# Circular Link list

## # Creation of Circular link list

```
void create( )
{ int ch;
printf("Enter the choice except 1");
 node *temp,*ptr;
 Scanf("%d",&ch);
  while(ch!=1)
  {
   temp=(node *)malloc(sizeof(node));
  printf("\nEnter the data :");
  scanf("%d",&temp->data);
  if(start ==NULL)
  {
   start= temp;
   ptr= temp;
  }
  else
  {
   ptr->next =temp;
   ptr=temp;
  }
printf("\n Do you want to create another");
 Scanf( "%d",&ch);
}
ptr ->next=start;
}
```

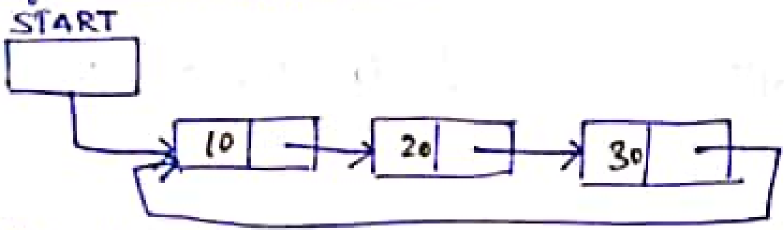
# # Display in Circular Link List

```
void display()
{
  node *ptr;
 Printf("The Linked List :\n");
  ptr=start;
 do
  {
  printf("%d|",ptr->data);
 printf("%p--->",ptr->next);
  ptr= ptr->next;
 }while(ptr!=start);
}
```

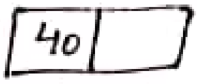
# Circular Linked list

## # Algorithm to insert an element in the begining of the list :-

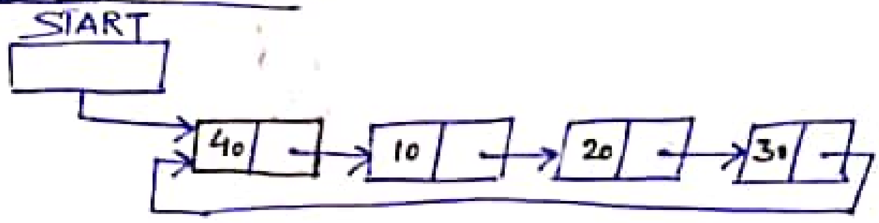
Before insertion :-



// Node to be inserted



After Insertion :-



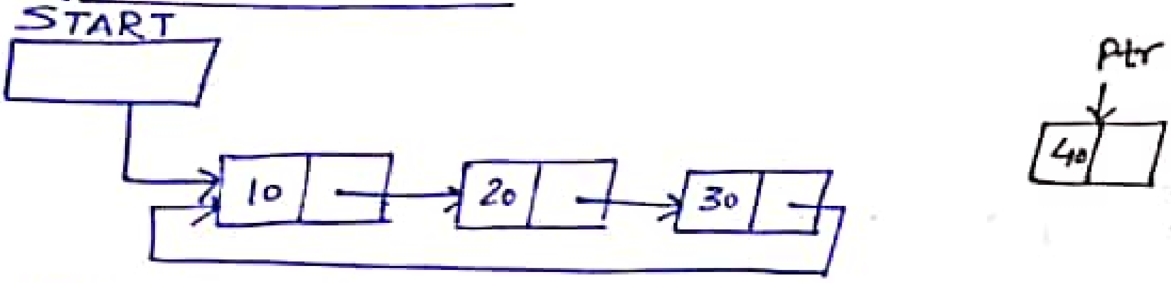
// Start will now contain address of new node and address of Start will be assigned to the next of new node.

```
Insertatbeg(START, NEXT, INFO, ELEMENT)
1) Create a node and address is assigned to Ptr.
2)   if ( Ptr = NULL)                 // Checks whether node
          Write : Overflow               Created or not.
               Exit
3)  Set INFO[Ptr] = ELEMENT
4)   if (START = NULL)                // If no node in
          Set NEXT[Ptr] = Ptr ;           list
                 START = Ptr
   else
   Set last = START
   while (NEXT[Last] != START)        // Since last node
   {                                     also contains the address
     set last = NEXT[Last]               of first node in Circular
   }                                     Link list.
```

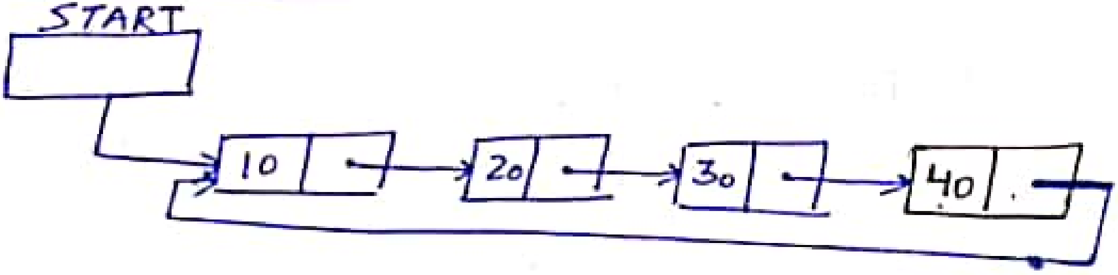
```
5) Set NEXT[Last] = Ptr        // Address of new node
                                  assigned to next
6) Set NEXT[Ptr] = START          pointer field of last
                                  node.
7) Set START = Ptr
8) Exit
```

# Algorithm to insert an element in the end of the list:-

Before insertion:-



After insertion:-

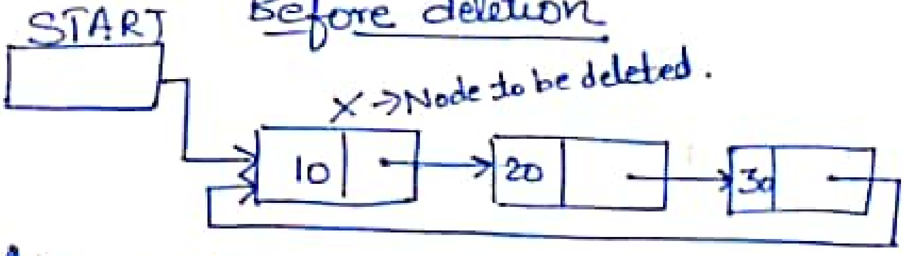


```
Insertatlast(INFO, NEXT, START, ELEMENT)
1) Create a node and address is assigned to Ptr.
2)    if (Ptr = NULL)
          Write: Overflow and Exit
3)    Set INFO(Ptr) = ELEMENT
4)    if (START == NULL)          // No node in list
          Set NEXT[Ptr] = Ptr     // Ptr is the first and
          Set START = Ptr            last node in that
                                     case
```

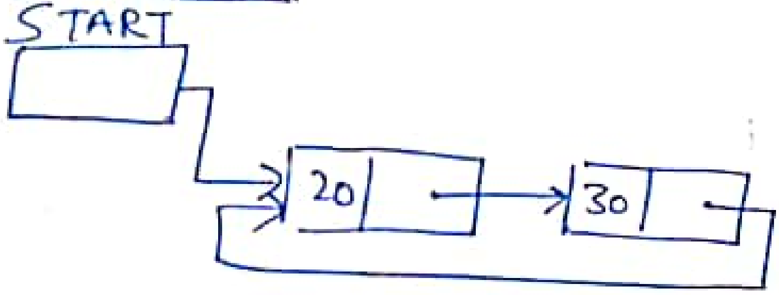
```
5) Set Temp = START        // Pointer for traversing to
                              track last node.
   while(NEXT[temp] != START)
   {
      Temp = NEXT[temp]
   }
6) Set NEXT[temp] = Ptr    // Last node found, now it
                              contains the address
                              of Ptr and Ptr is the
                              last node and its next
                              contains address of first
                              node.
7) NEXT [Ptr] = SPART
8) Exit
```

# Algorithm to delete a node in the beginning of the list :-

Before deletion



After deletion

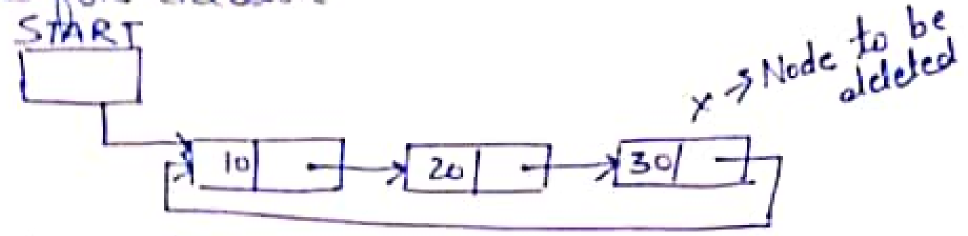


```
Deleteatbeg(START, NEXT, INFO)
1) if (START == NULL)
      Write: Underflow and Exit
2) Set Ptr = START         // Assigning the address of node
                              to be deleted to pointer Ptr.
```

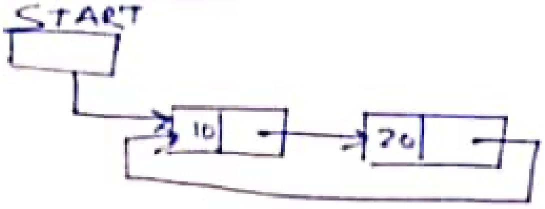
```
3) if (NEXT[START] = = START)          // that is only
                                          one node in list
    {
     temp = INFO[START]              // Data of first node stored
                                        in integer
                                        variable temp
     Set START=NULL
    }
   else
    {
    Set last = START
    while (NEXT[Last] != START]
     {
      Set last = NEXT[LAST]
     }
  Set NEXT[last] = NEXT[START]
   Set START = NEXT[START]
4) Set K= INFO[PtR]
5) free (Ptr)                    // Since initially the address
                                    of first node is stored in
                                    Ptr in point 2 of algorithm
6) Exit
```

# Algorithm to delete an element at the end of circular link list:-

Before deletion



After deletion



Algorithm

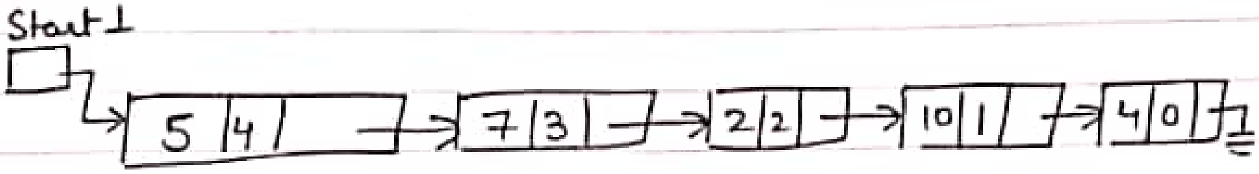
```
E  Deletatend( START, INFO, NEXT)
 i) if(START == NULL)
         Write: Underflow and Exit
 ii) if( NEXT[START] =START)     // if only one node in list
       {
         Set Ptr= START
         Set START = NULL
         Exit
       }
   else
     {
      last= START                // Pointer to traverse till last node.
      last1= NULL                // Pointer to remain one node before last.
   while( NEXT[Last] != START)
    {
     Set last1 = last
     Set last = NEXT [last]
    }
  Set NEXT[last1]= NEXT[last]    // When last node found give the address ~~to~~ in its next Pointer to preceeding node.
```

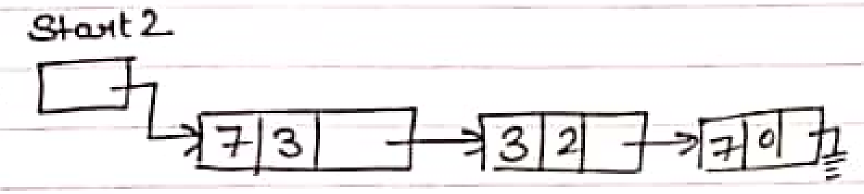
```
iii) Set K = INFO[last]
iv) free (last)
v) Print K
vi) Exit
```

# Addition of two polynomials

Eg;

$$\begin{array}{r} 5x^4 + 7x^3 + 2x^2 + 10x + 4 \\ 7x^3 + 3x^2 + 7 \\ \hline 5x^4 + 14x^3 + 5x^2 + 10x + 11 \end{array}$$





```
typedef struct nodetype
{
  int coeff;
  int expo;
  struct nodetype * next;
 }node;

node * start1 =NULL, *start2 =NULL, *ptr1, *ptr2,
  *start3 = NULL, *ptr = NULL, *temp, *start = NULL;

void create()
{
  int c=1;
  while (c!=0)
 {
  temp = (node *) malloc(sizeof(node));
  printf("Enter the coefficient\n");
   scanf("%d", &temp->coeff);
```

```
printf ("Enter the exponent\n");
 scanf ("%d", &temp→expo);
  temp→ next = NULL;
  if ( start == NULL)
  {
   start = temp;
   ptr = temp;
  }
 else
  {
   ptr→next=temp;
   ptr= ptr→next;
  }
  printf("Enter 0 to exit\n");
   scanf("%d",&c);
 }
}

void polyadd()
 {
  ptr1 = start1;
  ptr2 = start2;
 while(ptr1 != NULL && ptr2 != NULL)
  {
   temp= (node *)malloc(sizeof(node));
   if( ptr1 →expo > ptr2→ expo)
   {
    temp→expo = ptr1 →expo;
    temp→coeff = ptr1→coeff;
     ptr1 = ptr1→next;
   }
```

```
else if( ptr1->expo<ptr2->expo)
 {
  temp->expo=ptr2->expo;
  temp->coeff= ptr2->coeff;
    ptr2=ptr2->next;
 }
else
{
temp->expo= ptr1->expo;
 temp->coeff = ptr1 ->coeff + ptr2->coeff;
   ptr1=ptr1->next;
   ptr2= ptr2->next;
 }
 if (start 3==NULL)
  {
   start 3=temp;
    ptr=temp;
   }
  else
   {
    ptr->next=temp;
     ptr=ptr->next;
   }
 }
 while( ptr1 != NULL)
 {
  temp=(node*)malloc(sizeof (node));
    temp->expo=ptr1->expo;
    temp->coeff = ptr1->expo;
      ptr1=ptr1->next;
      ptr->next = temp;
         ptr=ptr ->next;
  }
```

```
while(ptr2 != NULL)
 {
 temp=(node*)malloc(sizeof(node));
  temp->expo =ptr2->expo;
   temp->coeff = ptr2->coeff;
    ptr2=ptr2->next;
   ptr->next =temp;
     ptr=ptr->next;
 }
 ptr->next = NULL; }

 void display()
  {
   ptr= start;
   while( ptr!=NULL)
    {
    printf("%d|%d->", ptr->coeff, ptr->expo);
     ptr=ptr->next,
    }
    printf("NULL\n");
 }
main()
{
 int ch;
printf("Enter1 to enter 1st polynomial\n Enter 2 to enter
      2nd polynomial\n Enter 3 to add\n Enter 0 to Exit\n");

while(1)
 {
 printf("Enter the choice\n");
scanf("%d", &ch);
switch(ch)
{
case 1: start=NULL;
        create();
        start1= start;
        display();
        break;
case 2: start= NULL;
        create();
        start2= start;
        break;
case 3: poly_add();
        start= start3;
        display();
        break;
case 0: exit(0);
        break; }}
```