19/05/21

3. Linked lists

Pointers :-

→ It is a variable to store the address of element

→ Pointers are used to access the information from a memory

→ Pointer is a variable to store the address of the another variable

Syntax :- datatype pointer variable ;

Eg :- int *a ; or int* a ;

Types of Pointers :-

→ There are 2 types of Pointers :-

1. Typed pointer
2. untyped pointer

Typed pointer :-

→ It Points to the specify the type of data.

Eg :- int *a, char *a[10]

untyped pointer :-

→ It can points to all the type of data.

→ It is also called as generic pointer.

Void * generic pointer

Two symbols :-

& :- This symbol it is used to give an address

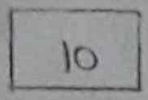
of the variable.

*:- It can store the address of the other variable.

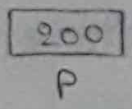
Program:-

```
void main( )
{
    int a=10;
    int *p;
    p=&a;
    Printf("%d", a); (10)
    Printf("%d", p); (200)
    Printf("%d", &a); (200)
    Printf("%d", &p); (300)
    Printf("%d", *p); (10)
}
```

consider,



a 200 → Address location



p

where,

p = pointer variable (used to store the address location of 'a')

→ For the above program 'a' gets allocated memory depending on the data type

→ At 200 address location 'a' variable is placed and it gets stored by a value of 10.

Structures:-

→ It is user defined data type.

→ using structures we can define a datatype which holds more than one element in different types.

→ struct is a keyword.

Syntax:- struct <identify>

Eg:- 1

```
struct student
{
    int rollno, year;
    char sname[20];
}
```

Eg:- 2

```
struct employee (4+20+4 = 28 bytes)
{
    int empno; (occupies 4 bytes of memory)
    char empname[20]; (occupies 20x1 = 20 bytes)
    float salary; (occupies 4 bytes)
}
```

20/05/21

→ In array we can insert the elements based on mentioned size. Beyond the mentioned size we cannot insert any other element and also we cannot update the size of array.

→ while performing deletion operation on array we should perform no. of shifting operations it will be easier when the size of an array is less than 10 or 20 if the size is greater it will be difficult for shifting.

→ These are the main drawbacks of an array

→ Arrays, stacks and queues are of static

memory collection type (continuous memory allocation)

→ It can be overcomed by introducing the concept called linked list.

Introduction to linked list :-

→ Linked list is a dynamic memory collection type.

→ In liked list the data will be stored in the form of nodes.

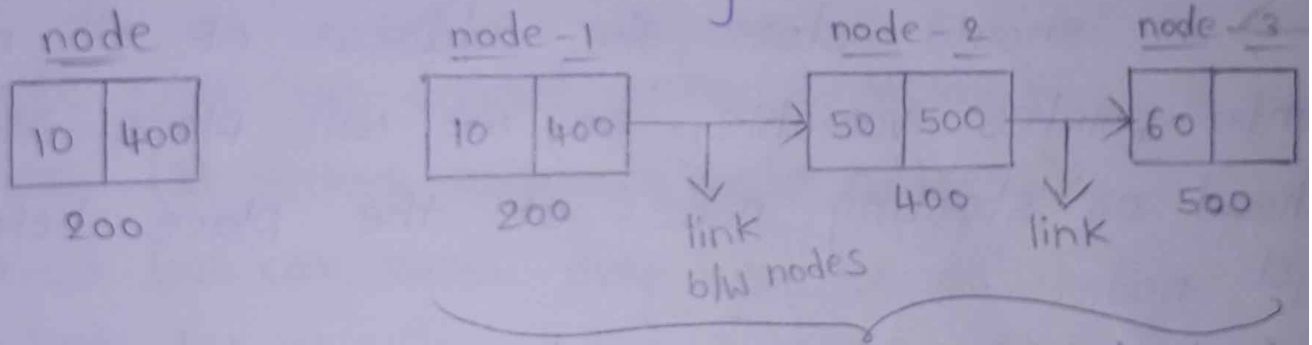
→ Insertion and deletion process are very simple compared to stacks, queues and arrays.

Node :-

→ Node is known as a separate memory block.

→ In linked lists the data will be stored in different or randaom memory locations.

Eg :-



→ we should link the nodes.

→ Different types of linked lists are :-

1. single linked list.
2. Double linked list.
3. circular linked list.

Single linked list :-

→ The data should be stored in the form of nodes.

→ To know the size of structure we will use malloc function.

Syntax :- malloc(size of (structure name));

→ Here malloc data type will be untyped pointer. The return type of untyped pointer is void but here we are placing struct node in place of void.

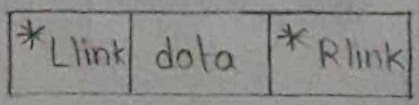
→ malloc() function may contain type casting

→ creating and allocating memory for a single linked list is given below :-

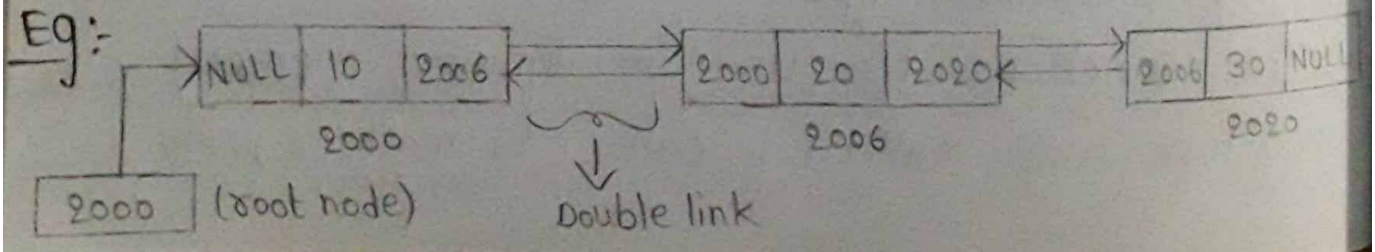
```
struct node
{
    int data;
    struct node *link;
};
struct node *root
root = (struct node *) malloc (size of (struct node));
```

continuation of Double linked list :-

Representation :-



Eg :-



→ In any node if Right link field contains NULL value then it is known as last node of the given double linked list.

creating and allocating memory for nodes using structures :-

```
struct node
{
  struct node *Llink;
  int data;
  struct node *Rlink;
};
struct node * root
root = (struct node *) malloc (sizeof (struct node));
```

circular linked list :-

→ In circular linked lists the data will be stored in the form of nodes.

→ These are classified into 2 types :-

i. single circular linked list.

ii. Double circular linked list.

single circular linked list :-

→ The data will be stored in the form of nodes.

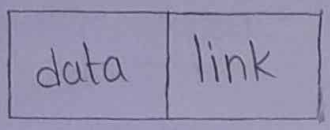
→ A node contains 2 fields :-

i. data field.

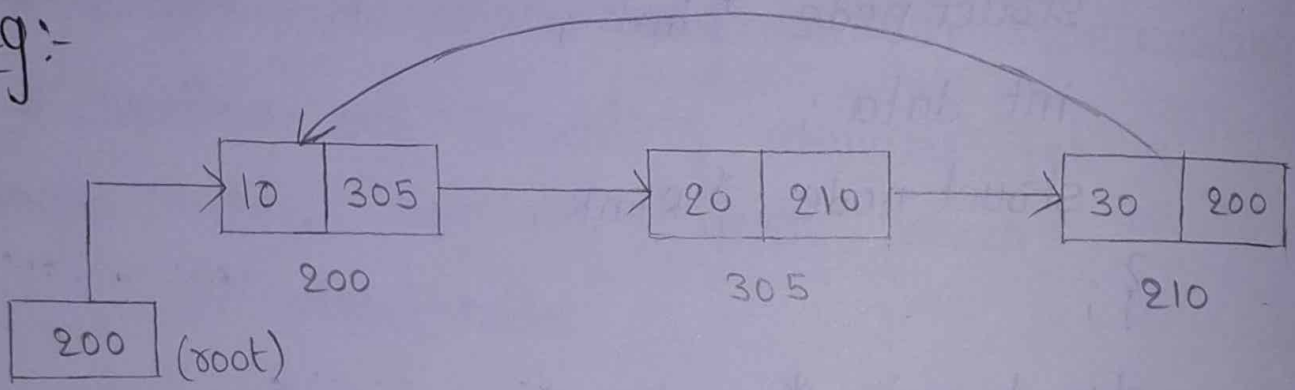
ii. link field.

→ If the last node link field contains the address of 1st node then the list is known as single circular linked list.

Representation :-



Eg :-



Double circular linked list :-

→ The data will be stored in the form of nodes.

→ A node having 3 fields :-

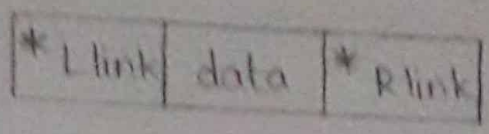
i. Left link

ii. Data field.

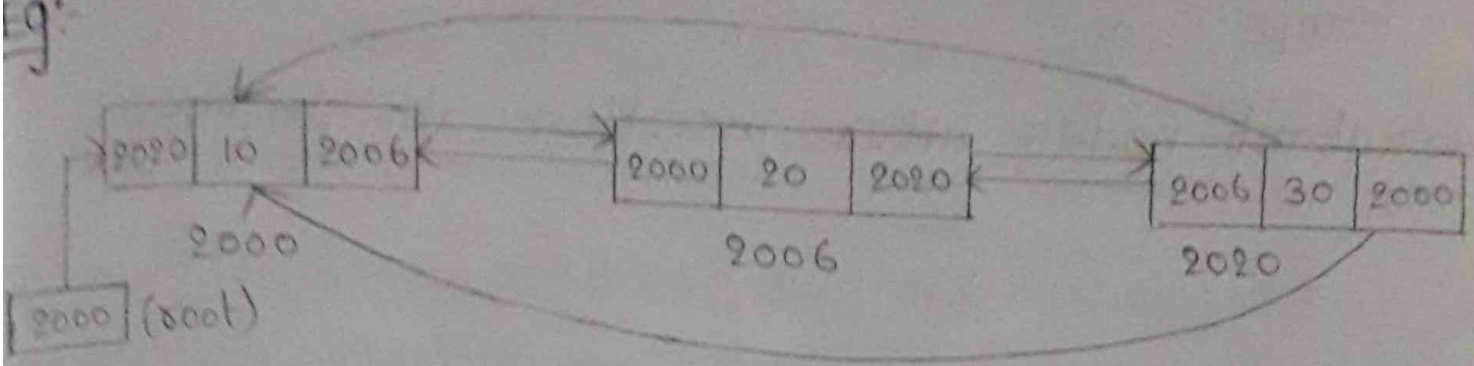
iii. Right link

→ The ~~left~~ Right link part of last node should contain the address of 1st node and left link part of 1st node should contain the address of last node then it is known as double circular linked list.

representation :-



Eg:



24/05/21

common operations performed on linked list :-

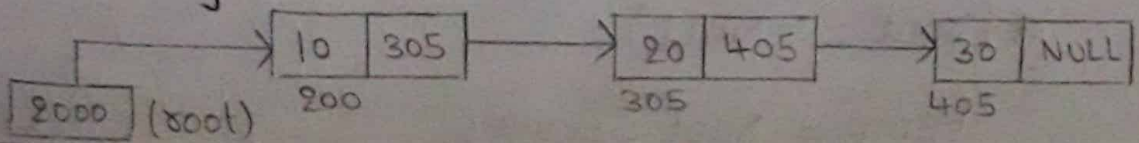
1. Insertion (Inserting a node into the given linked list)
2. Deletion (we should delete a node from given linked list)
3. Traversing (we should visit all the nodes from starting node to ending node in the given linked list atleast once)
4. searching (we have to search for a node in the given linked list)

operations on single linked list :-

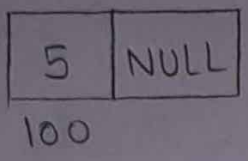
→ There are 3 ways to insert a node in the single linked list

i. Inserting a node at starting of list.
ii. Inserting a node at end of the list.
iii. Inserting a node at specified location.

(i) Inserting a node at starting of list :-

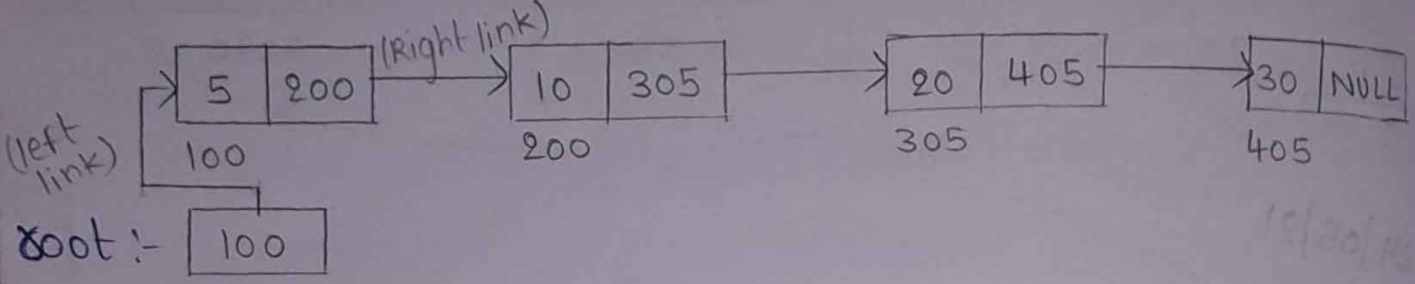


Notde to be inserted :-



temp : 100

Resultant nodes after insertion at starting :-



creation of Node while inserting at start of linked list :-

```
struct node
{
   int data;
   struct node *link;
};
struct node *root;
root = (struct node *) malloc (sizeof (struct node));
void startinglist ( )
{
   struct node * temp;
   temp = (struct node *) malloc (sizeof (struct node));
   printf (" Enter data");
   scanf (" %d", & temp -> data);
   temp -> link = NULL;
   if (root == NULL)
   {
      temp = root = temp;
```

(if 1st node link part is NULL then it is known as starting node)

}

25/05/21 else

{

temp→link = root ; (Right link) (NULL will be Replaced with root value)

root = temp ; (left link)

}

→ we should not disturb the root value while performing insertion and deletion of node

→ we should perform insertion and deletion by using temporary variable after performing the operation we should change the root value.

Algorithm :-

1. start
2. create a node and allocate the memory for that node.
3. insert the data i.e., temp → data
4. Assign temp → link = NULL
5. if root == NULL

root = temp ;

else

6. give the right link temp → link = root ;
7. give the left link root = temp ;
8. end

27/05/21

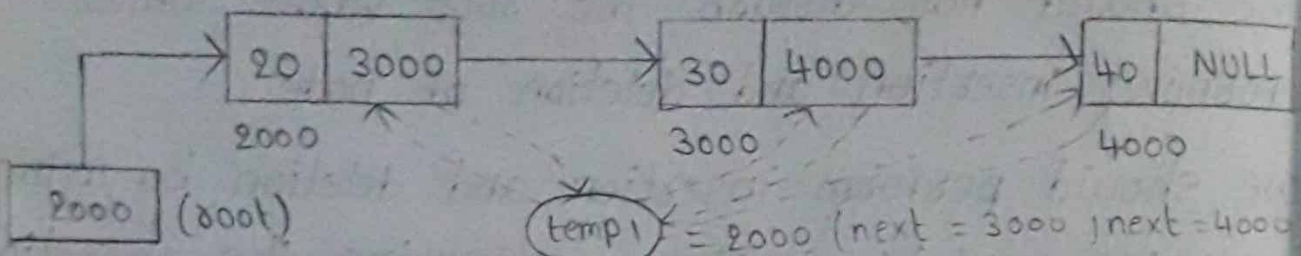
Inserting a node at end of the list :-

→ we have to perform traverse operation from

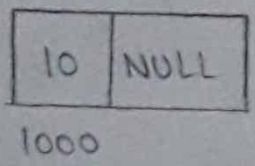
starting node to ending node. In traversing process we need to check whether the link field is NULL or not.

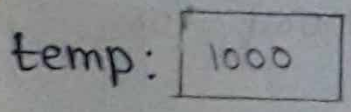
→ By using another temporary pointer variable we should perform traverse operation.

The linked list is :-



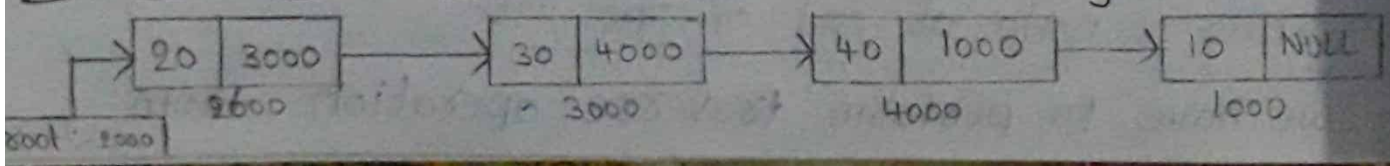
Node to be inserted :-





→ For traversing create temp1 variable and assign the root value to temp1 initially. In 1st case temp1 = 2000 and it points to 1st node and in 1st node link field = 3000 it points to 2nd node and in 2nd link field = 4000 it points to 3rd node but in 3rd node link field = Null. so, we should assign temp1 → link field = temp then the temp node is inserted at last.

Resultant node after insertion at ending :-



creating a node and performing traverse operation while inserting the created node at the ending of list :-

```
struct node
{
   int data;
   struct *link;
};
struct node *root;
root = (struct node *) malloc (size of (struct node));
void End of the list-( )
{
  struct node * temp;
  temp = (struct node *) malloc (sizeof (struct
                                        node));
  printf (" enter data");
  scanf (" %d", & temp -> data);
  temp -> link = NULL;
  if (root == NULL)
  {
     root = temp;
  }
  else
     {
       struct node * temp1;
       temp1 = root;
       while (temp1 -> link != NULL)
       {
```

```
        temp1 = temp1 -> link;
    }
    temp1 -> link = temp;
}
```

Algorithm :-

1. start
2. create a node and allocate the memory for that node.
3. insert the data i.e., temp -> data
4. assign temp -> link = NULL
5. if root == NULL
   root = temp;
   else
6. temp1 = root;
7. while(temp1 -> link != NULL)
8. temp1 = temp1 -> link repeate step until the condition is false
9. set temp1 -> link = temp
10. end

Inserting a node at a specified location :-

→ To insert a node at a specified location we should know whow many no. of nodes are available in the linked list.

→ so, we should find the length of the list.

To find length of the list :-

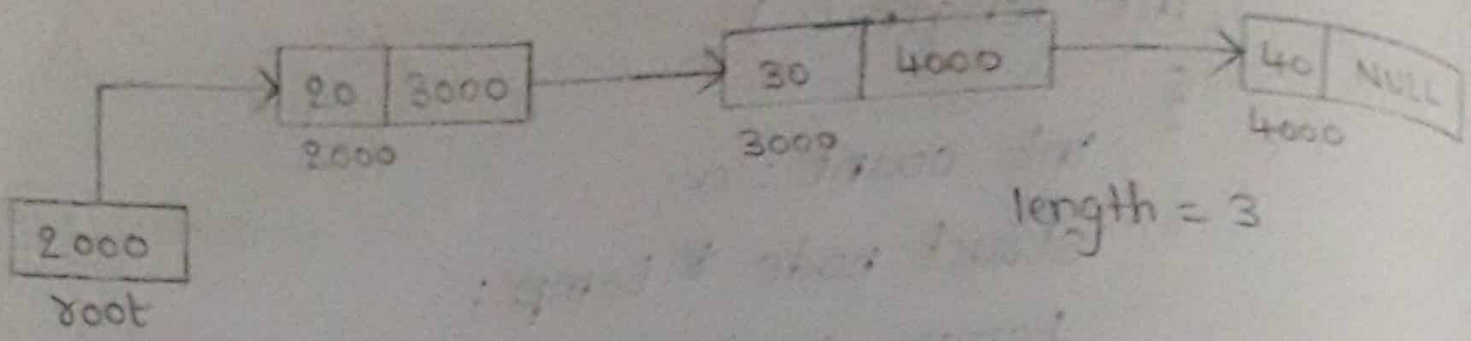
```
int length ( )
{
    int count = 0;
    struct node * temp;
    temp = root;
    while (temp != NULL)
    {
        count ++;
        temp = temp -> link;
    }
    return count;
}
```

Algorithm :-

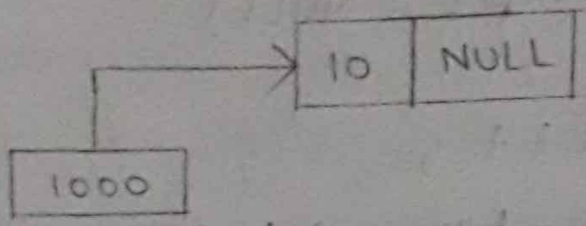
1. start
2. initialize count = 0
3. create temp pointer variable
4. temp = root;
5. while (temp != NULL)
6. increment count value by 1 (count ++)
7. temp = temp -> link; (repeated until the condition is false)
8. return count
9. end

31/05/21

The linked list is :-

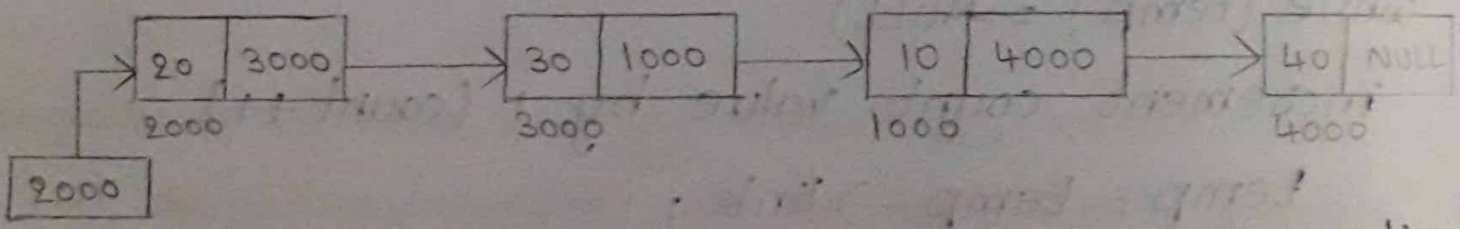


Node to be inserted :-



→ For inserting a node at specified location we need to create temp1 variable and assign root value to temp1. In the above case root points to 2000. so, temp1 will be 2000 and it points to 1st node and in the 1st node link field 3000 will get stored in temp1 and it points to 2nd node and it continues until temp1 reaches the n entered location. (while loop gets false)

Resultant nodes after inserting at specified location :-



creating a node and inserting at specified location :-

```
struct node
{
```

```
    int data;
    struct node *link;
}
struct node *root;
root = malloc(struct node *) malloc(size
                of(struct node));
void specifiedloc( )
{
    struct node *temp1;
    int loc, len, i=1;
    printf("enter location");
    scanf("%d", &loc);
    len = length( );
    if(loc>len)
    {
        printf("Invalid location")
    }
    else
    {
        temp1 = root;
        while(i<loc)
        {
            temp1 = temp1->link
            i++;
        }
    temp = (structnode*)malloc(sizeof(struct
                                node));
    printf("enter node data");
    scanf("%d", &temp->data);
```

```
        temp -> link = NULL;
        temp -> link = temp1 -> link;
        temp1 -> link = temp;
    }
```

To Display all the nodes in the linked list :-

```
void display()
{
    struct node * temp;
    temp = root;
    if (temp == NULL)
    {
        printf("No nodes are available in the
                                    list");
    }
    else
    {
        while (temp != NULL)
        {
            printf("%d", temp -> data);
            temp = temp -> link;
        }
    }
}
```

To search for an element in the given linked list :-

```
void search()
{
    struct node * temp;
```

```
int key, flag = 0;
temp = root;
printf(" Enter key value ");
scanf (" %d", & key);
while (temp != NULL)
{
   if(key == temp -> data)
   {
      flag = 1;
      break;
   }
   else
   {
      temp = temp -> link;
   }
 }
 if(flag == 1)
 {
   printf(" search element is available");
 }
else
{
   printf(" search element is not available");
}
}
```

02/05/21

Deletion operation on the single linked list :-

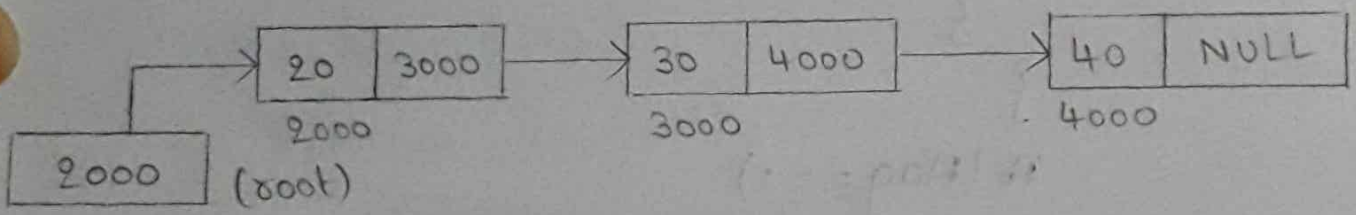
→ There are 3 ways to delete a node from the single linked list

1. Delete a node at starting of the list.
2. Delete a node at end of the list.
3. Delete a node at specified location of the list.

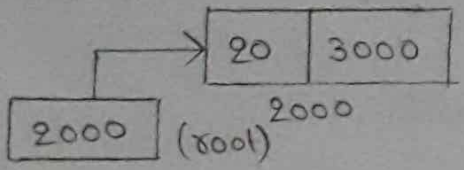
Delete a node at starting of the list :-

→ By using free function we can delete the memory for a node usi in the linked list. (dynamic memory allocation)

The linked list is :-

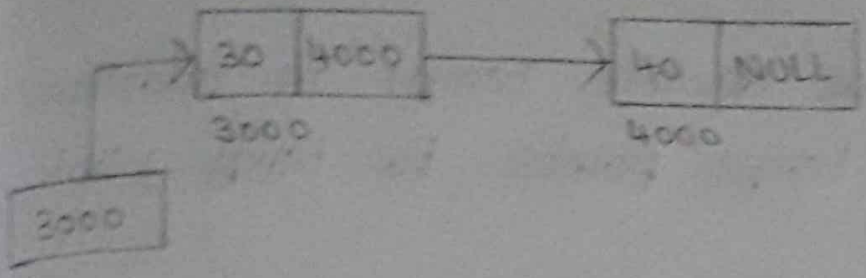


Node to be deleted :-



→ To delete a node we should creat temp variable and we should store root value to temp and we should remove both right link and left link of starting node. That is, temp points to 1st node and temp → link field should be replaced with NULL and root should be replaced with temp → link. So, that left and right links are removed.

Resultant nodes after deleting a node at starting :-

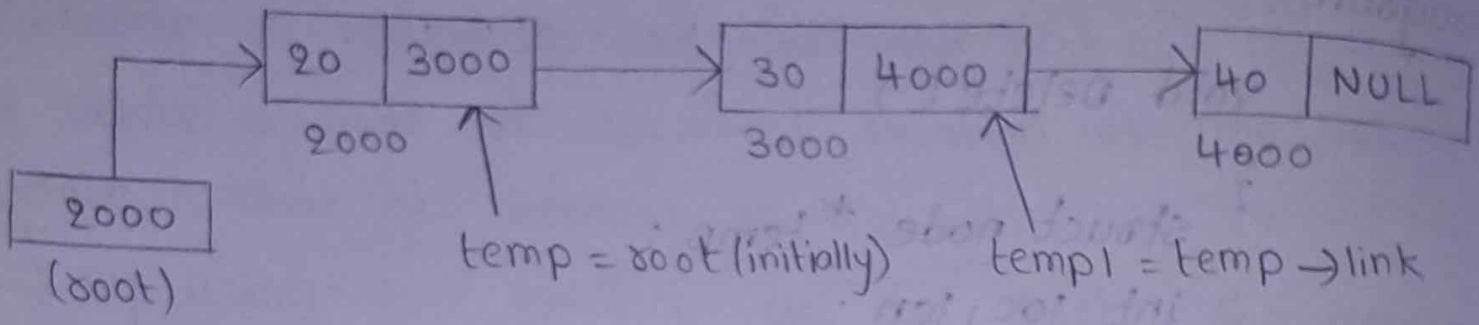


program :-

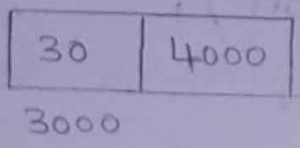
```
void Delete()
{
  struct node *temp;
  int loc, len;
  len = length();
  printf("enter location");
  scanf("%d", &loc);
  if(loc > len)
  {
    printf("invalid location");
  }
  else
  {
    if(loc == 1)
    {
      temp = root;
      temp->link = root;
      temp->link = NULL;
      free(temp); [the memory allocated for
                   the node named temp will
                   be deleted]
    }
  }
}
```

Delete a node at specified location of the list :-

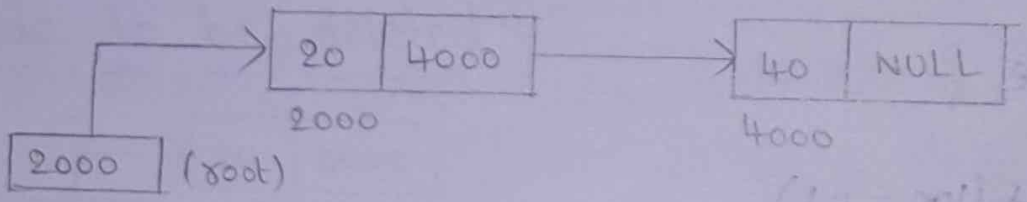
→ In this case we declare temp1 in addition with temp where temp1 points to temp → link. The linked list is :-



Node to be deleted :-



Resulting nodes after deleting a node at specified location :-



Program :-

```
Void specifiedloc( )
{
    struct node *temp = root, *temp1;
    int loc, len;
    len = length( );
    printf("enter location");
    scanf("%d", &loc);
```

```
if (loc>len)
{
  printf("Invalid location");
}
else
{
   int i=1;
   while (i<loc-1)        (if condition true i.e.,
                           loc=3 => 1<2)
   {
     temp = temp->link;
     i++;
   }
   temp1 = temp->link;
   temp->link = temp1->link;
   temp1->link = NULL;
   free(temp1);
}
}
```

loc=2 => 1<1 (condition false)

Delete a node at the end of the list:-

→ The processor is same as above but we should remove temp1->link = NULL because for the statement temp->link = temp1->link then automatically NULL will be placed in temp->link field.

Program:-

```
void end of ()
{
   struct node * temp=root, temp1;
```

```
int loc, len;
len = length();
Printf("enter location");
scanf("%d", &loc);
if(loc > len)
{
  printf("Invalid location");
}
else
{
  int i=1;
  while(i<.loc-1)
  {
    temp = temp -> link;
    i++;
  }
  temp1 = temp -> link;
  temp -> link = temp1 -> link;
  free(temp1);
}
}
```

loc = 3 , i = 1

1 < 3-1 => 1 < 2 (condition true)

i = 1+1 = 2

2 < 3-1 => 2 = 2 (condition false)

03/06/21

Double linked list :-

→ The data can be stored in the form of nodes

operations on double linked list :-

1. Insertion
2. Deletion
3. searching
4. Display

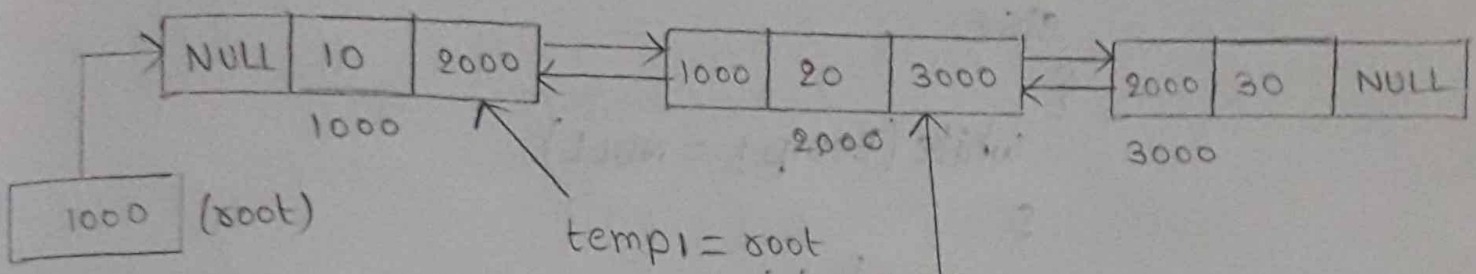
Insertion operation :-

→ There are three possibilities :-

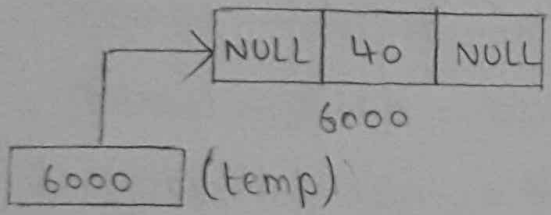
1. Inserting a node at starting of the list.
2. Inserting a node at end of the list.
3. Inserting a node at specified location.

Inserting a node at specified location :-

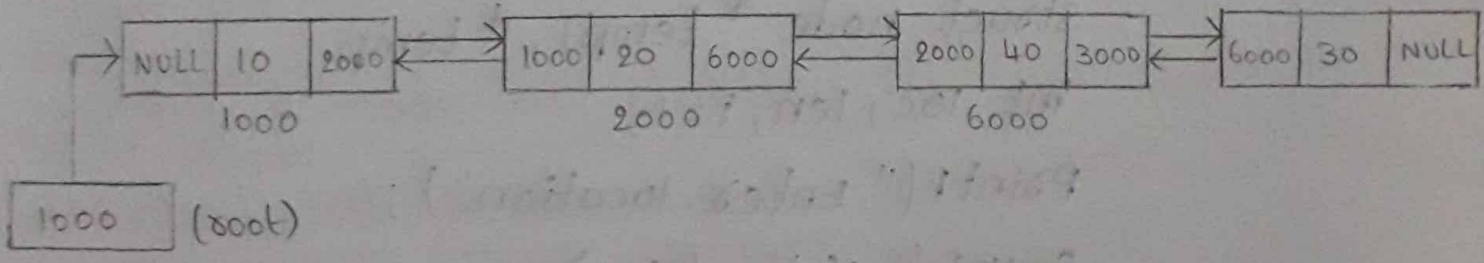
The linked list before insertion :-



Node to be inserted :-



Resultant nodes after insertion :-



→ For inserting at specified location, we should know the length of the linked list. To know the length we should create a temp1 variable and initialize the value of temp1 to root and perform traverse operation until the right link part of node is equal to NULL (before insertion)

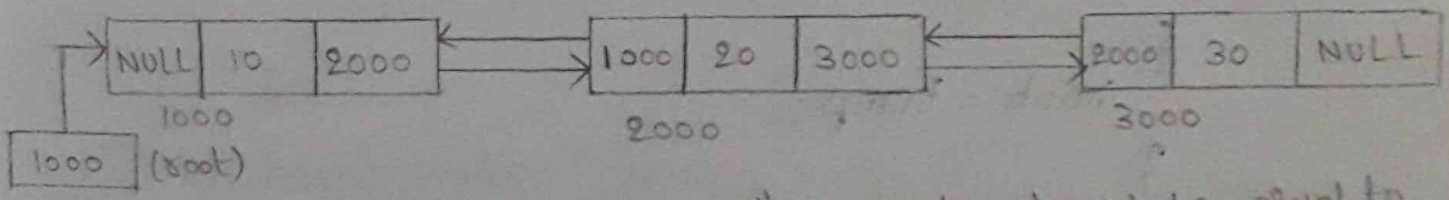
```
int length()
{
  struct node *temp;
  temp = root;
  int count = 0;
  if(temp == NULL)
  {
    printf("there is no list available");
  }
  else
  {
    while(temp != NULL)
    {
      count ++;
      temp = temp -> rlink;
    }
    return count;
  }
}
void addloc()
{
  struct node *temp1, *temp
  int loc, len, i=1;
  printf("Enter location");
  scanf("%d", &loc);
  len = length();
  if(loc>len)
  {
    printf("Invalid location");
  }
else
```

(temp1 variable is used to traverse from starting node to the provided location node) initially temp1 = root

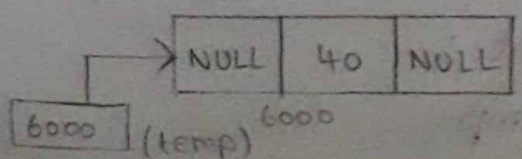
```
{
    temp = (struct node *) malloc (sizeof (struct node));
    printf(" enter node data ");
    scanf (" %d ", & temp -> data);
    temp -> llink = NULL ;
    temp -> rlink = NULL ;
    temp1 = root ;
    while (i<loc)
    {
        temp1 = temp1 -> rlink
        i++ ;
    }
    temp -> rlink = temp1 -> rlink ;
    temp1 -> rlink -> llink = temp ;
    temp -> llink = temp1 ;
    temp1 -> rlink = temp ;
}
}
```

Inserting a node at starting of the list :-

The linked list before insertion :-

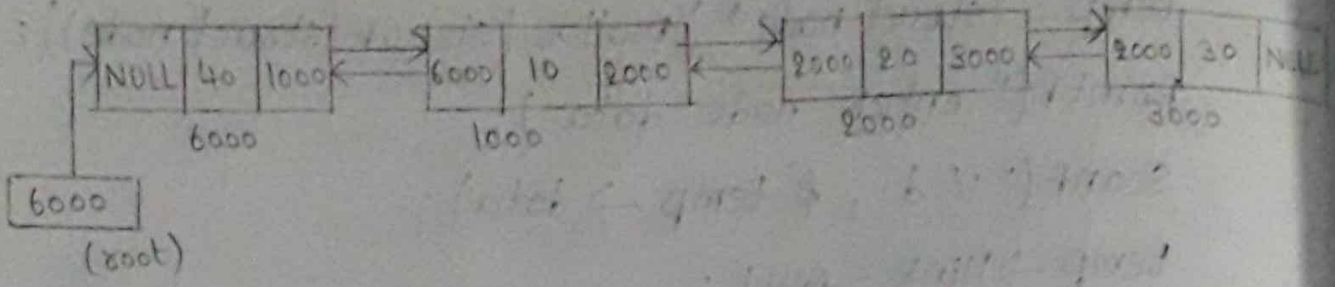


Node to be inserted :-



then root should be equal to temp value and root -> llink should be temp (6000) and temp -> rlink should be 1000 (any where 1st we should update rlink values

Resultant nodes after insertion :- after llink values and last root value



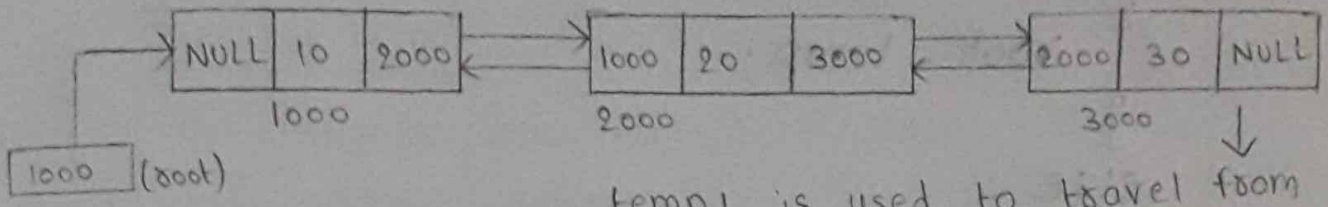
Program :-

```
struct node
{
  int data;
  struct node *llink,*rlink;
};
struct node *root;
root = (struct node*) malloc(sizeof(struct node));
void startinglist()
{
  struct node *temp;
  temp=(struct node*) malloc(sizeof(struct node));
  printf("enter data");
  scanf("%d", &temp->data);
  temp->llink = NULL;
  temp->rlink = NULL;
  if(root == NULL)
  {
    root = temp;
  }
  else
  {
    temp->rlink = root;
    root->llink = temp;
```

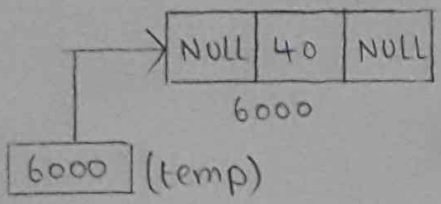
```
        root = temp;
      }
    }
```

Inserting a node at end of the list :-

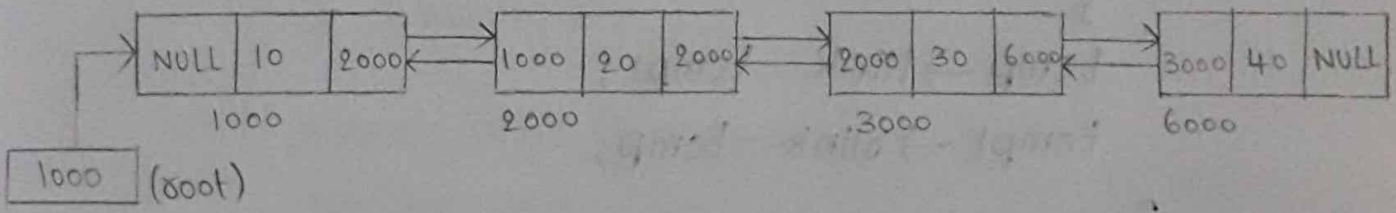
The linked list before insertion :-



Node to be inserted :-



Resultant nodes after insertion :-



Program :-

```
struct node
{
  int data;
  struct node *llink, *rlink;
};
struct node *root;
root = (struct node *) malloc (sizeof (struct node));
void end of list ( )
{
  struct node *temp;
```

```
temp = (struct node *) malloc (sizeof (struct node));
printf("enter data");
scanf("%d", &temp->data);
temp->llink = NULL;
temp->rlink = NULL;
if(root == NULL)
{
    root = temp;
}
else
{
    temp -> struct node *temp1;
    temp1 = root;
    while (temp1 != NULL)
    {
        temp1 = temp1->rlink;
    }
    temp->llink = temp1;
    temp1->rlink = temp;
}
}
```

07/06/21

Deletion operation on double linked list :-

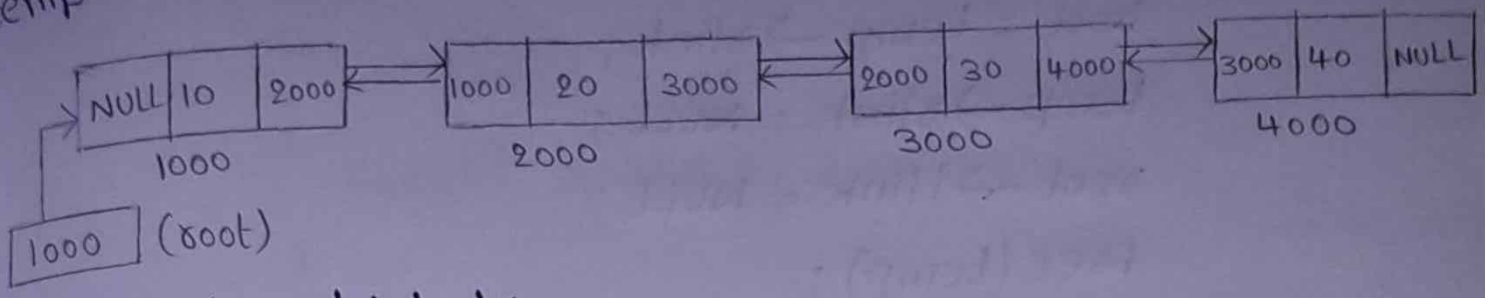
→ There are 3 possibilities :-

1. Deleting a node at starting of the list.
2. Deleting a node at end of the list
3. Deleting a node at specified location of the list.

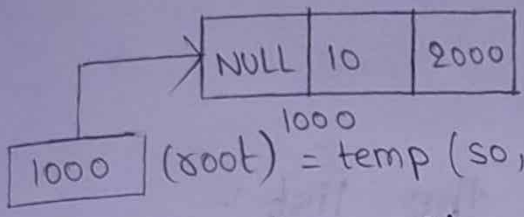
Deleting a node at starting of the list :-

→ Take a temp variable and assign root value to the

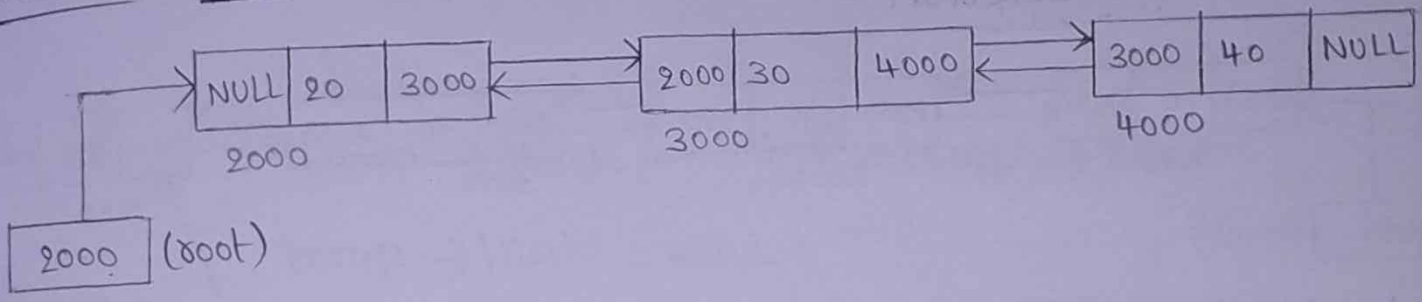
temp.



Node to be deleted :-



Resultant nodes after deletion :-



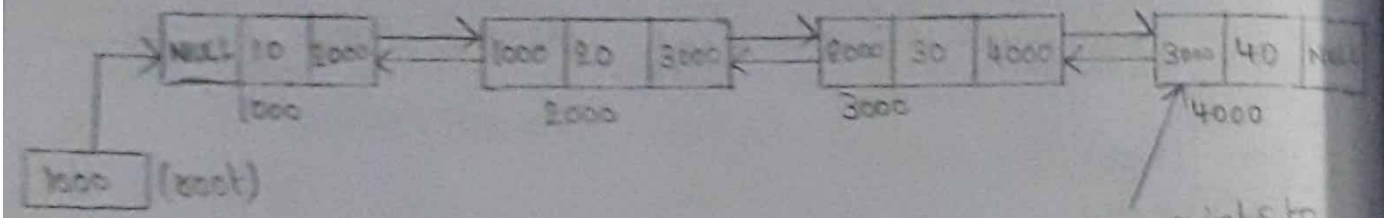
program :-

```
Void startinglist()
{
   struct node *temp;
   int loc, len;
   len = length();
   printf("enter location");
   scanf("%d", &loc);
  if(loc > len)
  {
    printf("Invalid location");
  }
  else
  {
     if(loc==1)
     {
```

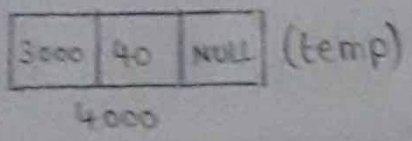
```
        temp = root;
        root = temp->rlink;
        temp->rlink = NULL;
        root->llink = NULL;
        free(temp);
      }
    }
  }
```

Deleting a node at the end of the list :-

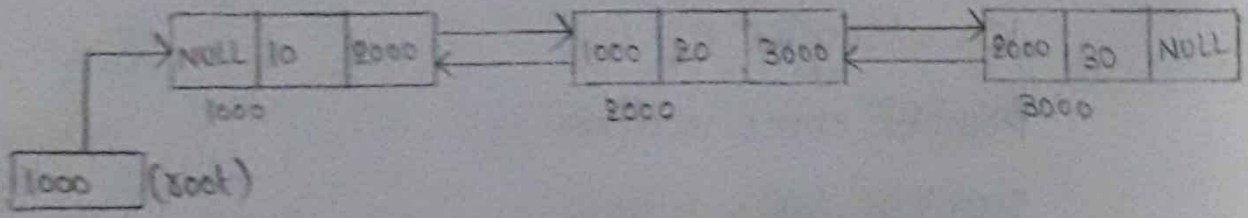
Nodes before deletion :-



Node to be deleted :-



Resultant nodes after deletion :-



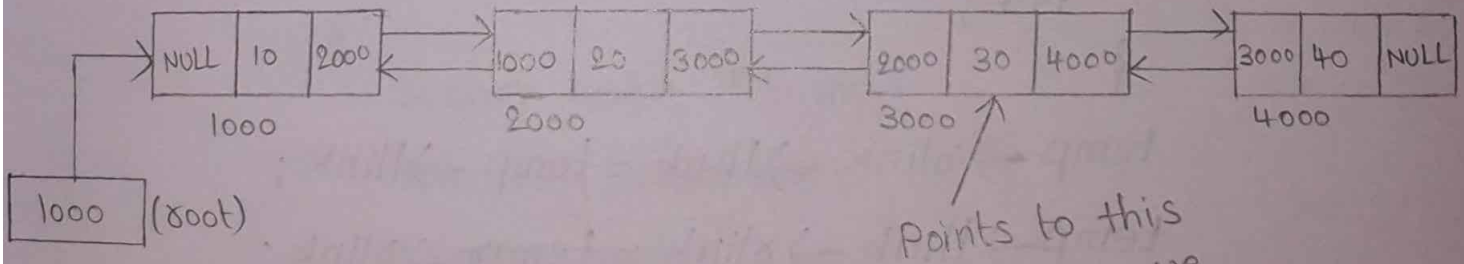
Program :-

```
void endofthelist()
{
  struct node *temp;
  int loc, len, i=1;
  len = length();
  printf("enter location");
```

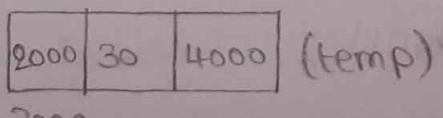
```
scanf("%d", &loc);
if(loc > len)
{
  Printf("Invalid location");
}
else
{
  while(i<=loc-1)
  {
    temp = temp -> rlink;
    i++;
  }
 temp -> llink -> rlink = temp -> rlink;
temp -> llink = NULL;
free(temp);
}
}
```

Deleting a node at specified location :-

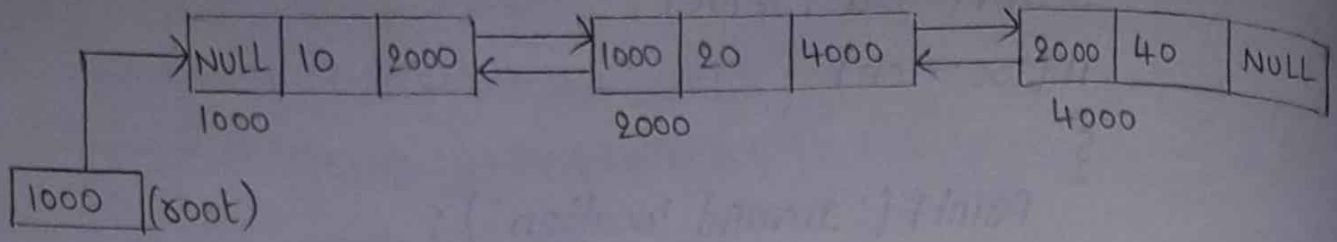
Nodes before deletion :-



Node to be deleted :-



Resultant nodes after deletion :-



Program :-

```
Void specifiedlist ( )
{
    struct node * temp ;
    int loc, len, i=1 ;
    len = length ( ) ;
    printf ("enter location") ;
    scanf ("%d", & loc) ;
    if (loc > len)
    {
      printf ("Invalid location") ;
    }
    else
    {
      while (i<=loc-1)
      {
        temp = temp -> rlink ;
        i++ ;
      }
    temp -> rlink -> llink = temp -> llink ;
    temp -> llink -> rlink = temp -> rlink ;
    temp -> rlink = NULL ;
    temp -> llink = NULL ;
    free (temp) ;
    }
}
```

To display all the nodes in the linked list :-

```
Void display( )
{
  struct node *temp;
  temp = root ;
  if(temp == NULL)
  {
    Printf (" No nodes are available");
  }
  else
  {
    while (temp != NULL)
    {
      Printf(" %d", temp -> data);
      temp = temp -> rlink ;
    }
  }
}
```

To search for an element in the double linked list :-

```
Void search ( )
{
  struct node *temp;
  int key, flag = 0;
  temp = root;
  Printf(" Enter key value");
  Scanf(" %d", & key);
  while (temp != NULL)
  {
    if(key == temp -> data)
```

```
        {
            flag = 1;
            break;
        }
        else
        {
            temp = temp -> rlink;
        }
    }
    if(flag == 1)
    {
        Printf(" search element is available ");
    }
    else
    {
        Printf(" search element is not available");
    }
}
```

08/06/21

operations to be performed on circular linked list:-

→ circular linked lists are of two types:-

1. single circular linked list
2. Double circular linked list

→ In circular linked list the tail pointer is used instead of root pointer to store the address of last node.

Insertion operation on single circular linked list:-

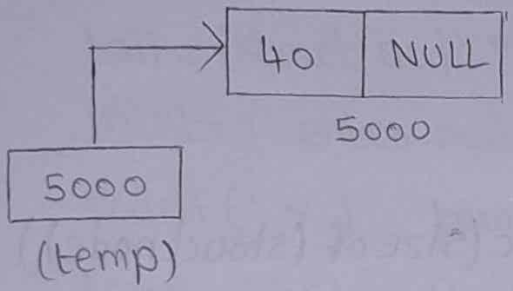
→ There are 3 possibilites:-

1. Inserting a node at starting of the linked list.
2. Inserting a node at ending of the linked list.
3. Inserting a node at specified location of the linked list.

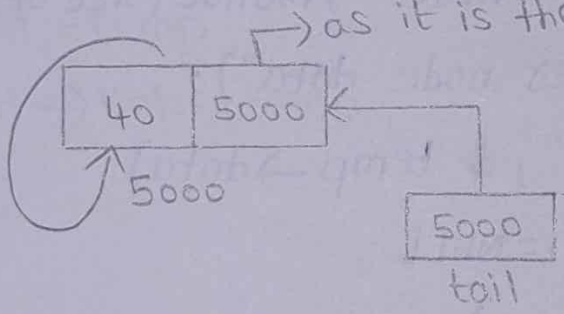
Insertion at starting of linked list :-

→ If tail is NULL then the list will be empty

Node to be inserted :-



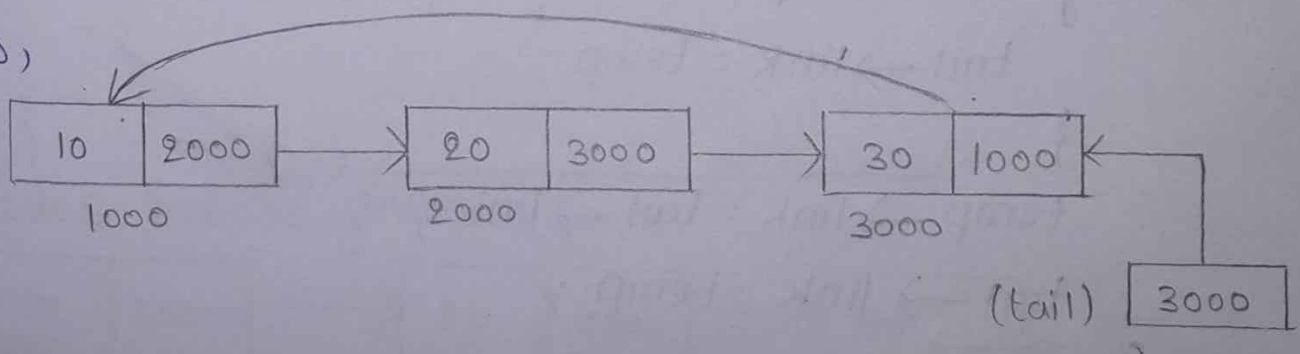
→ if tail is NULL then the node which we are inserting will be the 1st node and last node of the linked list. Then the list will be,



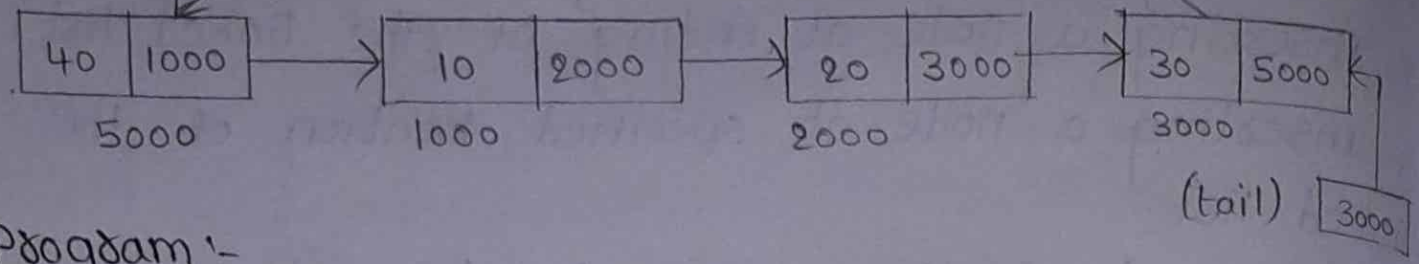
→ as it is the 1st node and last node it points to itself. so, link field will be 5000

→ Here tail = 5000. So, the tail points to the above inserted node.

→ If the linked list contains some nodes as below,



→ After insertion,



Program :-

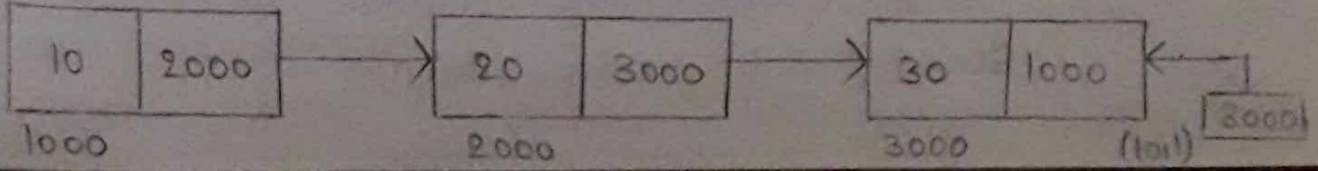
```
struct node
{
   int data;
   struct node *link;
};
struct node * tail;
tail = (struct node*) malloc(sizeof(structnode))
void insert begin()
{
   struct node *temp;
   temp = (struct node *) malloc (size of (struct node));
   printf("enter node data");
   scanf("%d", &temp->data);
   temp->link = NULL;
   if(tail == NULL)
   {
     printf("no nodes are available in the list");
   } tail = temp;
     tail->link = temp;
   }
   temp->link = tail->link;   } else {
   tail-> link = temp;
}
```

Insertion at end of the list :-

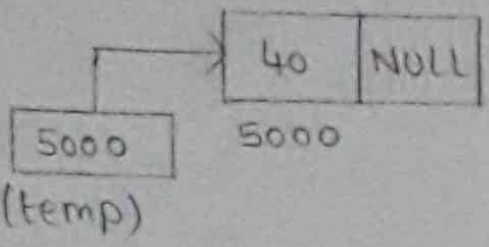
program :-

```
struct node
{
    int data;
    struct node *link;
};
struct node *tail;
void insertend()
{
tail = (struct node *) malloc (sizeof (struct node));
printf (" enter node data ");
scanf (" %d ", temp -> data);
temp -> link = NULL;
if (tail == NULL)
{
    printf (" no node available in the list ");
    tail = temp;
    tail -> link = temp;
}
else
{
    temp -> link = tail -> link;
    tail -> link = temp;
    tail = temp;
}
}
```

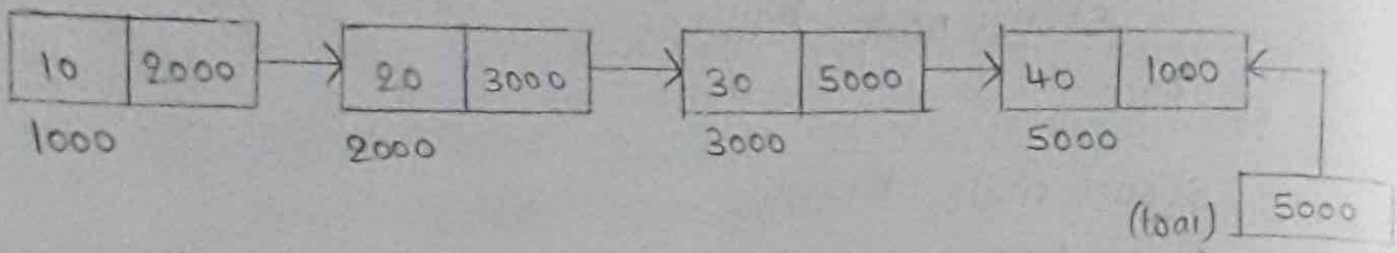
→ Nodes before insertion,



→ Node to be inserted,



→ Resultant nodes after insertion,



Insertion at specified location of the list :-

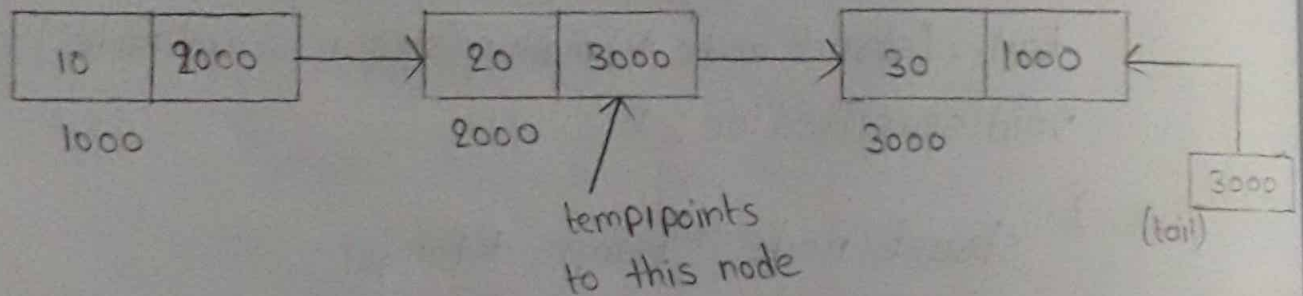
Program :-

```
int length( )
{
  struct node * temp ;
  temp =
struct node
{
  int data;
  struct node *link ;
};
struct node *tail ;
tail = (struct node *) malloc (size of (struct node));
void specified loc ( )
{
  struct node * ~~tail~~ temp;
  temp = (struct node *) malloc (size of (struct node)
  scanf ("%d", temp → data);  → printf ( enter node
                                  data );
```

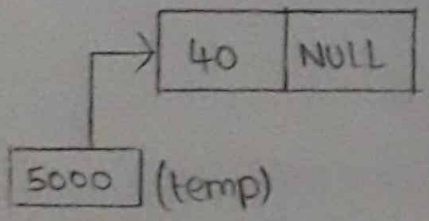
```
temp->link = NULL;
if(tail == NULL)
{
  printf("no node available in list");
  tail = temp;
  tail->link = temp;
}
else
      struct node
      {
        int data;
        struct node *link;
      };
      struct node *tail;
      tail = (struct node *) malloc (sizeof (struct node));
      void specifiedloc()
      {
        struct node *temp, *temp1;
        int loc, i=1, len;
        printf("Enter location");
        scanf("%d", &loc);
        len = length();
        if(loc>len)
        {
          printf("Invalid location");
        }
        else
        {
          temp = (struct node *) malloc (size of (struct node));
```

```
printf("enter node data");
scanf("%d", &temp->data);
temp->link = NULL;
temp1 = tail->link; (for above example it is
                     equal to 1000)
while (i<loc-1)
{
   temp1 = temp1->link;   } for traversing
   i++;                     from starting to
}                           specified location
temp1->link = temp;
temp->link = tail;
}
}
```

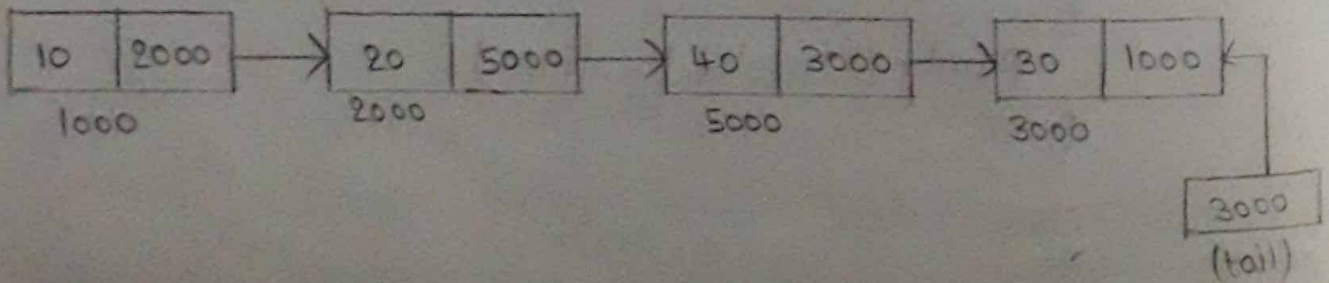
→ Nodes before insertion,



→ Node to be inserted,



→ Resultant nodes,



Deletion operation on single circular linked list:-

→ There are 3 possibilities:-

1. Deleting a node at starting of list.
2. Deleting a node at specified location of list.
3. Deleting a node at end of the list.

Deleting a node at starting of list:-

```
void delete()
{
    struct node *temp;
    int loc, len;
    len = length();
    printf("enter location");
    scanf("%d", &loc);
    if(loc>len)
    {
      printf("Invalid location");
    }
    else
    {
      if(loc==1)
      {
        tail->link = temp->link;
        temp->link = NULL;
        free(temp);
      }
    }
}
```

void Deleting a node at specified location :-

```
void specifiedloc()
{
  struct node *temp, *temp1 = tail->link;
  int loc, len;
  len = length();
  printf("enter location");
  scanf("%d", &loc);
  if(loc>len)
  {
    printf("Invalid location");
  }
  else
  {
    if(loc==1)
    {
      tail->link = temp->link;
      temp->link = NULL;
      free(temp),
    } int i=1;
  }  while(i<loc-1)
}    {
       temp1 = temp1->link;
       i++;
     }
     temp = temp1->link;
     temp1->link = temp->link;
     temp->link = NULL;
     free(temp);
  }
}
```

Deleting a node at end of list :-

```
void endof()
{
  struct node *temp, *temp1 = tail->link;
  int loc, len;
  len = length();
  printf("enter location");
  scanf("%d", &loc);
  if(loc>len)
  {
    printf("Invalid location");
  }
  else
  {
    int i=1;
    while(i<loc-1)
    {
      temp1 = temp1->link;
      i++;
    }
    temp1->link = tail->link;
    tail = temp1;
  }
}
```

To display all the nodes :-

```
void display()
{
  struct node *temp;
  temp = tail->link;
```

```
        if (temp -> link == NULL)
        {
          printf (" No nodes are available");
        }
        else
        {
          while (temp -> link != tail -> link)
          {
            printf (" %d", temp -> data);
            temp = temp -> link;
          }
        }
      }
```

To search for a node :-

```
      void search ( )
      {
        struct node * temp;
        int key, flag = 0;
        temp = tail -> link;
        printf (" Enter key value");
        scanf (" %d", & key);
        while (temp != NULL)
        {
          if (key == temp -> data)
          {
            flag = 1;
            break;
          }
          else
          {
```

```
            temp = temp -> link ;
        }
    }
    if (flag == 1)
    {
        Printf (" search element is available ");
    }
    else
    {
        Printf(" search element is not available ");
    }
}
```

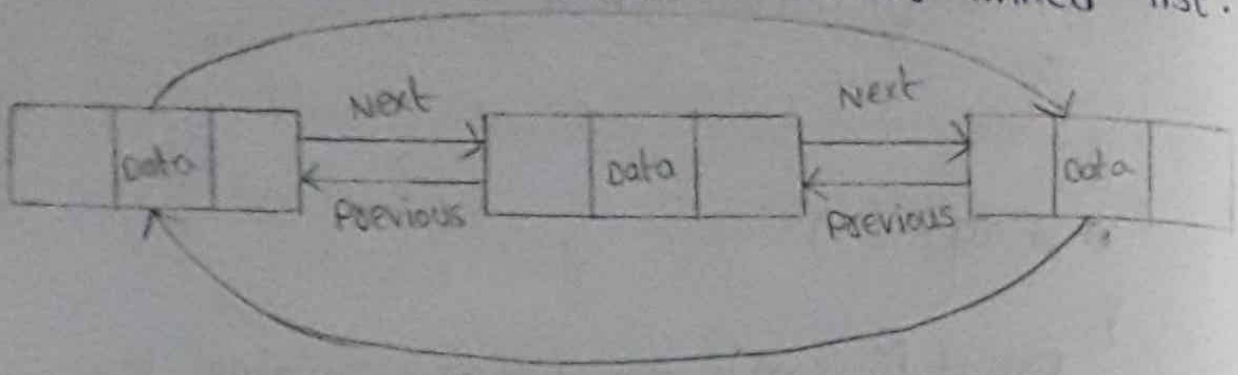
to find length :-

```
int length ( )
{
    int count = 0 ;
    struct node * temp ;
    temp = tail -> link ;
    while (temp != NULL)
    {
        count ++ ;
        temp = temp -> link ;
    }
    return count ;
}
```

Double circular linked list :-

→ The advantage of both double linked list and circular linked list are incorporated by into another

type of list structure called circular linked list.



→ Two consecutive elements are linked or connected by previous and next pointer and the last node points to first node by next pointer and also the first node points to last node by previous pointer.

Polynomial abstract data type :-

→ A polynomial object is a homogeneous ordered list of pairs i.e., exponents and coefficients where, each coefficient is unique.

→ A polynomial P(x) is the expression in variable x which is in the form $(ax^n + bx^{n-1} + \dots + jx + k)$ where a, b, c, ... , k fall in the category of real numbers and 'n' is non negative integer, which is called the degree of polynomial.

Eg :- $10x^2 + 26x$

→ Here 10 and 26 are coefficients and 2, 1 is its exponential values.

operations on polynomial ADT :-

→ creation of a polynomial addition.

→ of two polynomial subtraction of

→ two polynomial multiplication of
→ two polynomial polynomial
→ Evaluation.

Advantages :-

→ Easy to handle.

Disadvantages :-

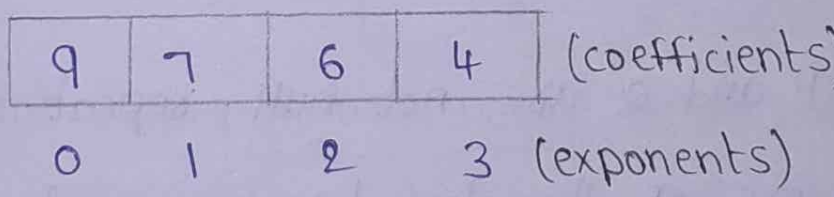
→ It is time consuming.
→ wastage of memory space.
→ we can change the size of an array.

Polynomial Representation :-

→ Polynomial can be represented in the various ways.

By using arrays :-

→ we can use a single dimensional array as shown below



→ we can use 2 dimensional array or array of structures.
→ Here an array of structure is used such that each structure consists of two data members coefficient and exponent.
→ The coefficient and exponent are bound together in a

structure to form one polynomial term and then array of such structures is used to represent a polynomial.

Eg:- consider the following polynomial equation,

$$4x^9 + 3x^7 + 8x^6 + 6x^3 + x^2 + 2x + 5$$

Representation:-

Index	coefficient	Exponent
0	4	9
1	3	7
2	8	6
3	6	3
4	1	2
5	2	1
6	5	0

Algorithm for addition of two polynomials:-

→ Assume that there are two polynomials P and Q. the polynomial result is stored in third array SUM.

Step 1: While P and Q are not null, repeat step 2.

Step 2: if Powers of the two terms are equal then insert the sum of the terms into the SUM Polynomial.

Advance P

Advance Q

Else if the Power of the first polynomial > Power of second Then insert the term from

first polynomial into SUM Polynomial Advance P.
lse insert the term from second polynomial into
SUM Polynomial Advance Q.

tep:-3 copy the remaining terms from the non empty polynomials into the SUM Polynomial.

tep 4: Print SUM

tep 5: Exit

Polynomial representation using linked list:-

→ Drawback in polynomial representation using arrays:

. if the exponent of the polynomial is large then an we need to take carry array of large size.

. if the given polynomial contains only one term with highest degree then storing it in the array may leads to wastage of memory.

Eg:-

arr	9	7	6	4	(coefficients)
	0	1	2	3	(exponents)

Polynomial representation using linked list:-

→ While representing the polynomial using linked list, the node of the Polynomial is as follows:

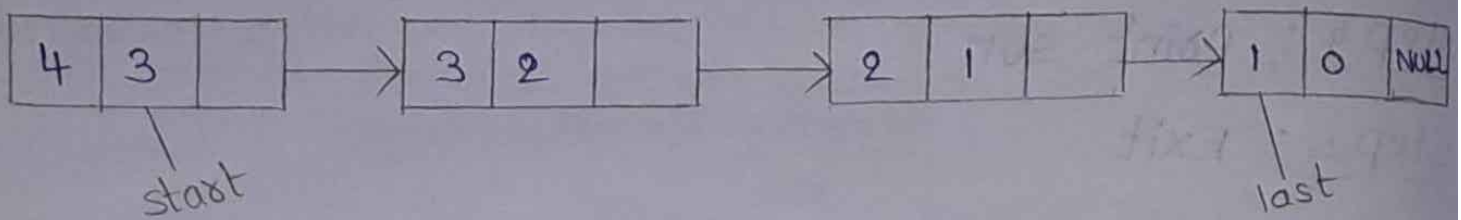
coefficient	Exponent	next

Syntax:-

```
struct poly
{
   int coeff;
```

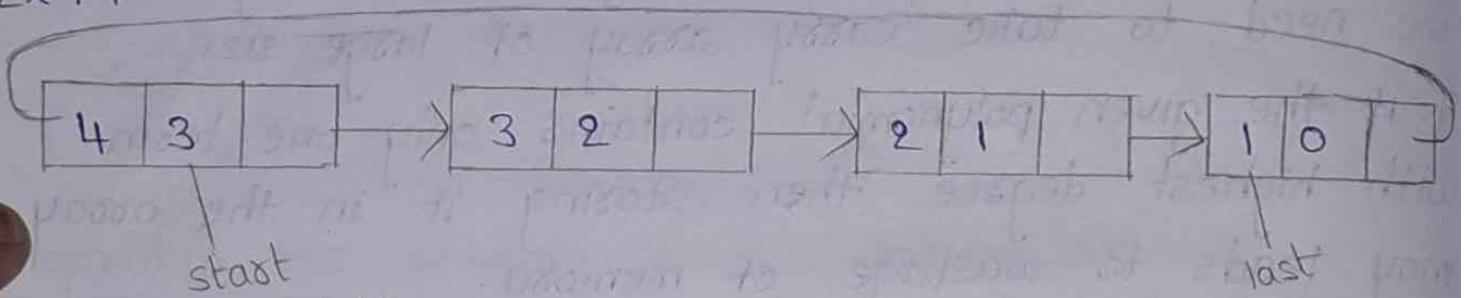
```
    int expo;
    struct poly *next;
};
```

Eg:- Let us consider a polynomial equation $4x^3+3x^2+2x+1$.



Polynomial representation using circular linked list :-

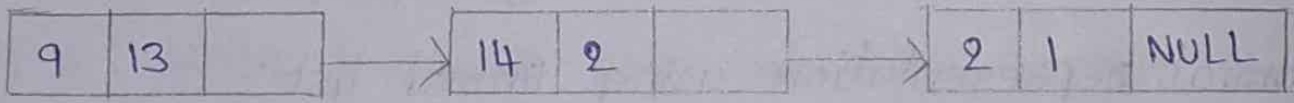
Eg:- Let us consider a polynomial equation $4x^3+3x^2+2x+1$.



Polynomial addition :-

→ consider two polynomials,

$A = 9x^{13}+14x^2+2x$



$B = 5x^{13}+8x+3$

step 1 :- A exponent == B exponent add A coeff + B coeff and store result in c.

step 2 :- A exponent > B exponent so add A to c

step 3 :- A exponent == B exponent add A coeff + B coeff and store result in C.

step 4 :- B != NULL add B to C.

sparse matrix and its representation :-

sparse matrix :-

→ A matrix is a two-dimensional data object made of m rows and n columns, therefore having total mxn values.

→ In many situations, the matrix size is very large but most of the elements of the matrix have 0 value (less important or irrelevant data), only a small fraction of the matrix is actually used, then such type of matrix is called a sparse matrix.

Eg:-

0	0	3	0	4
0	0	5	7	0
0	0	0	0	0
0	2	6	0	0

Dense matrix :-

→ If most of the elements of the matrix has non-zero values, then it is called a dense matrix.

sparse matrix representation :-

→ sparse matrix representation can be done in many ways following are two common representations :-

1. Array representation (triplet representation)

2. Linked list representation.

Using arrays:-

→ 2D array is used to represent a sparse matrix in which there are three ~~rows~~ name columns named as

Row:- Index of row, where non-zero element is located.

column:- Index of column, where non-zero element is located.

value:- value of the non zero element at index-(row, column)

Eg:-

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 9 & 0 \\ 0 & 8 & 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 5 \\ 0 & 0 & 2 & 0 & 0 & 0 \end{bmatrix}$$

→

Rows	columns	values
(5)	(6) No. of columns	(6) No. of values for a row
0	4	9
1	1	8
2	0	4
2	3	2
3	5	5
4	2	2

No. of rows

Sparse matrix representation using linked list:-

→ To represent sparse matrix in the form of linked lists we require nodes and each node should have four fields:-

i Row ii column iii non-zero iv. link field

Representation :-

Row	column	non-zero	*link

→ In this case we should identify at which location non-zero elements are placed.

→ From above example 9 is located in 0th row and 4th column and 8 is located in 1st row and 1st column.

→ creation of nodes is based on no. of non-zero elements. If non zero elements are 6 then we should create 6 nodes. (In above example non-zero elements = 6)

Representation :-