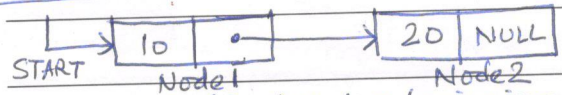
UNIT-3 Linked list: SLL: Representation in memory, operations (Traversing, Searching, insertion, deletion from LL); LL representation of Stack & Queue, DLL

A LL is a non-sequential representation of data items. For every data items in the linked list, there is an associated pointer that would give the address of next data item in the memory. Pointer is a necessary concept to implement linked lists. In LL the data items are not in a consequitive memory locations.

Example to show linked list:-




where START indicates the beginning of LL and NULL indicates the end of LL.

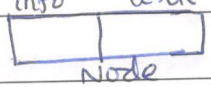
A LL is a collection of several nodes and all nodes are linked together. A node is a combination of two fields:

1. Data field
2. link field.

Data field:- It stores the actual data in a node.

Link field:- It holds the address of next node in the memory. This link field is implemented by pointers.

i.e




The structure of a node in a linked list.

Advantages of LL:-

1) LL are dynamic in nature:- LL size can grow and shrink during the program execution
2) Efficient memory utilization:- LL performs dynamic memory location. No memory gets wasted.
3) Insertion / deletion operations are easier: Performing insertion/deletion of data items is easy when compared to other data structures.
4) Can provide easier solutions to complex problems.

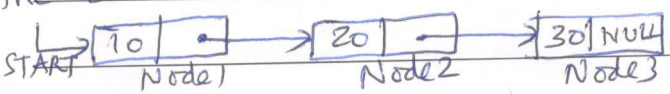
Disadvantages:- 1) Each node structure requires link field that stores the address of next node. So more memory is needed
2) Accessing a specified element is time consuming.

Structure of node:-

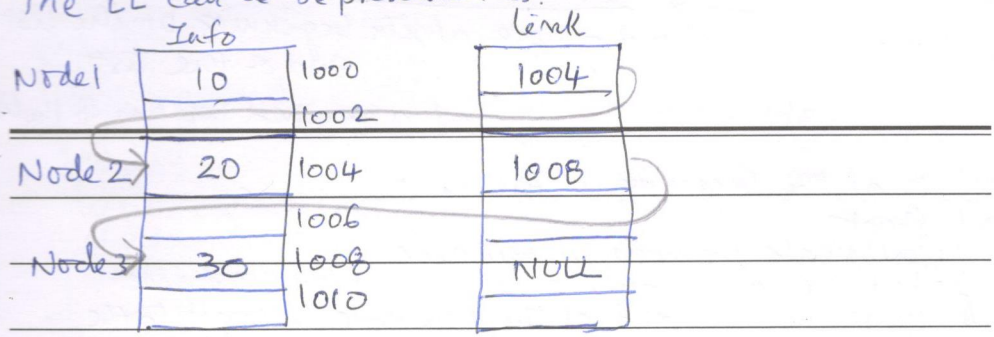
```
struct node
{
  int info;
  struct node *link;
};
typedef struct node NODE;   } To make ADT
```

where the 'info' field can store any data item and link field stores the address of next node in the linked list

Ex:- The LL with three nodes



The LL can be represented as:




Types of linked lists:-

1) Singly LL
2) Doubly LL
3) Circular LL

1. Singly linked list:- In this case nodes are linked together in sequential manner. This is also called as 'linear LL' because the nodes are connected in one direction only. Through the SLL it is not possible to access the previous node from the present node.

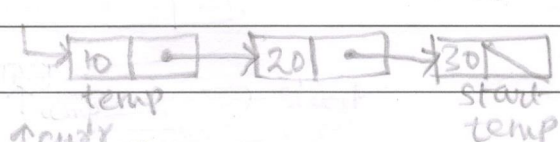
Basic Operations in SLL:-

1. Create 2. Traversing 3. Insert 4. Delete 5. Searching
6. Merging 7. Display.

// Program to create a SLL and display the elements

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
#include<stdlib.h>
void main()
{
struct node
{ int info;
  struct node *link;
};
typedef struct node NODE;
NODE *start, *curr, *temp;
int count=0, choice=1;
curr=0;
while(choice)
{
start=(NODE*)malloc(sizeof(NODE));
printf("\n Enter an element");
scanf("%d", &start->info);
if(curr!=0)
{
  temp->link=start;
  temp=start;
}
else
{
  curr=temp=start;
}
printf("\n Enter 1 for continue and 0 for terminate");
scanf("%d", &choice);
}
temp->link=NULL;
temp=curr;
printf("\n The contents of LL are:::");
while(temp!=NULL)
{
  printf("%d->", temp->info);
  count++;
  temp=temp->link;
}
printf("NULL");
printf("\n Total number of nodes in a singly linked list=%d", count);
getch();
}
```

count = 0 1 2 3
choice = 1 0
curr = 0 start

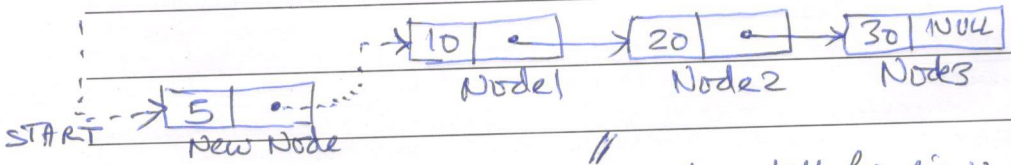


## Insertion of nodes in Single linked list

Three instances: 1) Insertion of a node at the beginning of the list
2) " " end of the list
3) " at a specified position of the list.

1. Insertion at the beginning of the list:-

Step 1: Start
2: allocate memory for new node
3. Put the value to the 'Info' field of the new node
4. Make the link field of the new node to point to the starting node of list
5. Assign the start to the new node
6. Stop




Code for the above algorithm

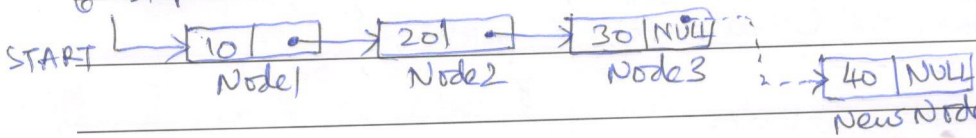
```
begin_insert(NODE *start)
{
NODE *newnode;
int data;
newnode=create_node();
newnode->info = item;
newnode->link=start;
start =newnode;
}
```

create_node() function::

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

2. Insertion of a node at the end of the list:-

Step1: Start
2: If the linked list is empty then create a newnode and ~~make it as a start node~~
3: If the linked list is not empty then goto the last node
4: Create a newnode, insert data into 'info' field of newnode then insert newnode at the end of Linked list
5. Make the link field of last node to NULL
6. Stop



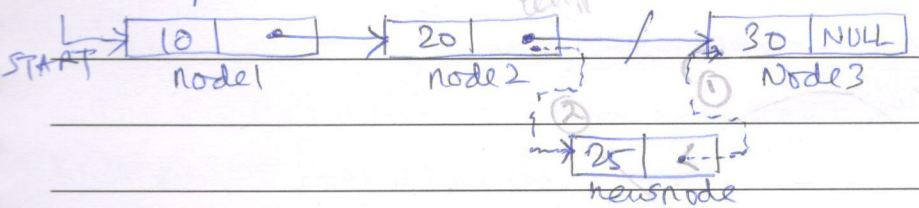
code for insert of a node at the end of LL

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

Insert of a node at the specified position in LL:

Step 1: start
2. Create a newnode
3. Load data into the info field of newnode
4. Find the position to insert a node
5. Insert a newnode at the specified position
6. stop

Specified position: 3
i.e between node2 and node3




```
position_insert(NODE *start, int count)
{
 NODE *newnode, *temp;
 int data, i;
 for(i=0, temp=start; i<count; i++)
 {
  temp=temp->link;
  if(temp==NULL)
  {
   printf("In less nodes in the list");
  }
 }
 newnode=create_node();
 newnode->info=data;
 newnode->link=temp->link;
 temp->link=newnode;
```

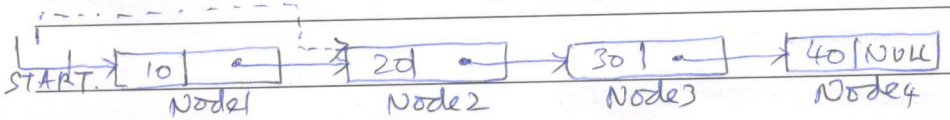
## Deletion of nodes in Single linked lists :-

There instances 1) Deletion of first node
2) Deletion of node at the end of list
3) Deletion of a node at a specified location.

---

### 1. Deletion of a node at the beginning of LL :-

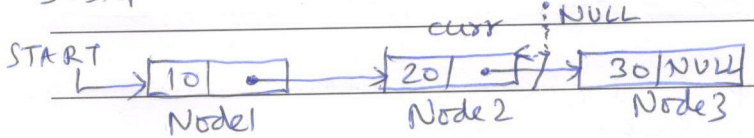
Step1: Start
2: If LL is empty then print " list is empty, Deletion is not Possible".
otherwise goto step3
3. Move the 'start' to the second node in LL
4. Delete the first node by using free() function
5. STOP




```
Begin delete(NODE *start)
{
  NODE *aux;
  int data;
  if(start==NULL)
  {printf("\n The LL is empty");
  }else
   { aux=start;
     start=start->link;
     free(aux);
   }
}
```

### 2. Deletion of last node in LL :-

Step1: Start
2: If LL is empty then print " LL is empty"
else goto Step3
3. Go to the last but one node in LL then set link field of last but one node to NULL
4. Delete the last node by using free() function
5. Stop.

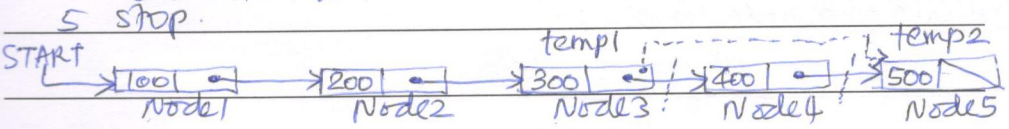



```
end_delete(NODE *start)
{ NODE *cur, *temp;
  if(start==NULL)
  { printf("\n List is empty. Deletion isn't Possible");
  }
  else
```

```
{
  temp=start;
  while(temp->link!=NULL)
  {
    cusr=temp;
    temp=temp->link;
  }
  cusr->link=NULL;
  free(temp);
}
```

3. Deletion of the node from the specified location:-

Step1: Start
2: If the LL is empty print "Deletion is not possible" otherwise goto Step3
3: Consider the position and take two variables. Bring the target node to delete between these two variables.
4. The link field of the left side variable should be connected to other variable
5 Stop.




consider node-4 to be deleted

```
position_delete(NODE *start)
{
  NODE *temp1, *temp2;
  int data;
  printf("\n Enter the element number to be deleted");
  scanf("%d", &data);
  if(start==NULL)
  {
    printf("\n Linked list is empty");
  }
  else
  {
    temp1=start;
    temp2=temp1;
    while(temp1!=NULL)
    {
      if(temp1->info==data)
      {
        temp2->link=temp1->link;
        free(temp1);
      }
      temp2=temp1;
      temp1=temp1->link;
    }
  }
}
```

Advantages of SLL:-

1. Processing of a node in forward direction is easy.
2. Insertion and deletion operations are easy.
3. Supports dynamic data structure.
4. Implementation is easy.

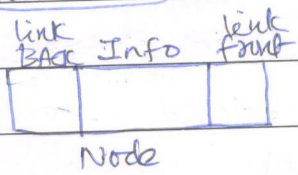
Disadvantages:

1. Memory usage
2. We can't access previous node from the current node.
3. Time consuming to access a node in the list.

Double-linked list:- (DLL)

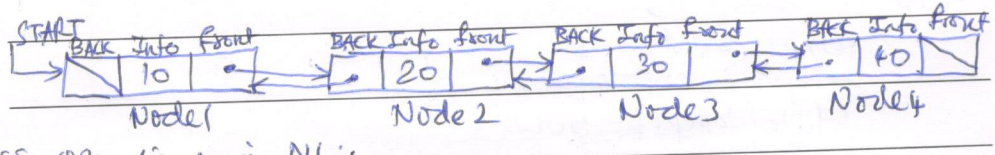
It is a list in which all nodes are linked together by two directions. It supports bidirectional traversing. Each node in DLL has two link fields.

Structure of a node in DLL:-




```
struct node
{
  int info;
  struct node *BACK;
  struct node *front;
};typedef struct node NODE;
```

Representation of a Double-linked list:-



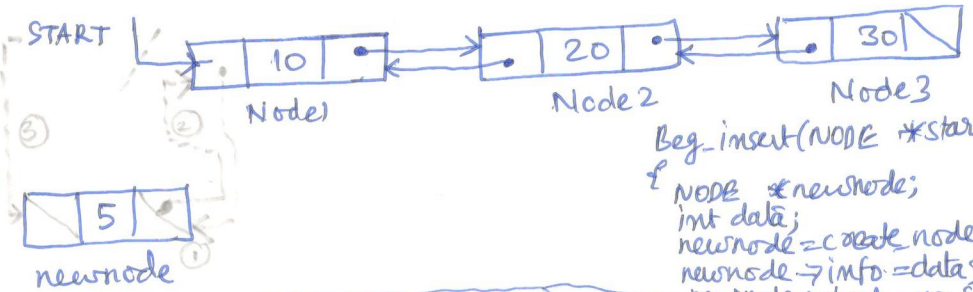

Basic operations in DLL:

1) create()
2) Traversing()
3) Insertion()
4) Deletion()
5) Searching()
6) Merging()
7) Display()

## Insertion of a node at the beginning in DLL :—

Algorithm :-

Step 1 : Start
2 : Create a newnode and assign data to 'info' field of newnode
3. Make 'BACK' and 'front' fields to NULL
4. Make the 'front' link of the newnode to point to the startnode of the list and make 'back' link of the startnode to point to the newnode
5. Replace the 'start' to the newnode
6. Stop.

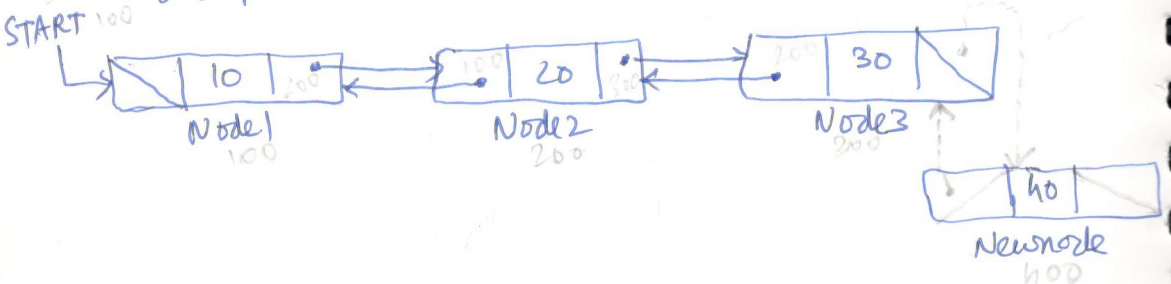



```
Beg_insert(NODE *start)
{
NODE *newnode;
int data;
newnode=create_node();
newnode->info=data;
newnode->back=newnode->front=NULL;
if (start!=NULL)
{
newnode->front=start;
start->back=newnode;
}
start=newnode;
}
```

## Inserting a node at the end of list :-

Algorithm :-

1. Start
2. Create a newnode
3. Store the data into 'info' field of newnode
4. Set the 'back' and 'front' fields of newnode to NULL
5. If the LL is not empty then traverse the list till the lastnode and make the 'front' fields of the lastnode to point to the newnode and 'back' field of the newnode to point to the lastnode
6. Stop



```
end_insert(NODE *start)
{
NODE *newnode, *temp;
int data;
newnode = create_node();
newnode->info = data;
newnode->back = newnode->front = NULL
if (start == NULL)
{ start = newnode;
}
else
{ temp = start;
  while (temp->front != NULL)
  { temp = temp->front;
  }
  temp->front = newnode;
  newnode->back = temp;
}
```

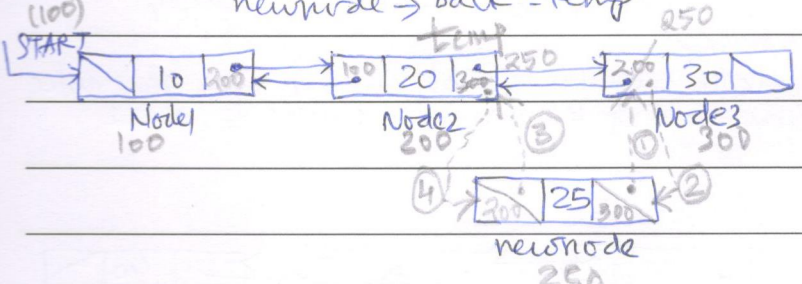
Inserting a node at a specified position in DLL :-

Algorithm

Step1: Start read position and get the count (no of nodes in list)

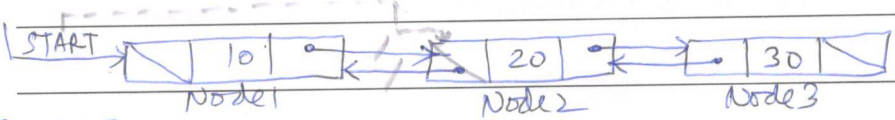
2. If (position < 1) || (pos >= count+1)
   print "Position is out of range to insert"
   else
   if (start == NULL) && (pos != 1)
   print "Empty list"
   else if (start == NULL) && (pos == 1)
   then
   1. create newnode and set 'back' and 'front' to NULL
   2. store data into 'info' field of newnode.
   3. Set newnode as start.
   else goto step3 set i=1 and temp = start
3. Traverse in the list until (i < position - 1)
   using temp = temp -> front
4. newnode -> front = temp -> front
   temp -> front -> back = newnode;
   temp -> front = newnode;
   newnode -> back = temp

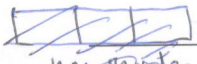
last = newnode.



Delete a node at the beginning in DLL :-

Algorithm: Step:1: Start
2: If the list is empty display "Deletion is not Possible".
else goto Step3
3. If (start == end) then free the startnode and then make start = end = NULL
4. Make the auxiliary pointer to point to the start node and make the start to point to the next node in the list. Then set the
3. If the list is not empty then we have to transfer the start to the next node and then set back of the second node to NULL.
4 free the first node after the transformation of start
5. Stop.

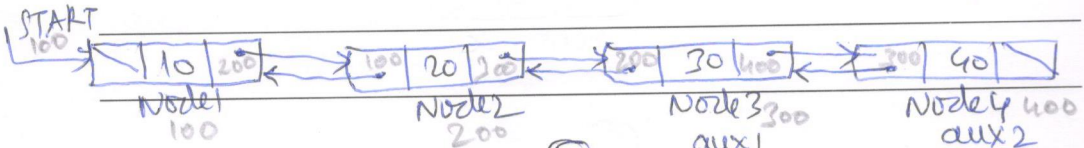



```
begin_delete (NODE *start)
{
 if(start == NULL)
    printf("In list is empty");
 else
   {
    aux = start;
    start = start -> front;
    if(start != NULL)
      { start -> back = NULL;
      }
    free(aux);
   }
}
```

Delete a node at the end of DLL:

Alg: Step1: Start
2: If the list is empty then print "list is empty"
else goto Step3
3. Traverse the list till the last but one node and make the 'front' field of the last but one node to NULL
4 Stop.

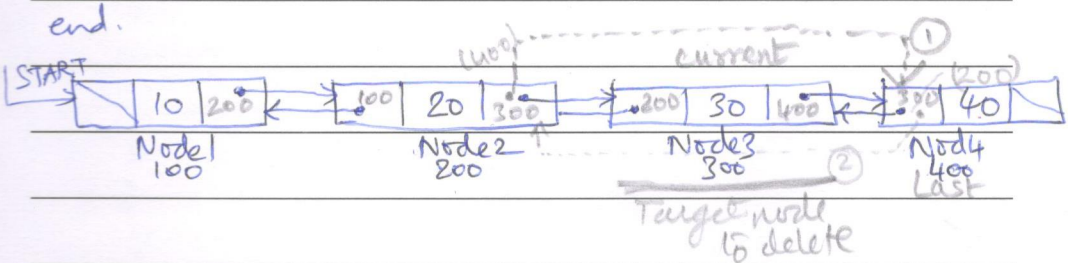


Input: last { Pointer to the lastnode of DLL }

```
Begin
    if (last == NULL) then
        print ("Empty List")
    end if
    else then
        aux = last;
        last = last -> back
        last -> front = NULL
        free (aux);
```

Delete a node in DLL from a specified position:-

```
Input: start, last, position
    current = start
    for i = 1 to position and current != NULL
        current = current -> front
    end for
    If (position == 1) then
        begin_delete()
    end if
    else if (current == last) then
        end_delete()
    end if
    else if (current != NULL) then
        current -> back -> front = current -> front
        if (current -> front != NULL) then
            current -> front -> back = current -> back;
        end if
        free (current)
    end if
    else
        print ("invalid position");
end if
end.
```



# Searching for an element in SLL

Algorithm :-

Step 1: Start
2. Initialize the Start pointer and current = Start.
3. Do the following while current is not NULL
   (a) current->info == key
       return true
   (b) current = current->front
4. Return false
5. Stop.

code :-

```
Search(NODE *start, key)
{
  Node* current = start
  while (current != NULL)
  {  if (current->info == key)
        return true;
     current = current->front;
  }
  return false;
}
```