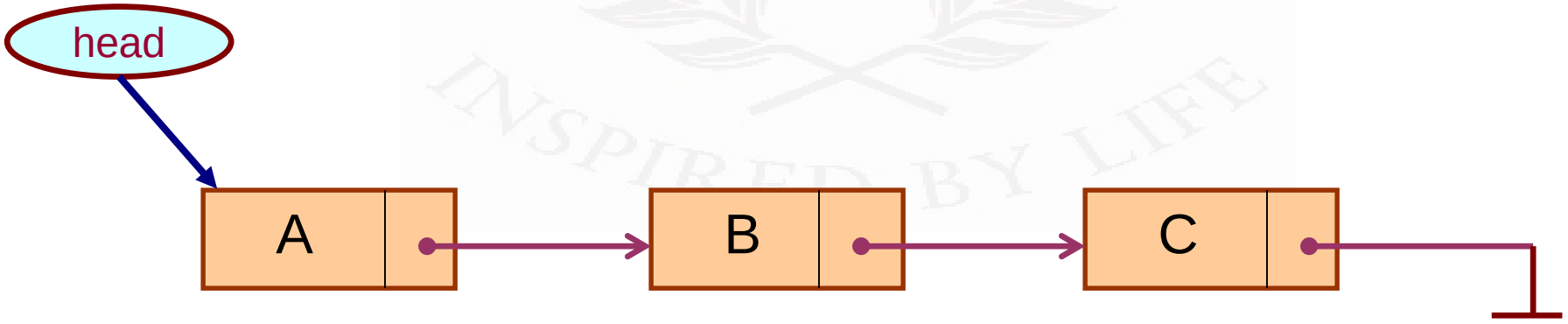
# Linked list

# Why Linked Lists?

- Advantages of Arrays:
  - ❖ Data access is faster
  - ❖ Simple
- Disadvantages:
  - ❖Size of the array is fixed.
  - ❖Array items are stored contiguously.
  - ❖Insertion and deletion operations involve tedious job of shifting the elements with respect to the index of the array.

# Introduction

- A linked list is a data structure which can change during execution.
  - Successive elements are connected by pointers.
  - Last element points to `NULL`.
  - It can grow or shrink in size during execution of a program.
  - It can be made just as long as required.
  - It does not waste memory space.

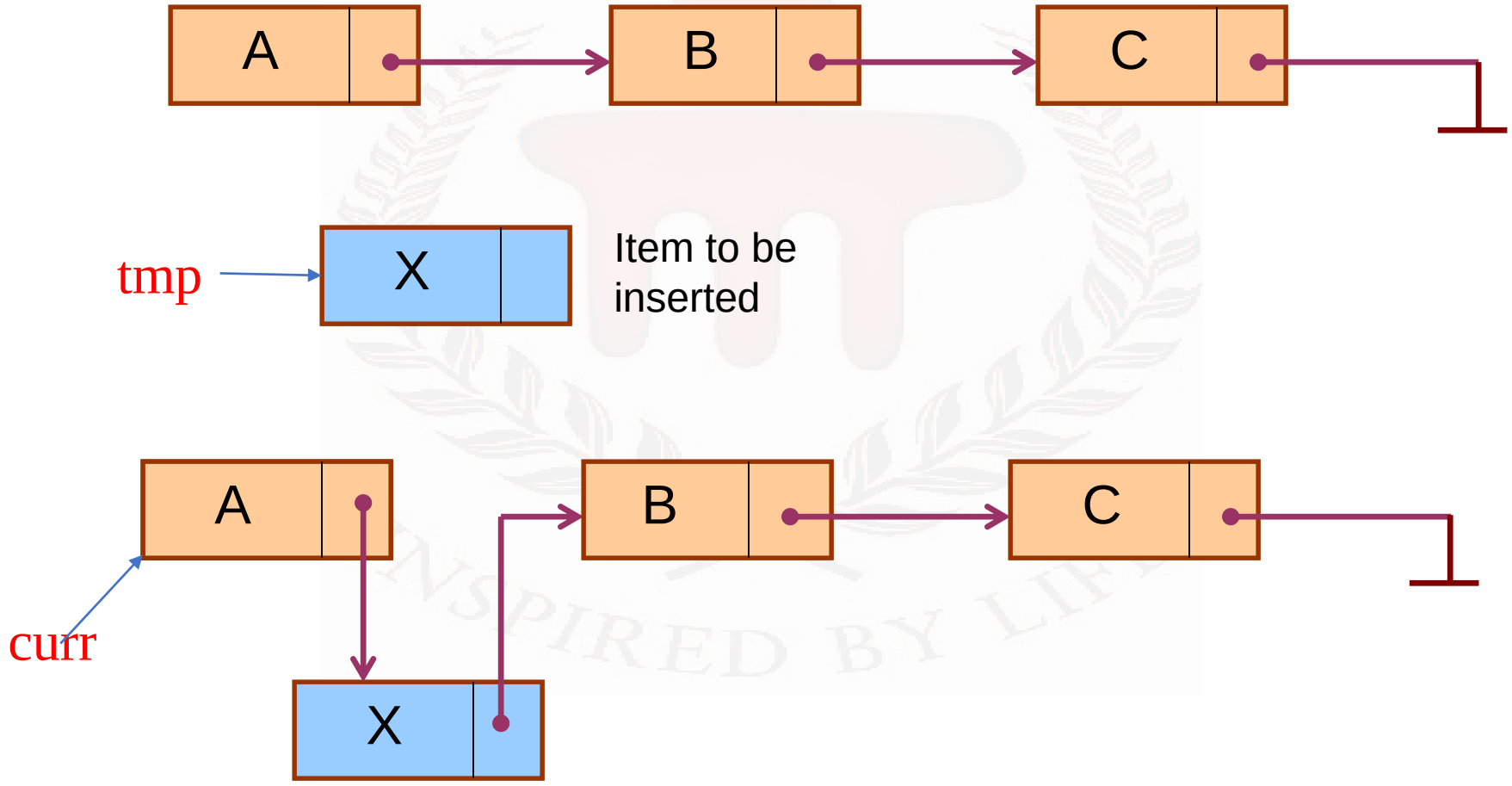


# Introduction

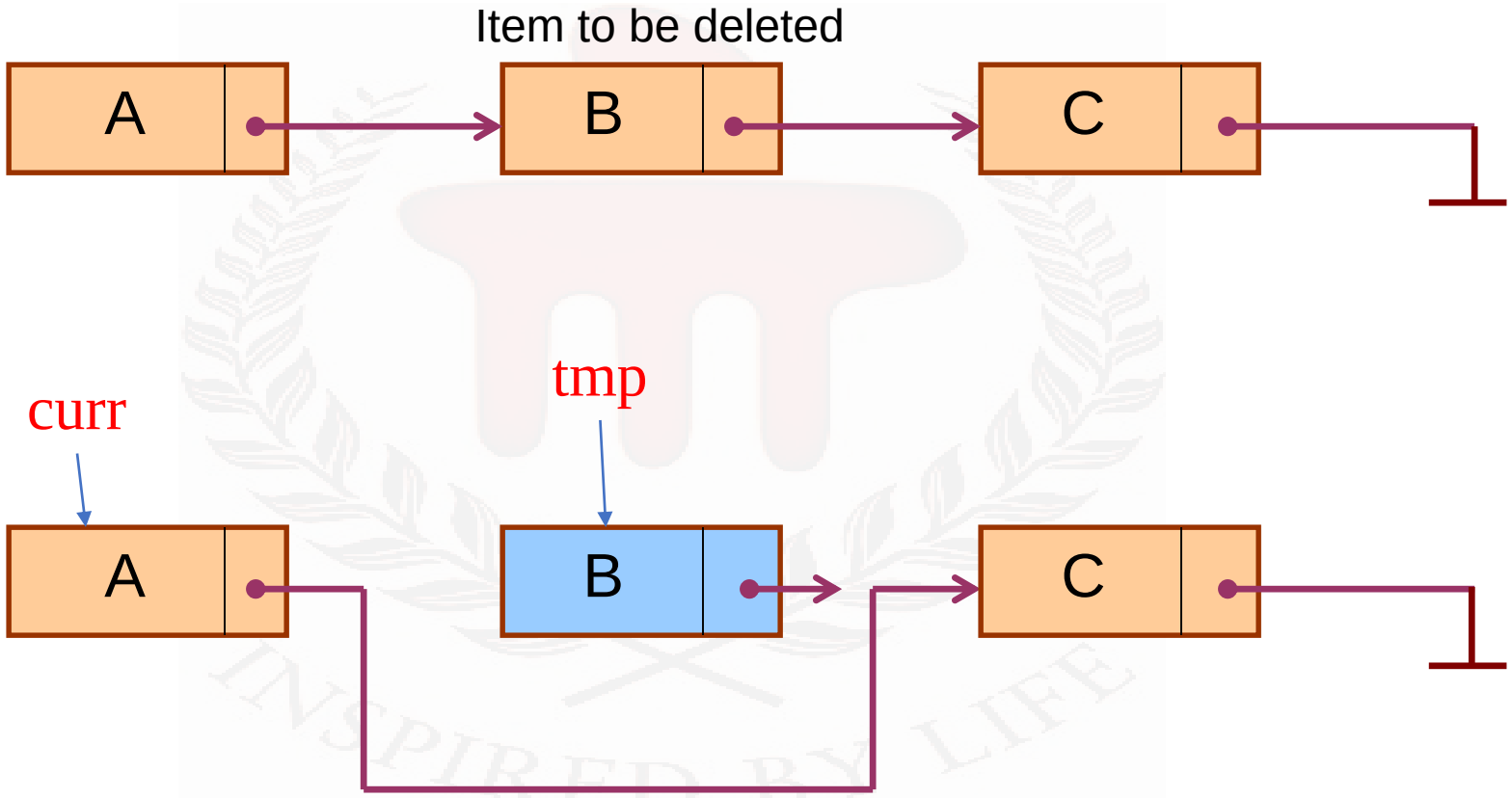
- Keeping track of a linked list:
  - Must know the pointer to the first element of the list (called *start*, *head*, etc.).

- Linked lists provide flexibility in allowing the items to be rearranged efficiently.
  - Insert an element.
  - Delete an element.

# Illustration: Insertion



# Illustration: Deletion



# Summary

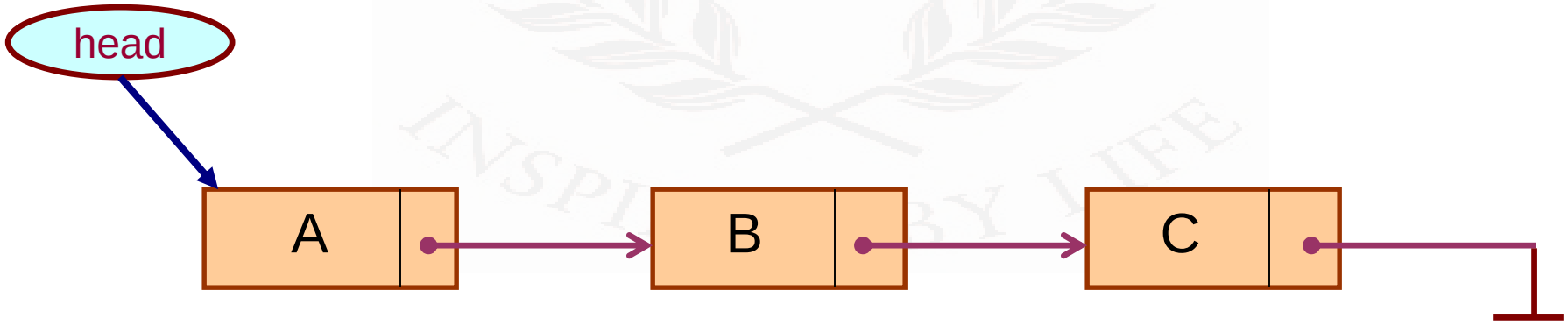
- For insertion:
  - A record is created holding the new item.
  - The next pointer of the new record is set to link it to the item which is to follow it in the list.
  - The next pointer of the item which is to precede it must be modified to point to the new item.
- For deletion:
  - The next pointer of the item immediately preceding the one to be deleted is altered, and made to point to the item following the deleted item.

# Array versus Linked Lists

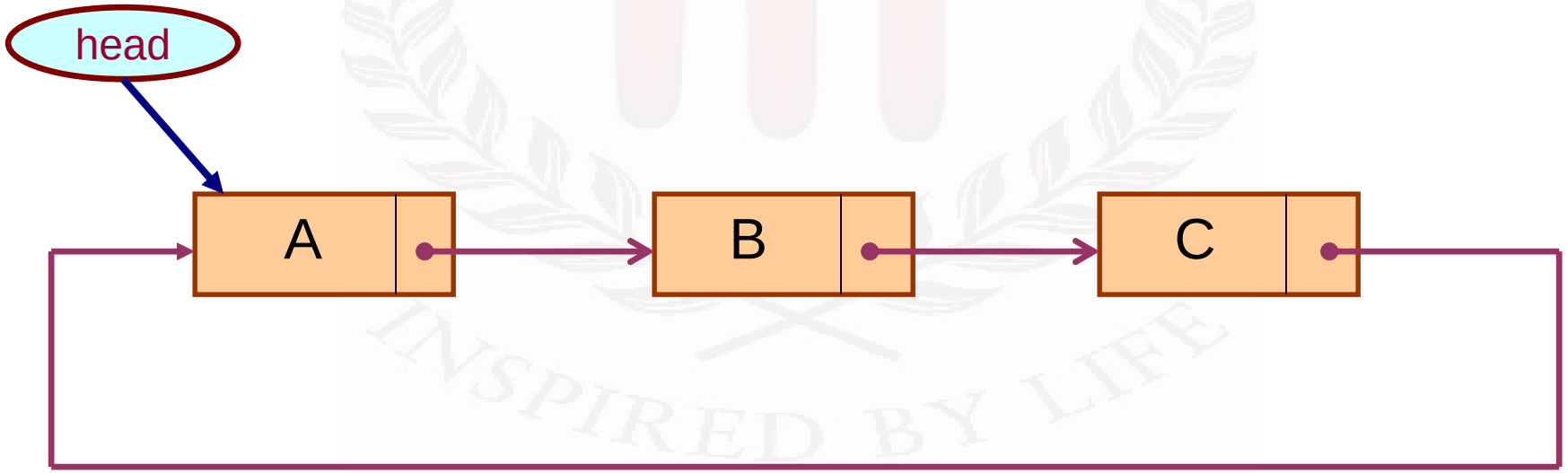
- Arrays are suitable for:
  - Inserting/deleting an element at the end.
  - Randomly accessing any element.
  - Searching the list for a particular value.
- Linked lists are suitable for:
  - Inserting an element.
  - Deleting an element.
  - Applications where sequential access is required.
  - In situations where the number of elements cannot be predicted beforehand.

# Types of Lists

- Depending on the way in which the links are used to maintain adjacency, several different types of linked lists are possible.
  - Linear singly-linked list (or simply linear list)
    - One we have discussed so far.

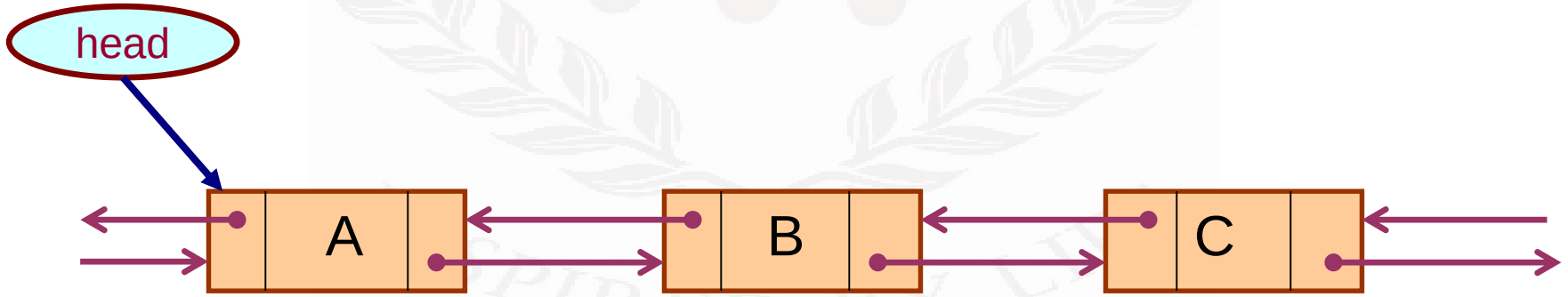


- Circular linked list
  - The pointer from the last element in the list points back to the first element.



- Doubly linked list
  - Pointers exist between adjacent nodes in both directions.
  - The list can be traversed either forward or backward.

.



# Basic Operations on a List

- Creating a list
- Traversing the list
- Inserting an item in the list
- Deleting an item from the list
- Concatenating two lists into one

# Implementing Linked Lists: Singly Linked List(SLL)

```
class node
{

    int info;
    node *next;
   public:
          node();
          void insert_beg();   void ins_end();    void ins_pos();
          void Ins_before(); void ins_after();
          void print();
          void del_beg();  void delete_end();  void del_pos();  void del_item();
          void del_before(); void del_after();

}*head;
```

```
node::node()
{


}
```

```
Void node:: ins_end()
{
 node *cur;
 cur=head;
 node *temp=new node;
 cout<<"Enter the value:";
 cin>>temp->info;
 temp->next=NULL;
 if(head==NULL)
 {
    head=temp;
 }
 else
   {
            while(cur->next!=NULL){
            cur=cur->next;}
            cur->next=temp;
   }}
```

```
void node::ins_beg()
{
 node *temp=new node;
 cout<<"Enter the value:";
 cin>>temp->info;
 temp->next=NULL;
 if(head==NULL)
 {
    head=temp;
 }
 else
    {
     temp->next=head;
     head=temp;
    }
}
```

```
void node::ins_pos()
{
// node *temp=new node;
 node *t,*t1;
 cout<<"Enter the value to be inserted \n";
 cin>>temp->info;
 temp->next=NULL;
 cout<<"Enter the position:\n";
 int pos;
 cin>>pos;
```

```
 if(head==NULL)
  {
   head=temp;
  }
  else
  {
   t=head;
   for(int i=1;i<pos-1;i++)
      t=t->next;
   t1=t->next;
   t->next=temp;
   temp->next=t1;
  }
 }
```

```
void node::print()
{
  node *h=head;
  if(h==NULL)
    cout<<"List is empty\n";
  while(h!=NULL)
  {
   cout<<"->"<<h->info;
   h=h->next;
  }
  cout<<endl;
}
```

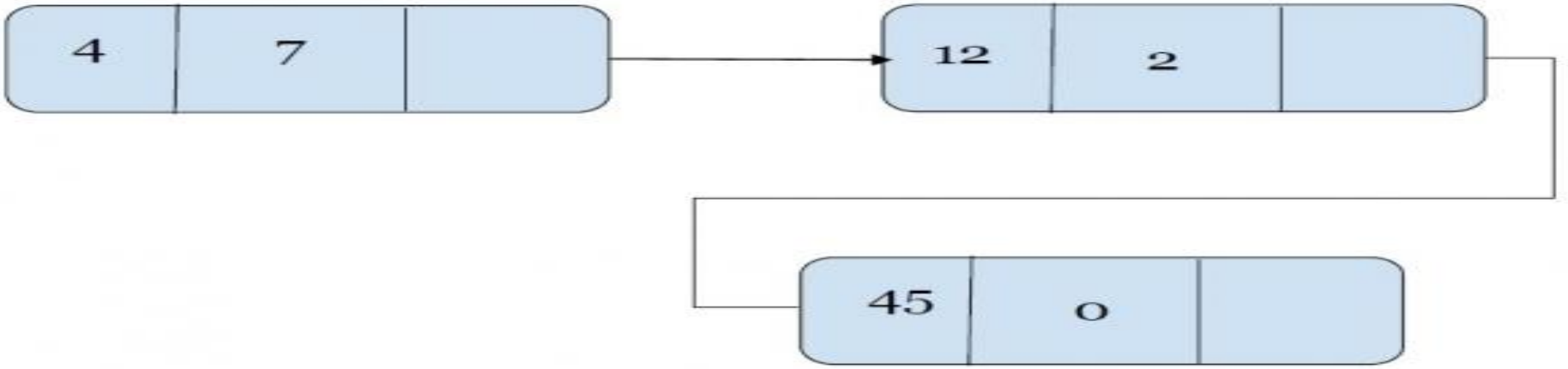
```
void node::del_beg()
{
 node *temp=new node;
 temp=head;
 head=head->next;
 cout<<"\nDeleted element is:"<<temp->info;
 delete(temp);
}
```

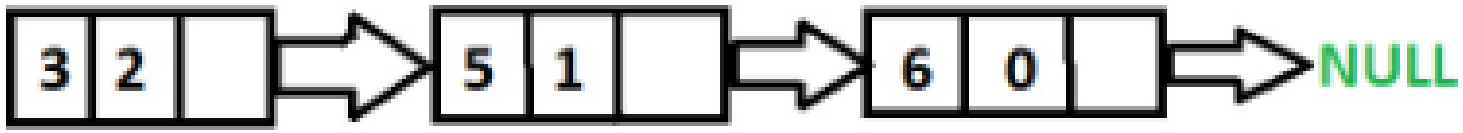
```
void node::del_item()
{
 node *cur, *prev;
 int data;
 if(head==NULL)
    cout<<"No records to delete\n";
 else
 {
  cout<<"Enter the data to be deleted: ";
  cin>>data;
  cur=head;
  while((cur!=NULL)&&(cur->info!=data))
  {
   prev=cur;
   cur=cur->next;
  }
  if(cur==head)
  {
   head=head->next;
   cout<<"Data Deleted: "<<data<<endl;
  }
  if(cur==NULL)
  {
   cout<<"Record not found\n";
   return;
  }
  else
  {
   prev->next=cur->next;
   cout<<"Data deleted is: "<<data<<endl;
  }
  delete(cur);
 }
}
```

```
int main()
{
 node n1;
 while(1)
 {
  cout<<"1.Insert Beginning \t 2. Insert end \t 3. Insert anywhere \t 4. Print\t 5. Delete Beg \t 6. Delete from Anywhere \t 7. Exit\n";
  int ch;
  cin>>ch;
  switch(ch)
  {
   case 1: n1.insert_beg();
                        break;
   case 2: n1.ins_end(); break;
   case 3: n1.ins();break;
   case 4: n1.print(); break;
   case 5: n1.del_beg();break;
   case 6: n1.del_pos(); break;
   case 7: exit(0);
  }
 }
 return 0;
}
```

# Polynomial represention:

Polynomial : $4x^7 + 12x^2 + 45$




```
Node {
    int coeff, power;
    Node* next;
};
```

What is the polynomial?

# Polynomials – SLL

```
void printList( Node* ptr)
{
   while (ptr->next != NULL) {
       cout << ptr->coeff << "x^" << ptr->power ;
      if( ptr->next!=NULL && ptr->next->coeff >=0)
         cout << "+";


       ptr = ptr->next;
   }
   cout << ptr->coeff << "\n";
}
```

# Practice Questions

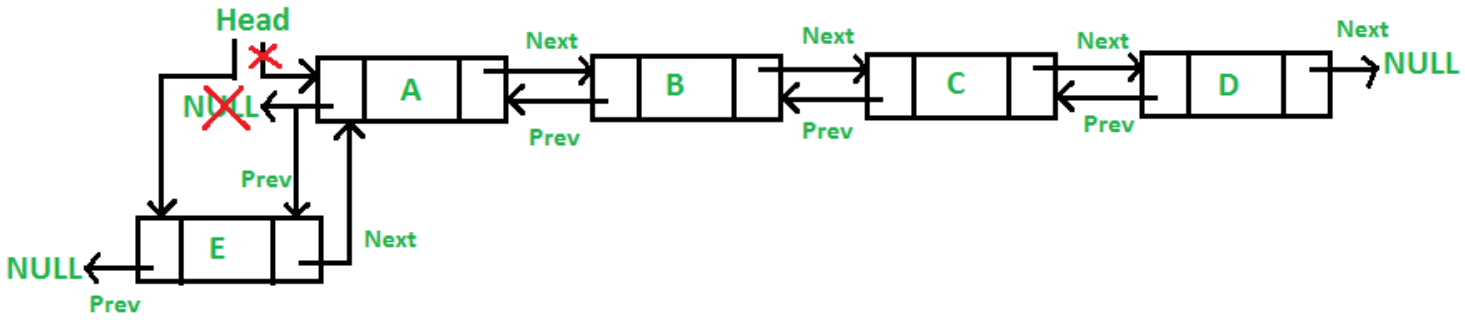
- Concatenate two lists
- Merge two lists
- Reverse a list.

# Doubly Linked lists

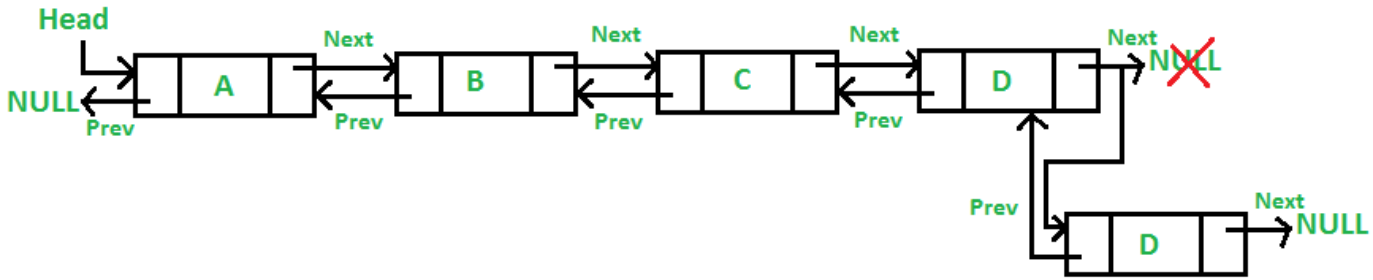
# class dnode

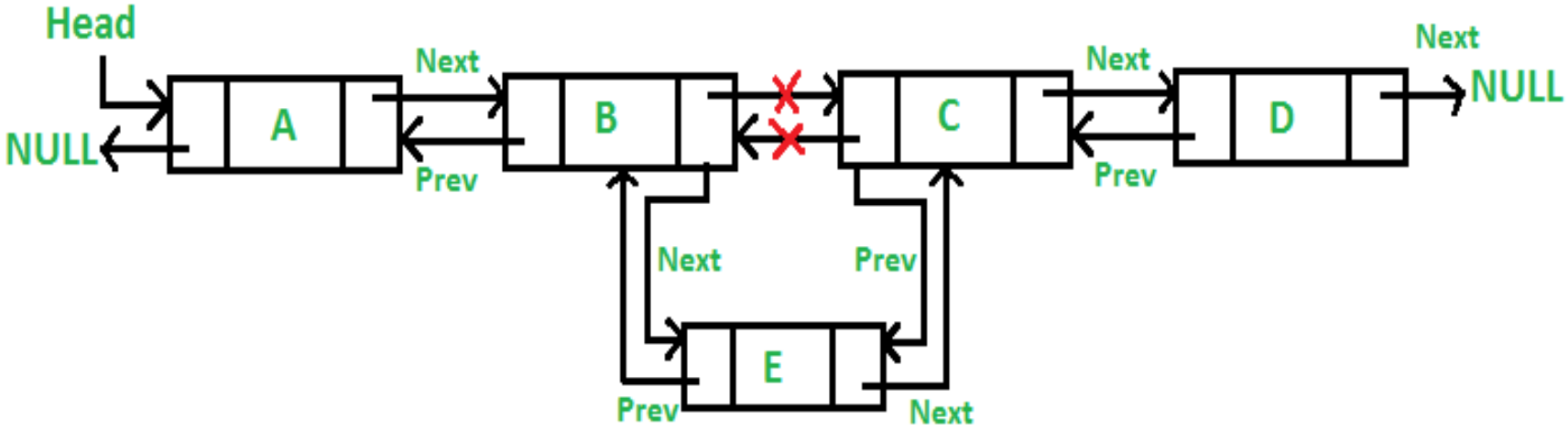
```
{
  int info;
  dnode *next;
  dnode *prev;
 public:
  dnode* insb(dnode*);
  dnode* inse(dnode*);
  void deledata(int);
  void print(dnode*);
};
```

```
dnode* dnode::insb(dnode *head)
{
 dnode *temp=new dnode;
 cout<<"\nInfo: ";
 cin>>temp->info;
 temp->prev=temp->next=NULL;
 if(head==NULL)
 {
   head=temp;
   return head;
 }
 head->prev=temp;
 temp->next=head;
 head=temp;
 return head;
}
```



```
dnode *dnode::inse(dnode *head)
{
 dnode *temp=new dnode;
 cout<<"\nInfo: ";
 cin>>temp->info;
 temp->prev=temp->next=NULL;
 if(head==NULL)
 {
   head=temp;
   return head;
 }
 dnode *cur=head;
 while(cur->next!=NULL){
  cur=cur->next;
 }
 cur->next=temp;
 temp->prev=cur;
 return head;
}
```




Head
NULL
A
Next
Prev
B
C
Next
Prev
D
Next
NULL
Next
Prev
E
Prev
Next

```
/* Given a node as prev_node, insert
a new node after the given node */
void insertAfter(Node* prev_node, int new_data){
        /*1. check if the given prev_node is NULL */
        if (prev_node == NULL)
        {
                cout<<"the given previous node cannot
be NULL";
                return;
        }
        /* 2. allocate new node */
        Node* new_node = new Node();
        /* 3. put in the data */
        new_node->data = new_data;
        /* 4. Make next of new node as next of prev_node */
        new_node->next = prev_node->next;
```

- /* 5. Make the next of prev_node as new_node */
- prev_node->next = new_node;

- /* 6. Make prev_node as previous of new_node */
- new_node->prev = prev_node;

- /* 7. Change previous of new_node's next node */
- if (new_node->next != NULL)
- new_node->next->prev = new_node;
- }

- Reference:geek2geeks

- **Deletion of a node can be handled in two steps if**
  - **The pointer of the node to be deleted and the head pointer is known.**
    - If the node to be deleted is the head node then make the next node as head.
    - If a node is deleted, connect the next and previous node of the deleted node.

```
void deldata(int num)
{
   if(head != NULL)
   {
      link * cur_ptr, *prev_ptr, *del_ptr;
      cur_ptr = head;
      prev_ptr = cur_ptr;
      while(cur_ptr->next != NULL)
      {
         if(head->data == num)
         {
            del_ptr = cur_ptr;
            head = cur_ptr->next;
            head->prev = NULL;
            free(del_ptr);
          }
```

```
if(cur_ptr->data == num)
         {
            del_ptr = cur_ptr;
            prev_ptr->next = cur_ptr->next;
            cur_ptr->next->prev = prev_ptr;
            free(del_ptr);
            cur_ptr = prev_ptr;
         }
         prev_ptr = cur_ptr;
         cur_ptr = cur_ptr->next;
      }
```

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

```
void main()
{
 clrscr();
 dnode d,*head=NULL;
 int c,ele;
 for(;;)
 {
  cout<<"1.Ins b \n 2.Ins e\n 3.Print \n 4.del f\n 5. Del e\n";
  cin>>c;
  switch(c)
  {
   case 1:head=d.insb(head);
                        break;
   case 2:head=d.inse(head);
                        break;
   case 3: d.print(head); break;
   case 4: cout<<“enter element to delete”;
                       cin>>ele;
                       d.deldata(ele);
   default:exit(0);
  }
 }
}
```

# Circular Doubly linked list