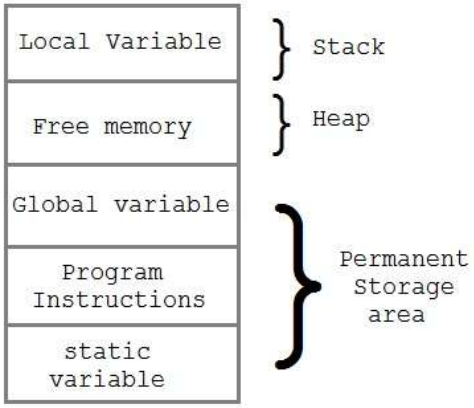
❖ Explain Dynamic Memory Allocation.

- The process of allocating memory at runtime is known as **dynamic memory allocation**. Library routines known as "memory management functions" are used for allocating and freeing memory during execution of a program. These functions are defined in **stdlib.h**.

Function	Description
malloc()	allocates requested size of bytes and returns a void pointer pointing to the first byte of the allocated space
calloc()	allocates space for an array of elements, initialize them to zero and then return a void pointer to the memory
free	releases previously allocated memory
realloc	modify the size of previously allocated space

- **Memory Allocation Process**

Global variables, **static** variables and program instructions get their memory in **permanent** storage area whereas **local** variables are stored in area called **Stack**. The memory space between these two region is known as **Heap** area. This region is used for dynamic memory allocation during execution of the program. The size of heap keep changing.



- **Allocating block of Memory**

1.malloc():- function is used for allocating block of memory at runtime. This function reserves a block of memory of given size and returns a pointer of type void. This means that we can assign it to any type of pointer using typecasting. If it fails to locate enough space it returns a NULL pointer.

Example using malloc() :

```
int *x;
x = (int*)malloc(50 * sizeof(int));    //memory space allocated to variable x
```

```
free(x);                    //releases the memory allocated to variable x
```

2.calloc():- is another memory allocation function that is used for allocating memory at runtime. **calloc** function is normally used for allocating memory to derived data types such as **arrays** and **structures**. If it fails to locate enough space it returns a NULL pointer.

Example using calloc() :

```
struct employee
{
 char *name;
 int salary;
};
typedef struct employee emp;
emp *e1;
e1 = (emp*)calloc(30,sizeof(emp));
```

realloc() :-changes memory size that is already allocated to a variable.

Example using realloc() :

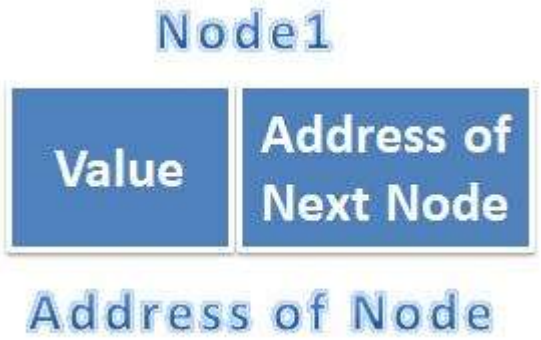
```
int *x;
x=(int*)malloc(50 * sizeof(int));
x=(int*)realloc(x,100); //allocated a new memory to variable x
```

- **Diffrence between malloc() and calloc()**

calloc()	malloc()
➢ calloc() initializes the allocated memory with 0 value.	❖ malloc() initializes the allocated memory with garbage values.
➢ Number of arguments is 2	❖ Number of argument is 1
➢ **Syntax :** (cast_type *)calloc(blocks , size_of_block);	❖ **Syntax :** (cast_type *)malloc(Size_in_bytes);

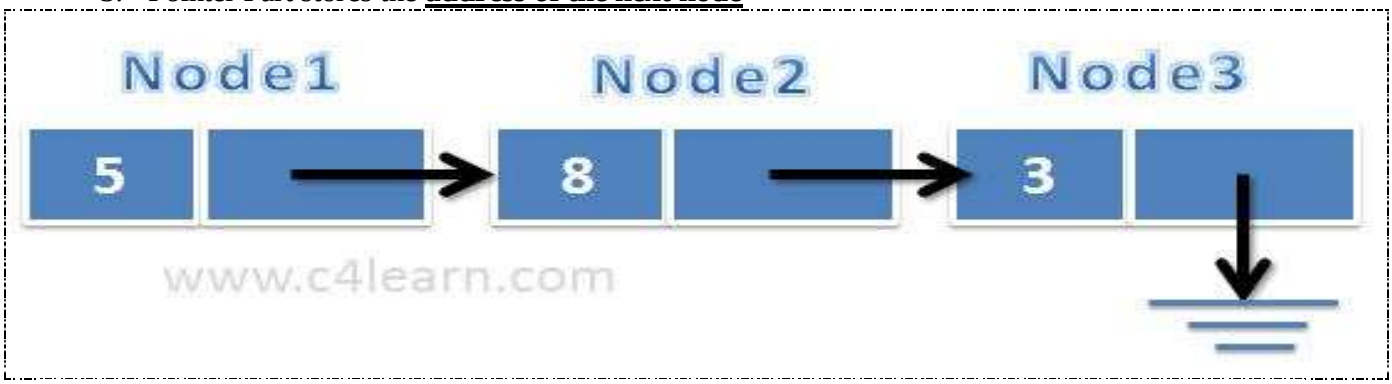
❖ Introduction to Linked List

1. It is a data Structure which consists if group of nodes that forms a sequence.
2. It is very common data structure that is used to create tree,graph and other abstract data types.



Linked list comprise of group or list of nodes in which each node have link to next node to form a chain

- **Linked List definition**
  1. Linked List is **Series of Nodes**
  2. Each node Consist of two Parts viz **Data Part & Pointer Part**
  3. Pointer Part stores the **address of the next node**



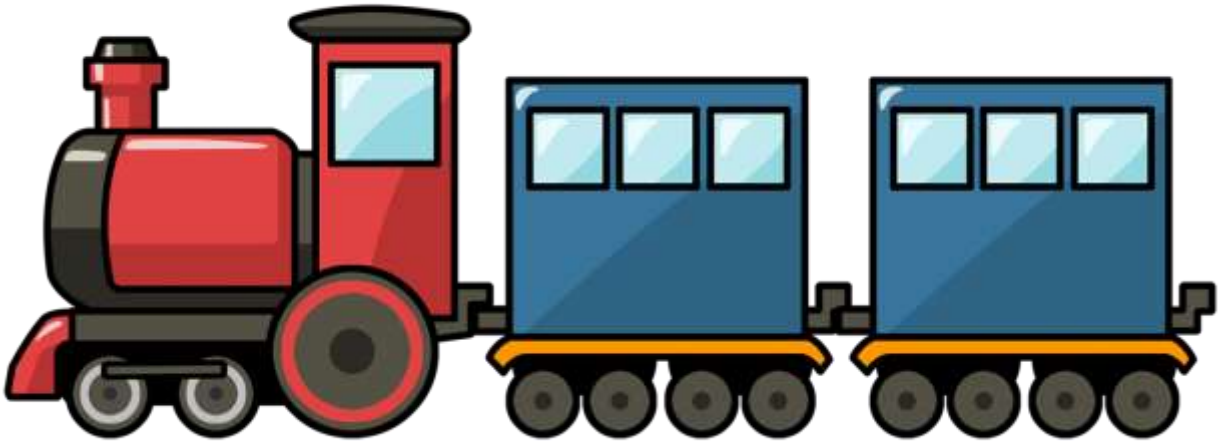
- **What is linked list Node ?**
  1. Each Linked List Consists of Series of Nodes
  2. In above Diagram , **Linked List Consists of three nodes A,B,C etc**
  3. Node A has two part one data part which consists of the 5 as data and the second part which contain the address of the next node (**i.e it contain the address of the next node**)

- **Linked list Blocks**

Linked list is created using following elements –

No	Element	Explanation
1	Node	Linked list is collection of number of nodes
2	Address Field in Node	Address field in node is used to keep address of next node
3	Data Field in Node	Data field in node is used to hold data inside linked list.

We can represent linked list in real life using train in which all the buggies are nodes and two coaches are connected using the connectors.



In case of railway we have peoples seating arrangement inside the coaches is called as data part of lined list while connection between two buggies is address filed of linked list.
Like linked list, trains also have last coach which is not further connected to any of the buggie. Engine can be called as first node of linked list

- **List of advantages :**
  1. Linked List is **Dynamic data Structure** .
  2. Linked List **can grow and shrink during run time**.
  3. **Insertion and Deletion** Operations are Easier
  4. **Efficient Memory Utilization** ,i.e no need to pre-allocate memory
  5. Faster Access time,can be expanded in **constant time without memory overhead**
  6. Linear Data Structures such as Stack,Queue can be **easily implemeted**using Linked list
- **Need of linked list**

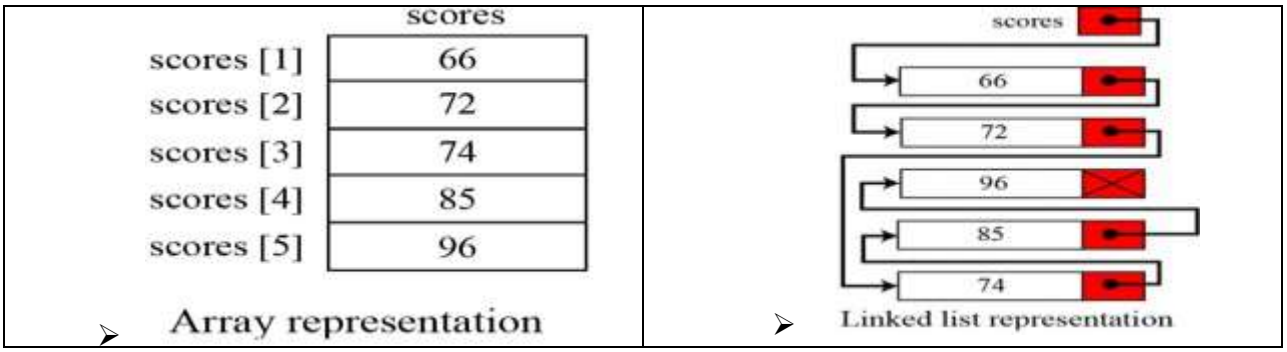
Suppose you are writing a program which will store marks of 100 students in maths. Then our logic would be like this during compile time –

```
int marks[100];
```

Now at run time i.e after executing program if number of students are 101 then how you will store the address of 101th student ?
Or if you need to store only 40 students then again you are wasting memory unnecessarily.
Using linked list you can create memory at run time or free memory at run time so that you will able to fulfil your need in efficient manner

❖ Comparison between Array and Link list

Array	Linked List
➢ Array is a collection of elements having same data type with common name.	➢ Linked list is an ordered collection of elements which are connected by links/pointers.
➢ In array, elements can be accessed using index/subscript value, i.e. elements can be randomly accessed like arr[0], arr[3], etc. So array provides fast and **random access.**	➢ In linked list, elements can't be accessed randomly but can be accessed only **sequentially**and accessing element takes **0(n) time**.
➢ In array, elements are stored in **consecutive** manner in memory.	➢ In linked list, elements can be stored at any available place as address of node is stored in previous node.
➢ Insertion & deletion takes more time in array as elements are stored in consecutive memory locations.	➢ nsertion & deletion are fast & easy in linked list as only value of pointer is needed to change.
➢ In array, memory is allocated at compile time i.e. **Static Memory Allocation**.	➢ In linked list, memory is allocated at run time i.e.**Dynamic Memory Allocation.**
➢ Array can be **single dimensional**, two dimension or**multidimensional**.	➢ Linked list can be **singly**, **doubly** or **circular**linked list.
➢ In array, each element is independent, no connection with previous element or with its location.	➢ In Linked list, location or address of elements is stored in the link part of previous element/node.
➢ In array, no pointers are used like linked list so no need of extra space in memory for pointer.	➢ In linked list, adjacency between the elements are maintained using pointers or links, so pointers are used and for that extra memory space is needed.



❖ Create single Link list

1. In this type of Linked List two **successive nodes** are linked together in**linear fashion** .
2. Each Node contain **address of the next node** to be followed.
3. In Singly Linked List only Linear or Forward **Sequential movement is possible**.
4. Elements are accessed sequentially , **no direct access** is allowed.

- **Explanation :**

1. It is most basic type of Linked List in C.
2. It is simple **sequence of Dynamically allocated Nodes**.
3. Each Node has its successor and predecessor.
4. **First Node does not have predecessor** while **last node does not have any successor**.
5. Last Node have successor reference as "**NULL**".
6. In the above Linked List We have 3 types of nodes.

```
6.1     .First Node
6.2     Last Node
6.3.Intermediate Nodes
```

7. In Singly Linked List access is given only in **one direction thus Accessing Singly Linked is Unidirectional**.
8. We can have multiple data fields inside Node but **we have only single "Link" for next node**.

- **C Program to Create Singly Linked List :**

```
void creat()
{
        char ch;
         do
        {
                struct node *new_node,*current;
                new_node=(struct node *)malloc(sizeof(struct node));
                printf("nEnter the data : ");
               scanf("%d",&new_node->data);
                new_node->next=NULL;
                if(start==NULL)
               {
                        start=new_node;
                         current=new_node;
                }
```

```
                else
                 {
                        current->next=new_node;
                       current=new_node;
                 }
          printf("nDo you want to creat another : ");
          ch=getche();
         }while(ch!='n');
}
```

- **Step 1 : Include Alloc.h Header File**

1. We don't know, how many nodes user is going to create once he execute the program.
2. In this case we are going to allocate memory using Dynamic Memory Allocation functions such as Alloc & Malloc.
3. Dynamic memory allocation functions are included in alloc.h

```
4. #include<alloc.h>
```

- **Step 2 : Define Node Structure**

We are now defining the new global node which can be accessible through any of the function.

```
struct node {
  int data;
  struct node *next;
}*start=NULL;
```

- **Step 3 : Create Node using Dynamic Memory Allocation**

Now we are creating one node dynamically using malloc function.We don't have prior knowledge about number of nodes , so we are calling malloc function to create node at run time.

```
new_node=(struct node *)malloc(sizeof(struct node));
```

- **Step 4 : Fill Information in newly Created Node**

Now we are accepting value from the user using scanf. Accepted Integer value is stored in the data field.

Tips:- Whenever we create new node , Make its Next Field as NULL

```
printf("nEnter the data : ");
scanf("%d",&new_node->data);
new_node->next=NULL;
```

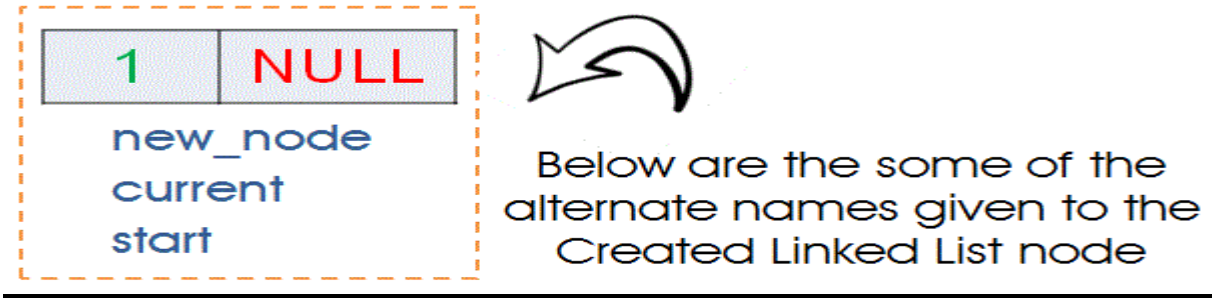
- **Step 5 : Creating Very First Node**

If node created in the above step is very first node then we need to assign it as Starting node. If start is equal to null then we can identify node as first node –

```
start == NULL
```

First node has 3 names : new_node,current,start

```
if(start == NULL) {    {
   start = new_node;
   curr = new_node;
}
```



- **Step 6 : Creating Second or nth node**

1. Lets assume we have 1 node already created i.e we have first node. First node can be referred as "new_node","curr","start".
2. Now we have called create() function again

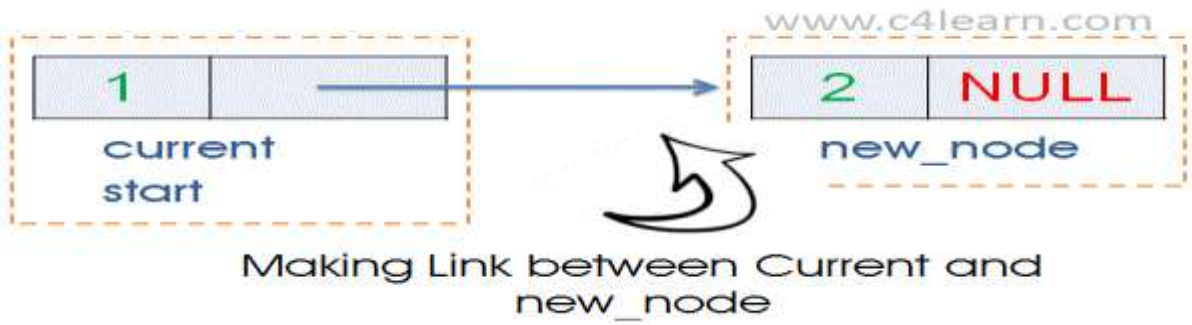
Now we already have starting node so control will be in the else block –

```
else
    {
    current->next = new_node;
    current = new_node;
    }
```

Inside Else following things will happen –

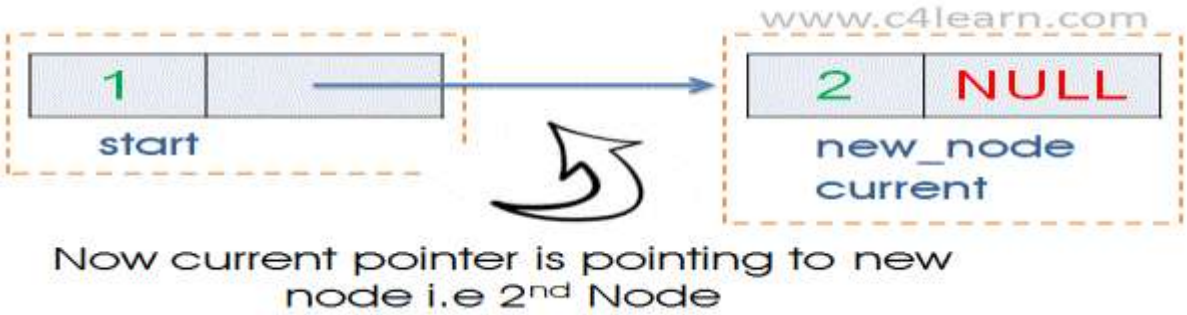
In the else block we are making link between new_node and current node.

```
current->next = new_node;
```



Now move current pointer to next node –

```
current = new_node;
```



❖ Example of Create Singly Linked List .

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
//-----------------------------------------------
struct node
{
        int data;
        struct node *next;
}*start=NULL;
//-------------------------------------------------------------
void creat()
{
        char ch;
         do
        {
                struct node *new_node,*current;
                new_node=(struct node *)malloc(sizeof(struct node));
                printf("nEnter the data : ");
                 scanf("%d",&new_node->data);
                new_node->next=NULL;
```

```
                if(start==NULL)
                {
                        start=new_node;
                         current=new_node;
                 }
                else
                {
                        current->next=new_node;
                         current=new_node;
                 }
                printf("nDo you want to creat another : ");
                ch=getche();
        }while(ch!='n');
}
//-------------------------------------------------------------
void display()
{
        struct node *new_node;
         printf("The Linked List : n");
         new_node=start;
         while(new_node!=NULL)
         {
                 printf("%d--->",new_node->data);
                  new_node=new_node->next;
          }
         printf("NULL");
}
//-----------------------------------------------
void main()
{
        create();
        display();
}
//-----------------------------------------------
```

Output :

```
Enter the data : 10
Do you want to creat another :  y
Enter the data : 20
Do you want to creat another : y

Enter the data : 30
Do you want to creat another : n

The Linked List :
10--->20--->30--->NULL
```

❖ Summary of Different Linked List Terms :

Declaration Term	Explanation
`struct node *new_node,*current;`	Declaring Variable of Type Node.
`*start = NULL;`	Declared a global variable of type node. “start” is used to refer the starting node of the linked list.
`start->next`	Access the 2nd Node of the linked list. This term contain the address of 2nd node in the linked list. If 2nd node is not present then it will return NULL.
`start->data`	It will access “data” field of starting node.
`start->next->data`	It will access the “data” field of 2nd node.
`current = start->next`	Variable “current” will contain the address of 2nd node of the linked list.
`start = start->next;`	After the execution of this statement, 2nd node of the linked list will be called as “starting node” of the linked list.

❖ Various operation using link list

1. Insert node at Start/First Position in Singly Linked List

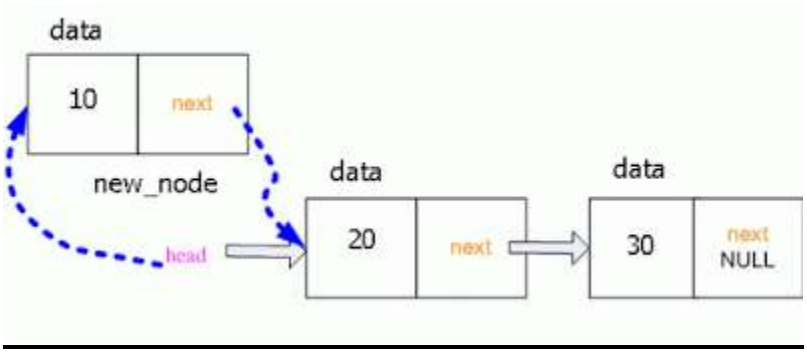
Inserting node at start in the SLL (Steps):

1. **Create** New Node
2. Fill Data into “**Data Field**“
3. Make it’s “**Pointer**” or “**Next Field**” as **NULL**
4. Attach This newly **Created node to Start**
5. Make newnode as **Starting node**

```
void insert_at_beg()
{
        struct node *new_node,*current;
        new_node=(struct node *)malloc(sizeof(struct node));

        if(new_node == NULL)
                printf("nFailed to Allocate Memory");

        printf("nEnter the data : ");
        scanf("%d",&new_node->data);
        new_node->next=NULL;

        if(start==NULL)
         {
```

```
                start=new_node;
                current=new_node;
        }
        else
        {
                new_node->next=start;
                start=new_node;
        }
}
```

Diagram :



Attention :

1. If starting node is not available then **"Start = NULL"** then following part is executed

```
if(start==NULL)
{
       start=new_node;
       current=new_node;
}
```

2. If we have previously created First or starting node then **"else part"** will be executed to insert node at start

```
else
{
       new_node->next=start;
       start=new_node;
 }
```

2. insert node at Last / End Position in Singly Linked List

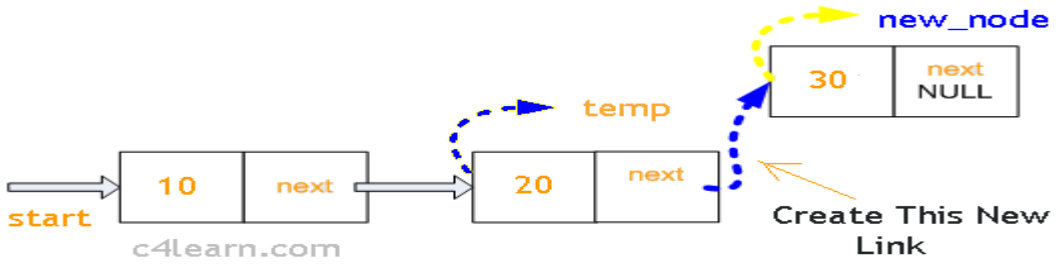
Inserting node at start in the SLL (Steps):

1. **Create** New Node
2. Fill Data into "**Data Field**"
3. Make it's "**Pointer**" or "**Next Field**" as **NULL**

4. Node is to be inserted at Last Position so we need to traverse **SLL upto Last Node**.
5. Make link between **last node and newnode**

```
void insert_at_end()
{
        struct node *new_node,*current;
        new_node=(struct node *)malloc(sizeof(struct node));
        if(new_node == NULL)
                printf("nFailed to Allocate Memory");
        printf("nEnter the data : ");
        scanf("%d",&new_node->data);
        new_node->next=NULL;
        if(start==NULL)
        {
                start=new_node;
                current=new_node;
        }
        else
        {
                temp = start;
                 while(temp->next!=NULL)
                {
                        temp = temp->next;
                }
                temp->next = new_node;
        }
}
```

Diagram :



Attention :

1. If starting node is not available then **"Start = NULL"** then following part is executed

```
if(start==NULL)
       {
       start=new_node;
       current=new_node;
       }
```

2. If we have previously created First or starting node then **"else part"** will be executed to insert node at start
3. Traverse Upto Last Node., So that **temp** can keep track of Last node

```
else
       {
       temp = start;
   while(temp->next!=NULL)
  {
  temp = temp->next;
  }
```

4. Make **Link between Newly Created node and Last node** (temp)

```
temp->next = new_node;
```

To pass Node Variable to Function Write it as –

```
void insert_at_end(struct node *temp)
```

3. Linked-List : Insert Node at Middle Position in Singly Linked List

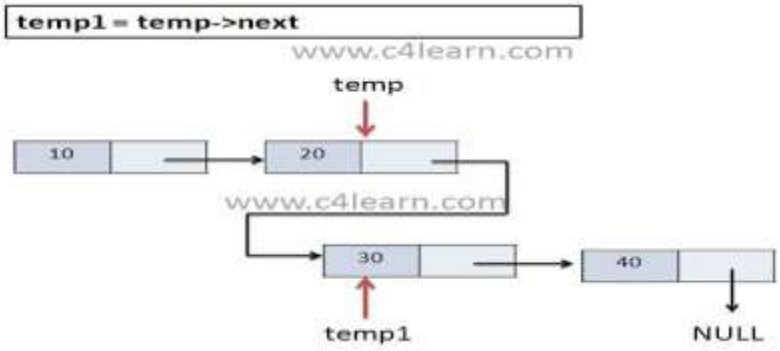
```
void insert_mid()
{
    int pos,i;
    struct node *new_node,*current,*temp,*temp1;

    new_node=(struct node *)malloc(sizeof(struct node));

    printf("nEnter the data : ");
    scanf("%d",&new_node->data);

    new_node->next=NULL;
    st :
    printf("nEnter the position : ");
    scanf("%d",&pos);

    if(pos>=(length()+1))
       {
       printf("nError : pos > length ");
       goto st;
       }
    if(start==NULL)
       {
       start=new_node;
       current=new_node;
       }
    else
       {
       temp = start;
             for(i=1;i< pos-1;i++)
```

```
                {
                temp = temp->next;
                }
        temp1=temp->next;
        temp->next = new_node;
        new_node->next=temp1;
        }
}
```

Explanation :

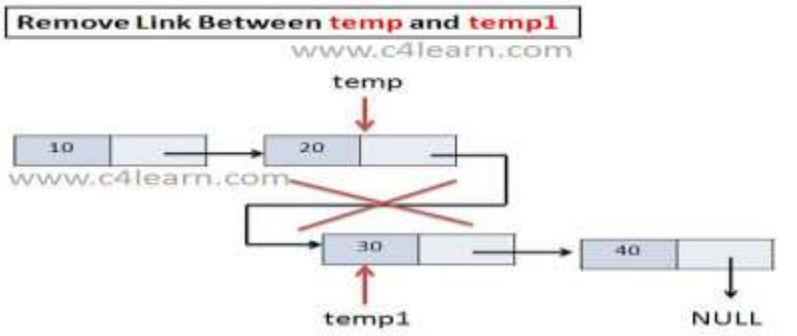
Step 1 : Get Current Position Of "temp" and "temp1" Pointer.

```
temp = start;
                for(i=1;i< pos-1;i++)
  {
  temp = temp->next;
  }
```



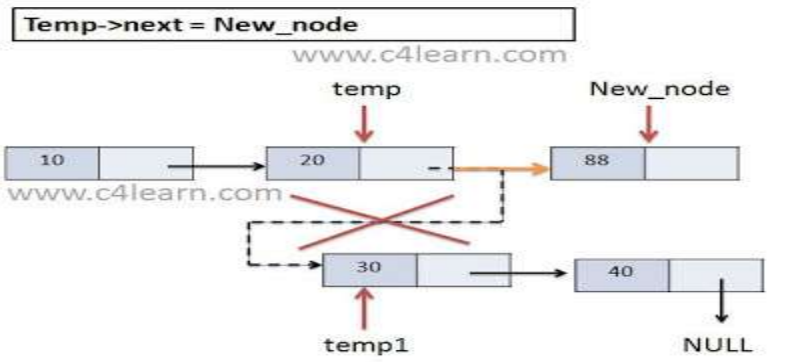
Step 2 :

```
temp1=temp->next;
```



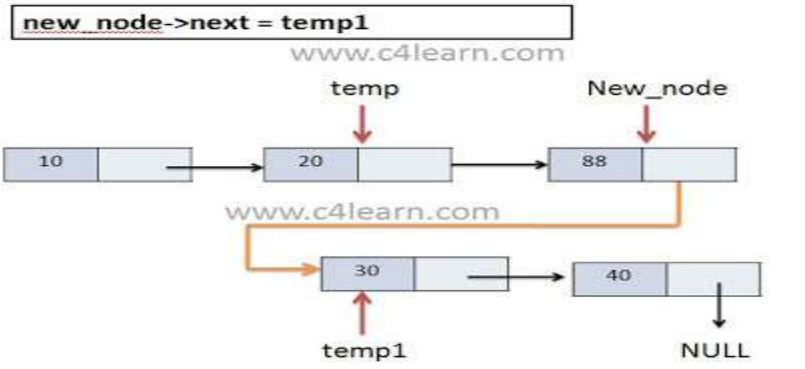
Step 3 :

```
temp->next = new_node;
```



Step 4 :

```
new_node->next = temp1
```



Delete First Node from Singly Linked List

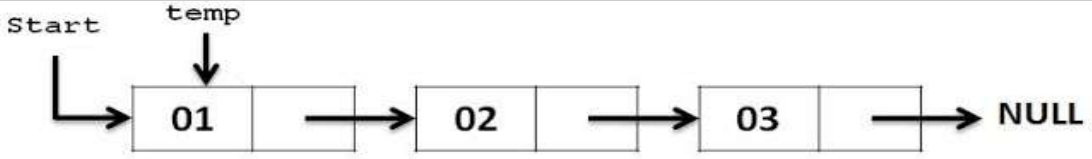
Program :

```
void del_beg()
{
        struct node *temp;
        temp = start;
        start = start->next;
        free(temp);
        printf("nThe Element deleted Successfully ");
}
```

Attention :

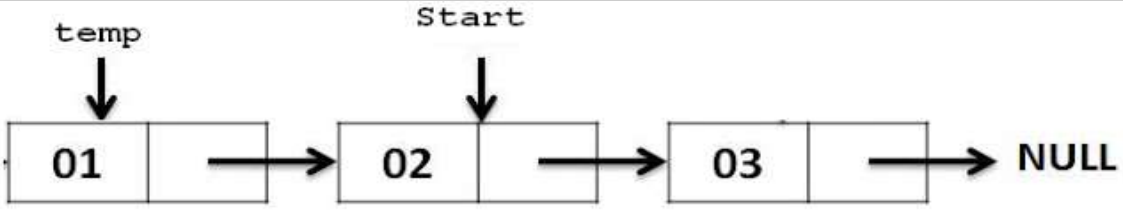
Step 1 : Store Current Start in Another Temporary Pointer

```
temp = start;
```



Step 2 : Move Start Pointer **One position Ahead**

```
start = start->next;
```



Step 3 : Delete temp i.e Previous Starting Node as we have Updated Version of Start Pointer

```
free(temp);
```