      nn->coef=t2->coef;
      nn->pow=t2->pow;
      t2=t2->link;
    }
    else
    {
      nn->coef=t1->coef+t2->coef;
      nn->pow=t2->pow;
      t1=t1->link;
      t2=t2->link;
    }
    if(P3==NULL)
    {
      P3=nn;
      t=nn;
    }
    else
    {
      t->link=nn;
      t=t->link;
    }
  }
  while(t1!=NULL)
  {
    nn=(struct node*)malloc
      (sizeof(struct node));
    nn->coef=t1->coef;
    nn->pow=t1->pow;
    t->link=nn;
    t=nn;
    t1=t1->link;
  }
  nn->link=NULL;
```

```
while(t2!=NULL)
{
 nn=(struct node*)malloc
     (sizeof(struct node));
 nn->coef=t2->coef;
 nn->pow=t2->pow;
 t->link=nn;
 t=nn;
 t2=t2->link;
 nn->link=NULL;
}
}
void main()
{
 clrscr();
 printf("\n Enter Polynomial1 ");
 p1=create();
 printf("\n Enter Polynomial2");
 p2=create();
 clrscr();
 printf("\n First Polynomial is ");
 display(p1);
 printf("\n Second Polynomial is");
 display(p2);
 add(p1,p2);
 printf("\n Sum of two Poly
   nomials is ");
 display(p3);
}
```

// Merging of two lists

L1: 1→2→3→4→NULL
L2: 11→22→33→44→NULL

After Merging
L1: 1→11→2→22→3→33→4→44
→NULL.

```
struct node
{
 int data;
 struct node *link;
} *h1,*h2;
#include<stdio.h, conio.h>
struct node *create()
{
 struct node *first=NULL,*t,*nn;
 int d;
 while(1)
 {
  printf("\n Enter data,-1 to stop");
  scanf("%d",&d);
  if(d==-1)
   break;
  else
  {
   nn=(struct node*)malloc(sizeof
      (struct node));
   nn->data=d;
   nn->link=NULL;
   if(first==NULL)
     first=t=nn;
   else
   {
     t->link=nn;
     t=nn;
   }
  }
 return first;
}
void display(struct node *t)
{
 while(t!=NULL)
 {
  printf("%5d",t->data);
  t=t->link;
 }
}
void merge(struct node *t1,struct
 node *t2)
{
 struct node *f2;
 if(t1==NULL)
 {
  h1=t2;
  return;
 }
 if(t2==NULL)
   return;
 while(t1->link!=NULL && t2->link
 !=NULL)
 {
  f2=t2;
  t2=t2->link;
  f2->link=t1->link;
  t1->link=f2;
  t1=t1->link->link;
 }
 if(t1->link==NULL)
   t1->link=t2;
 else if(t2->link==NULL)
 {
  t2->link=t1->link;
  t1->link=t2;
 }
```

```
void main()
{
  clrscr();
  printf("\n Enter data for L1");
  l1 = create();
  printf("\n Enter data for L2");
  l2 = create();
  printf("\n List-1 is ...");
  display(l1);
  printf("\n List-2 is ...");
  display(l2);
  merge(l1, l2);
  printf("\n After merging the
      list is ...");
  display(l1);
  getch();
}
```

Adding Polynomials using linked lists

Example:- Polynomial-1 : $5x^2+4x+2 \Rightarrow 5x^2+4x^1+2x^0$

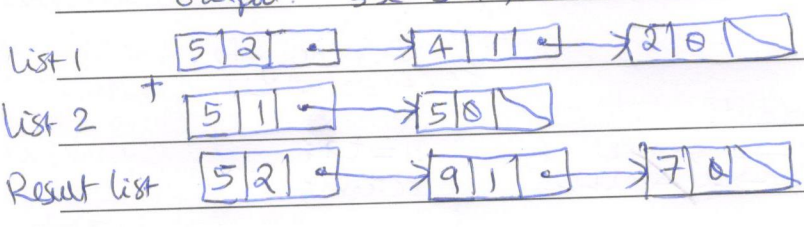
2: $5x+5 \Rightarrow 5x^1+5x^0$

output: $5x^2+9x^1+7x^0$

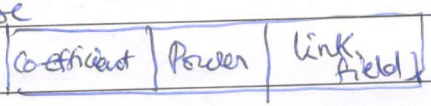
Example:2 Polynomial 1: $5x^3+4x^2+2x^0$

2: $5x^1+5x^0$

output: $5x^3+4x^2+5x^1+7x^0$

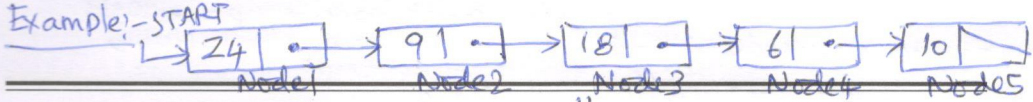



Node Structure



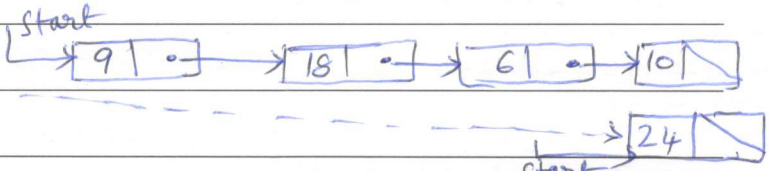
## Sorting the linked list :-

Example:- START

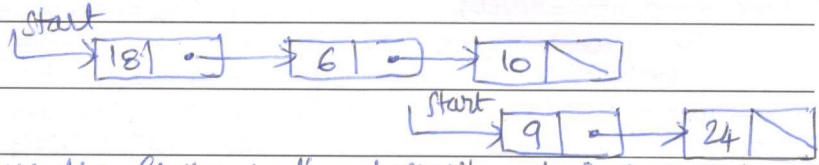



Step1: Remove the first node "node(24)" from the original list and assign the new head to that

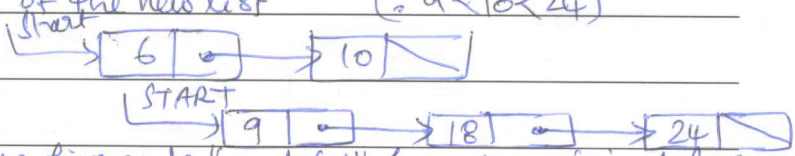
i.e



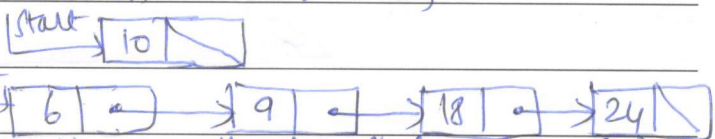

Step2: Remove the first node "node(9)" from the original list and insert before node(24) of the new list (∵ 9<25)



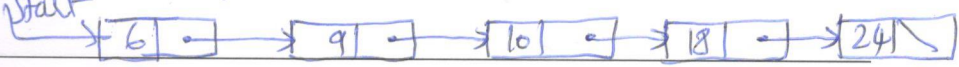

Step3: Remove the first node "node(18)" and insert it before node(24) of the new list (∵ 9<18<24)




Step4: Remove the first node "node(6)" from the original list and insert it before "node(9)" (∵ 6<9)




Step5: Remove the first node "node(10)" from the original list and insert it before node "node(18)" of new list




// Program for sorting the list items using insertion sort technique

```
#include<stdio.h>
#include<stdlib.h>
struct node{
        int info;
        struct node *next;
};
void print_list(struct node *start)
{
  printf("\n START -> ");
  while(start)
  { printf("%d->", start->info);
    start=start->next;
  }
  printf("NULL");
}
```

```
void insert_front(struct node **start,
                  int data)
{
 struct node *newnode=NULL;
 newnode=(struct node*)malloc
         (sizeof(struct node));
 if(newnode==NULL)
   printf("Failed to insert. out of memory");
 newnode->info=data;
 newnode->next=*start;
 *start=newnode;
}
void insert_sorted(struct node **start,
                   struct node *newnode)
{
 struct node *tmp_node=NULL;
 if(*start==NULL || newnode->info<
                    (*start)->info)
 {
  newnode->next=*start;
  *start=newnode;
  return;
 }
 tmp_node=*start;
 while(tmp_node->next && newnode->info
                    >tmp_node->next->info)
 {
  tmp_node=tmp_node->next;
 }
 newnode->next=tmp_node->next;
 tmp_node->next=newnode;
}
```

```
void sort_list(struct node **start)
{
 struct node *news_start=NULL;
 struct node *tmp=*start;
 struct node *tmp1=NULL;
 while(tmp)
 {
  tmp1=tmp;
  tmp=tmp->next;
  insert_sorted(&news_start,tmp1);
 }
 *start=news_start;
}
void main()
{
 int count=0,i,data;
 struct node *start=NULL;
 clrscr();
 printf("\n\n Enter no of nodes");
 scanf("%d",&count);
 for(i=1;i<=count;i++)
 {
  printf("\n Enter %d th element:",i);
  scanf("%d",&data);
  insert_front(&start,data);
 }
 printf("\n Linked list before sorting");
 print_list(start);
 sort_list(&start);
 printf("\n Sorted linked list is:");
 print_list(start);
 getch();
}
```