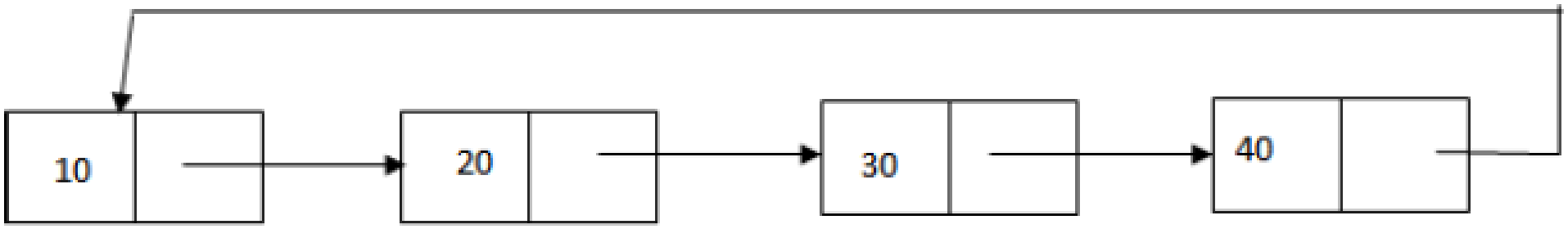
# Circular Singly Linked List

# Disadvantages of Singly Linked List

- There is only one link field and hence traversing is done in only one direction
- To delete a designated node X, address of the first node in the list should be given

# Circular Singly Linked List

- A circular list is a variation of the ordinary list in which link field of the last node contains the address of the first node.

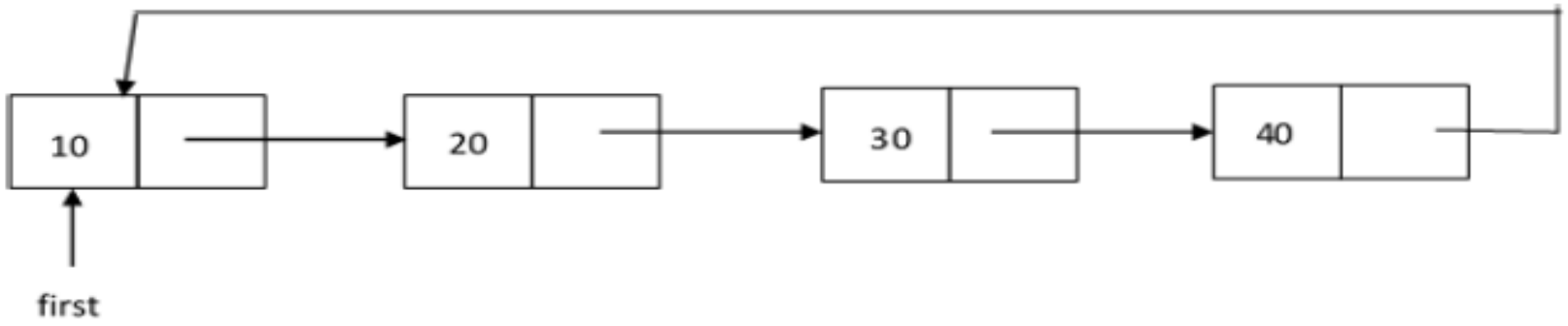


# Advantages of Circular List

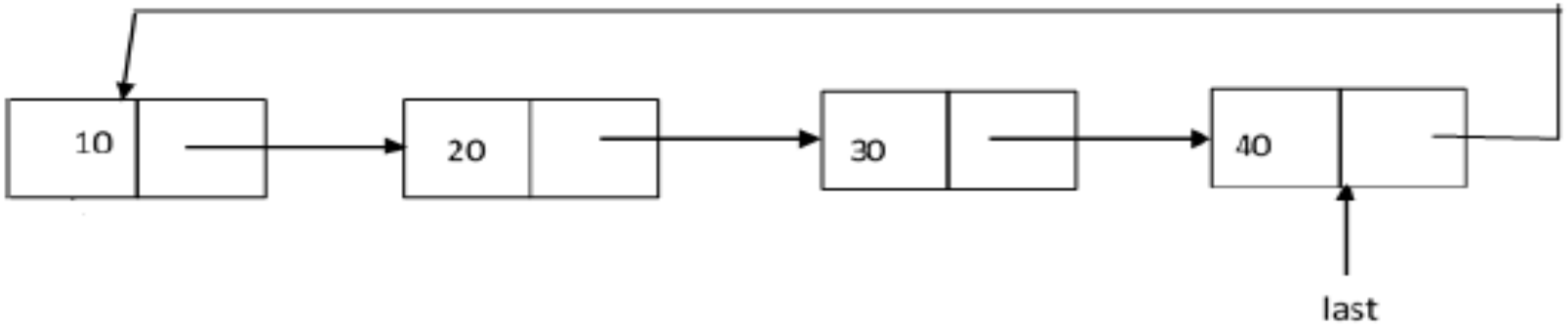
- Every node is accessible from a given node by traversing successively using the link field
- To delete a node, the address of the first node is not necessary. Search for the predecessor of the current node can be initiated by cur itself.

# Approaches

- A pointer ***first*** is designated to the starting node of the list. Traverse the list till the last element (which is the predecessor of the designated first)




- A pointer variable last is designated to the last node and the node that follows last, will be the first node of the list.



```
class cnode
{
 int info;
 cnode* next;
public:
 cnode* insrt(cnode*);
 cnode* insfrnt(cnode*);
 cnode* insfrl(cnode*);
 cnode* inslas(cnode*);
 cnode* rem_dup(cnode*);
 cnode* delle(cnode*);
 cnode* dellb(cnode*);
 cnode* delfe(cnode*);
 void print(cnode*);
 void printl(cnode*);
};
```

```
//Inserting in beginning using the last ptr
cnode* cnode::insfrl(cnode *last)
{
 cnode *temp=new cnode;
 cout<<"\nEnter the element:\n";
 cin>>temp->info;
 if(last==NULL)
 {
   last=temp;
   temp->next=last;
 }
 else
 {
  temp->next=last->next;
  last->next=temp;
 }
 return last;
}
```

```
//Inserting in end using the last ptr
cnode* cnode::inslas(cnode *last)
{
   cnode *temp=new cnode;
 cout<<"\nEnter the element:\n";
 cin>>temp->info;
 if(last==NULL)
 {
   last=temp;
   temp->next=last;
 }
 else
 {
  temp->next=last->next;
  last->next=temp;
  last=temp;
 }
 return last;
}
```

```
//Inserting in end using the first ptr
cnode* cnode::insrt(cnode *head)
{
 cnode *temp=new cnode;
 cnode *cur;
 cout<<"Enter the value to be inserted:";
 cin>>temp->info;
 temp->next=NULL;
 if(head==NULL) {
  head=temp;
  temp->next=head;
 }
 else {
   cur=head;
   while(cur->next!=head)
                    cur=cur->next;
   cur->next=temp;
   temp->next=head;
 }
 return head;
}
```

```
//Inserting in beginning using the first ptr
cnode* cnode::insfrnt(cnode *head)
{
 cnode *temp=new cnode,*cur=head;
 cout<<"Enter the value to be inserted:";
 cin>>temp->info;
 temp->next=NULL;
 if(head==NULL) {
  head=temp;
  temp->next=head;
 }
 else {
  temp->next=head;
  while(cur->next!=head)
     cur=cur->next;
  cur->next=temp;
  head=temp;
 }
 return head;
}
```

```
void cnode::print(cnode *head)
{
 cnode *h=head;
 cout<<h->info<<"->";
 h=h->next;
 while(h!=head)
 {
  cout<<h->info<<"->";
  h=h->next;
 }
}

void cnode::printl(cnode *last)
{
 cnode *h=last->next;
 while(h!=last)
 {
  cout<<h->info<<"->";
  h=h->next;
 }
 cout<<h->info;
}
```

# //Deleting an element from the end using first pointer

```
cnode* cnode::delfe(cnode *head)
{
 cnode *cur;
 if(head==NULL)
 {
  cout<<"\nNo records to delete";
  return NULL;
 }
 if(head->next==head)
 {
  cout<<"\nDeleted item:"<<head->info;
  delete head;
  return NULL;
 }
 cur=head;
   while((cur->next)->next!=head)
   {
     cur=cur->next;
   }
   cnode *t=cur->next;
   cur->next=head;
   cout<<"\nItem deleted:"<<t->info;
   delete t;
   return head;
}
```

## //Deleting and element from the end using a last pointer

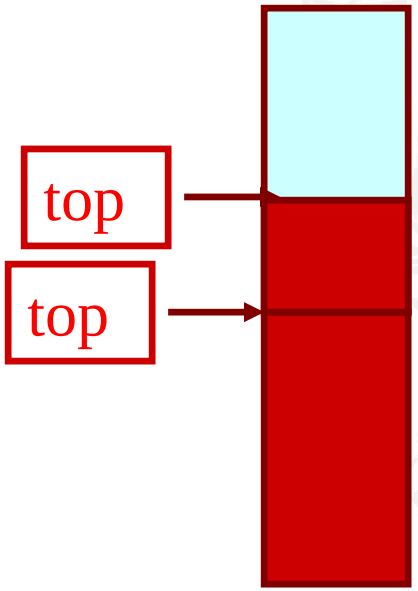
```
cnode* cnode::delle(cnode *last)
{
 if(last==NULL)
 {
  cout<<"\nNo elements to delete:";
  return NULL;
 }
 if(last->next==last)
 {
  cout<<"Element deleted is:"<<last->info;
  delete (last);
  return NULL;
 }
 cnode *cur=last->next;
   while(cur->next!=last)
   {
    cur=cur->next;
   }
   cur->next=last->next;
   cout<<"\nItem deleted: "<<last->info;
   delete(last);
   last=cur;
  return last;
 }
```

//Deleting an element from the beginning using last pointer

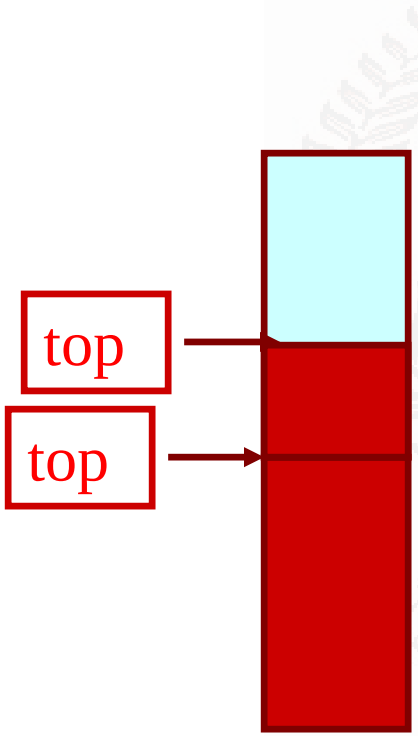
```
cnode* cnode::dellb(cnode* last)
{
 cnode *cur;
 if(last==NULL)
 {
  cout<<"\nNo nodes to delete";
  return NULL;
 }
 if(last->next==last)
 {
  cout<<"Element deleted is: "<<last->info;
  delete (last->next);
  return NULL;
 }
 cur=last->next;
  last->next=cur->next;
  cout<<"\nItem deleted:"<<cur->info;
  delete cur;
  return last;
 }
```

# Stack Implementations: Using Array and Linked List

# STACK USING ARRAY

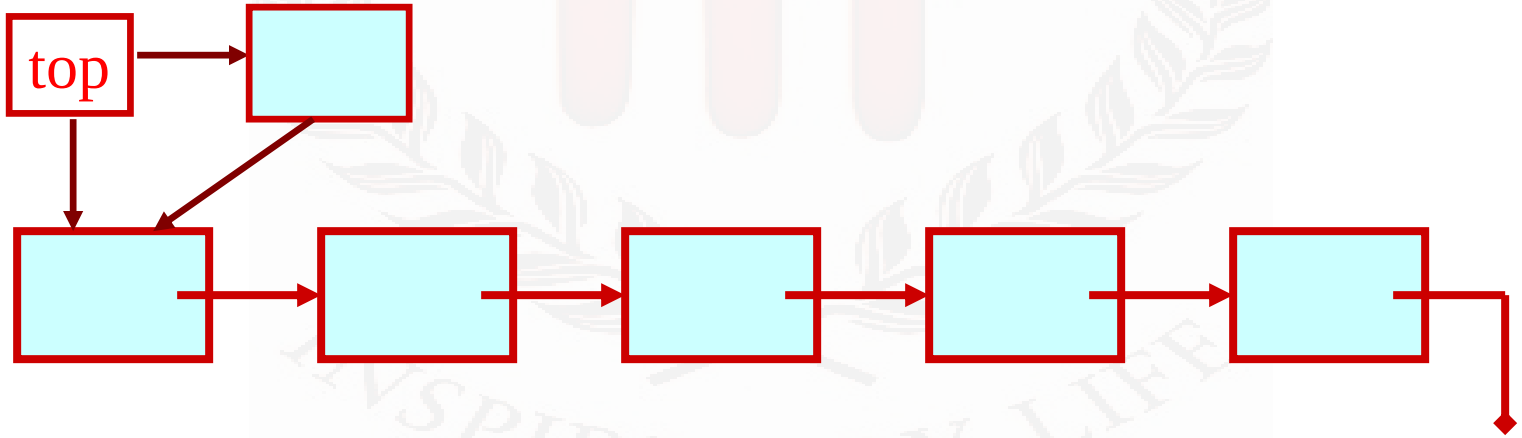


# STACK USING ARRAY



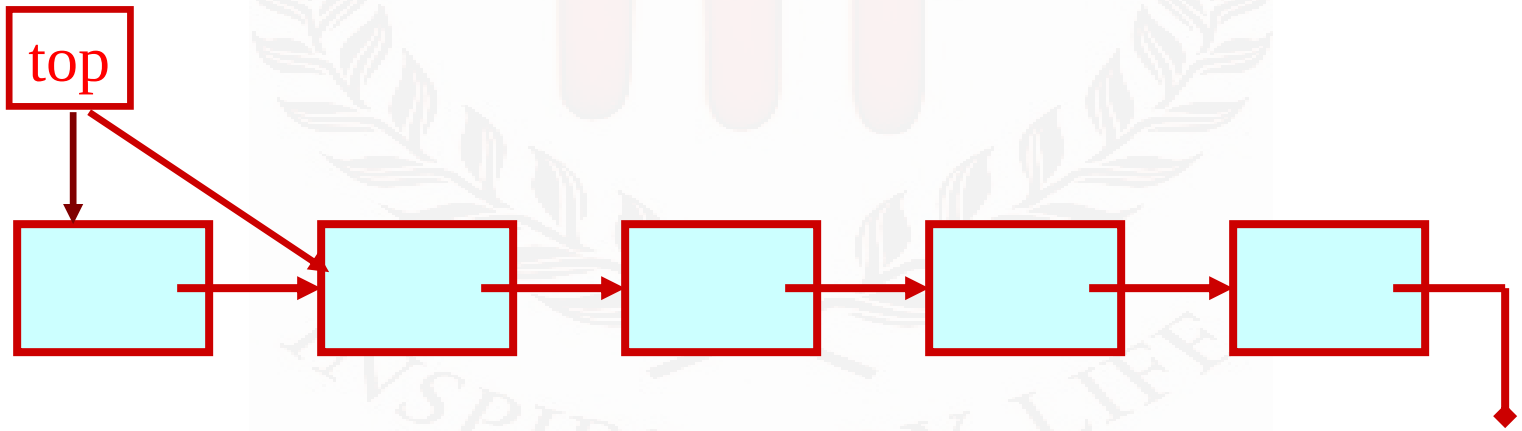
# Stack: Linked List Structure

PUSH OPERATION



# Stack: Linked List Structure

POP OPERATION



# QUEUE: LINKED LIST STRUCTURE