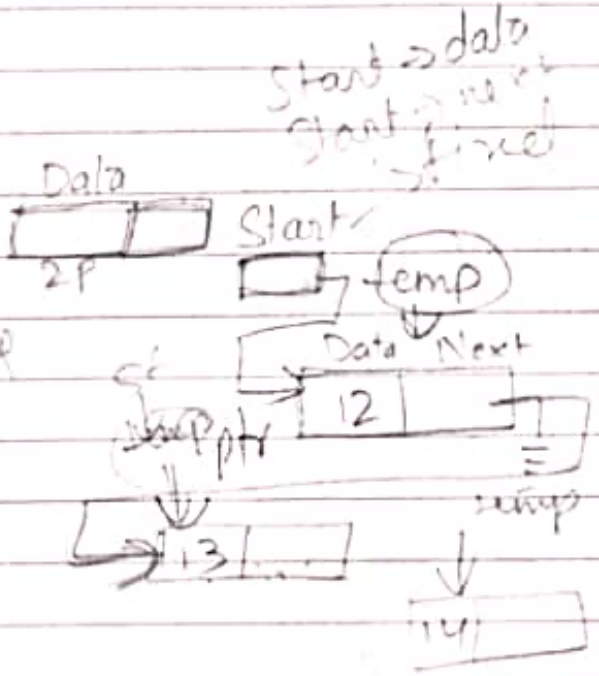
Link list

→ An element in the list is called a node. The node is a self-referential structure, having two parts: Data and Next.

→ The Next of last node points to NULL.

Declaration of Node:-




```
typedef struct nodetype
{
 int data;
 struct nodetype * next;
} node;
node * start = NULL;
 node * ptr , * temp;
```

Create N nodes

```
void create ()
{
 int ch;
printf (" Enter the choice\n");
 scanf ("%d", &ch);
 while (ch != 1)
 {
  temp= (node *) malloc (sizeof (node));
  printf (" Enter the info\n");
  scanf ("%d", &temp->data);
  temp->next = NULL;

 if ( start == NULL)
 {
```

```
    start = temp;
    ptr = temp;
  }
  else
   {
    ptr -> next = temp;
     ptr = ptr -> next;
   }

  printf("Press two to continue\n");
   scanf("%d", &ch);
  }
 }
```

// Display

```
 void display()
 {
  node *ptr;
  ptr = start;
  while( ptr != NULL)
  {
  printf("%d", ptr -> info);
   ptr = ptr -> next;
  }
}
```

Write an algo to find the no. of nodes :-

```
void count ()
{
 node *ptr = start;
  int c = 0;
 while( ptr != NULL)
  {
     c++;
    ptr = ptr -> next;
  }
 printf("\n %d is the total no. of nodes", c);
}
```

Write a C function to search an element in the list and returns its location.

```
void search ()
{
 int key, c = 0,
 printf(" Enter the key element to be searched\n");
  scanf("%d", &key);
  ptr = start;
 while( ptr != NULL)
 {
  if(ptr -> data == key)
  {
  printf(" Element found at location %d and address is %p", c, ptr);
    break;
  }
  ptr = ptr -> next;
    c++;
}
```

Insertion at the beginning

```
void insbeg( )
{
  temp=(node *)malloc(sizeof(node));
  printf("Enter the element \n");
  scanf("%d", &temp->data);
    temp->next=start;
    start = temp;
}
```

Algorithm

Insert(INFO, NEXT, START, ELE)
{

1. Create a node and return its address to temp.
2. ~~INFO~~ DATA[Temp]= Ele
3. NEXT[Temp] = START
4. START=TEMP
5. Exit

Insertion at the end of the list

```
void insend()
{
 temp=(node *)malloc(sizeof(node));
 printf("Enter the element \n");
  scanf("%d", &temp->data);
  temp -> next= NULL;
   ptr = start.
 while(ptr->next != NULL)
  {
```

```
    ptr = ptr->next;
  }
  ptr->next = temp;
}
```

Algorithm

InsertEnd(DATA, NEXT, START, ELE)

1. Create a node and return its address to temp.
2. DATA[Temp] = Ele
3. NEXT[TEMP] = NULL
4. Set Ptr = Start
5. While(NEXT[Ptr] != NULL)
6. {
   Ptr = NEXT[Ptr]
   }
7. NEXT[Ptr] = Temp.
8. Exit

Insertion at any position after a particular node.

```
void insertany()
{ int key;
  printf("\n Enter data of the node after which new node is to
          be inserted: ");
  scanf("%d ", &key);
  temp = (node *) malloc(sizeof(node));
  printf("\n Enter the element\n");
  scanf("%d", &temp->data);
// Traverse till key is found or end of the link list is reached.
```

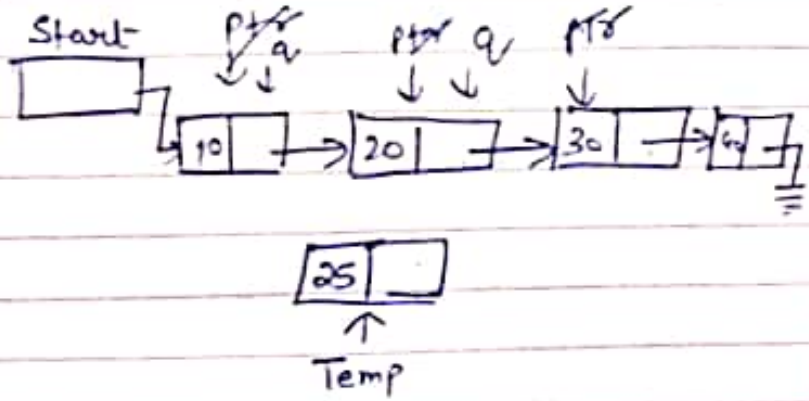
```
ptr = start;
while( ptr -> next != NULL && ptr -> data != key)
 {
    ptr = ptr -> next;
 }
 if( ptr -> data == key)
 {
  temp -> next = ptr -> next;
   ptr -> next = temp;
 }
else
 {
 printf("\n Value %d not found\n", key);
 }
}
```

Insertion at any position before a particular node

```
void insertbefore()
{
 int key; struct node *q;
printf("\n Enter data of the node before which new
        node is to be inserted: ");
scanf("%d", &key);
temp = (node *) malloc(sizeof(node));
printf("\n Enter the element\n");
 scanf("%d", &temp -> data);

  ptr = start;
  while (ptr -> next != NULL && ptr -> data != key)
  {
     q = ptr;
```

```
    ptr = ptr -> next;
  }
  if (ptr->data == key)
  {
    temp->next = ptr;
    q->next = temp;
  }
  else
  {
  printf("\n Value %d not found \n", key);
  }
}
```

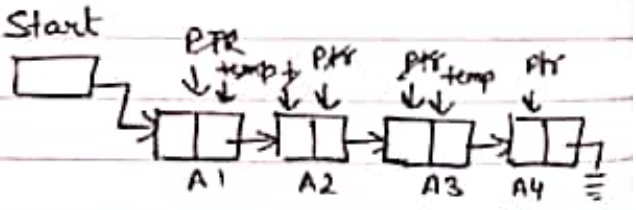


Deletion of first element in Link list

```
void delete_front()
{
  if (start == NULL)
  {
    printf("\n Empty Linked list. Deletion not possible. \n");
  }
  else
  {
    ptr = start;
    start = start -> next;
    free(ptr);
  }
}
```

Deletion of last element in the list

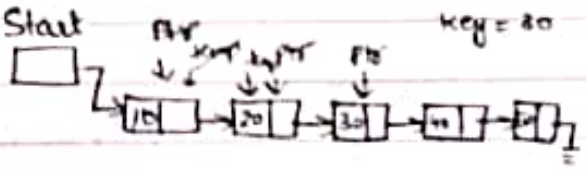
```
void delend()
{
 if(start==NULL)
 {
  printf("\n Underflow");
   return;
 }
 if(start->next==NULL) //List has single node
   {
      ptr = start;
      start = NULL;
      free(ptr);
   }
  else
   {
    ptr=start;
    while(ptr->next!=NULL)
     {
      temp=ptr;
      ptr=ptr->next;
     }
    temp->next=NULL;
     free(ptr);
  }
}
```



Deletion of any node

```
void delete_any()
{
  int key;
  if(start==NULL)
  {
   printf("\n Empty link list. Deletion not possible.\n");
    return;
  }
 else
 {
  printf("\n Enter the data of the node to be deleted");
  scanf("%d",&key);

    ptr = start;
    while( ptr->next!=NULL && ptr->data!=key)
    {
       temp = ptr;
       ptr = ptr->next;
    }
    if(ptr->data==key)
    {
     temp->next = ptr->next;
      free(ptr);
    }
   else
    {
     Printf("\n Value %d not found. Deletion not possible.\n",key);
  }
 }
}
```



Reverse of Singly link list

```
void reverse ()
{
    node *p, *q, *r;
    p = start;
    q = NULL;
    r = NULL;
    while (p != NULL)
    {
        r = q;
        q = p;
        p = p->next;
        q->next = r;
    }
    start = q;
}
```