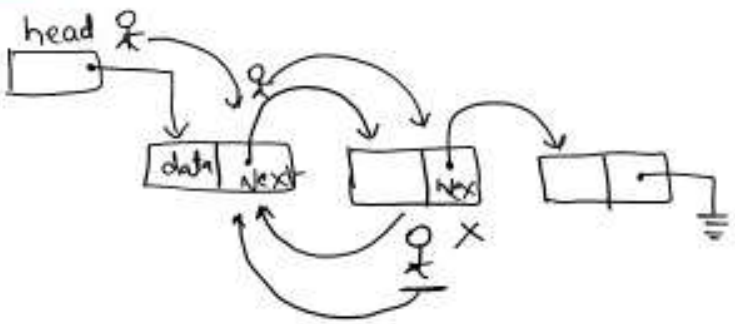
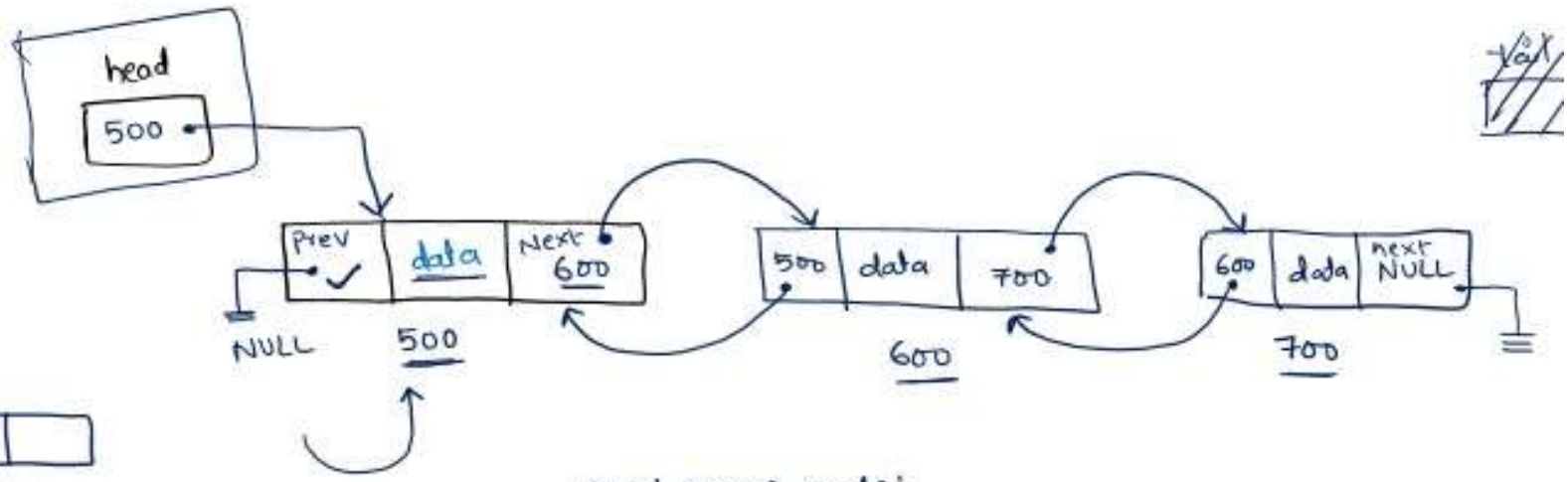
Double linked List:-



Problem:- of S.LL we can traverse only in single direction: from head to last node but we can not go back

DLL:- In DLL we are maintaining previous node address



```
struct Node
{
    int data;
    Struct Node *prev;
    Struct Node *next;
}
```

Creat e new node:

```
struct Node *nn = (struct xlode *) malloc (sizeof (st

Struct Node *head = NULL;

int n = 0;
```

no of elements in the list

① insert ✓
   ↳ at begin, end, given position

② delete
   ↳ at begin, end, given position

③ display.

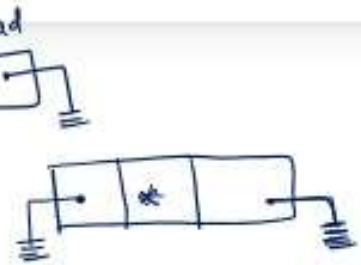
④ search

flag = 0
status = 1

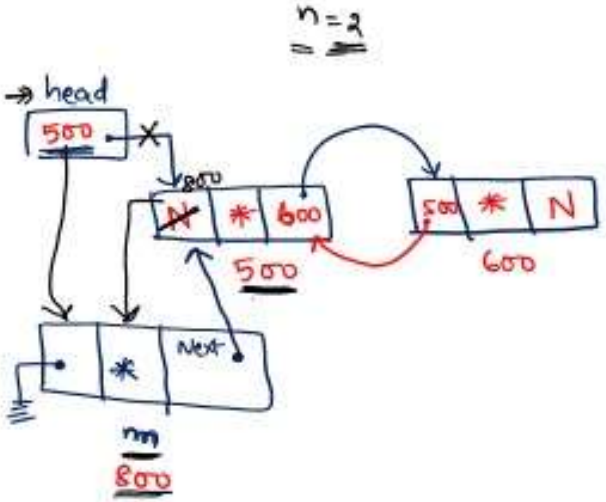
insert at begin;

```
void insert at begin(int item);
{
 struct Node *nn = (struct Node *) malloc(sizeof(SN));
    nn → prev = NULL;
    nn → data = item;
 if ( head == NULL)
   {
       nn → next = NULL;
   }
 else{
       nn → next = head;
       head → prev = nn;
       head = nn;
   }
}     n++; // it shows no of elements in list-
```

Case 1:



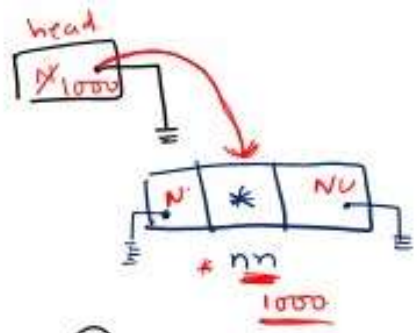
Case 2:



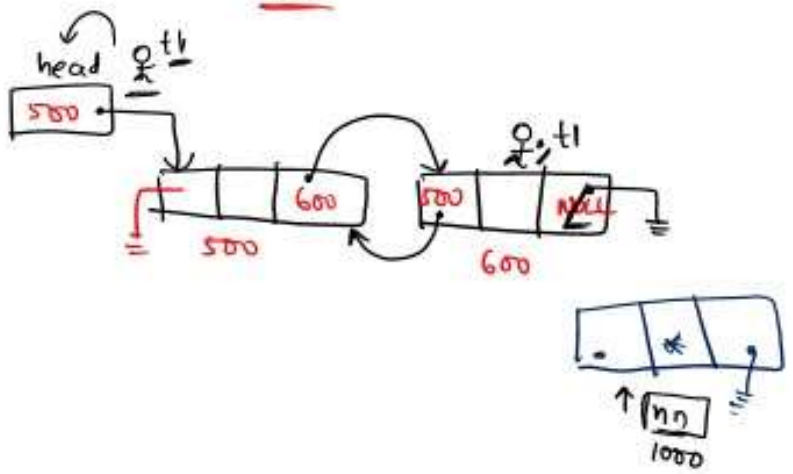
insert at end:-

```
void insertAtEnd(int item)
{
struct Node *nn = (struct Node *) malloc( sizeof(struct N));
   nn->data = item;
   nn->next = NULL;
   if (head == NULL)
   {
      nn->prev = NULL;
      head = nn;
   }
   else{
        struct Node *tl = head;
        while (tl->next != NULL)
        {
           tl = tl->next;
        }
         tl->next = nn;
         nn->prev = tl;
              n++;
}
```

Case 1:-



Case 2:-



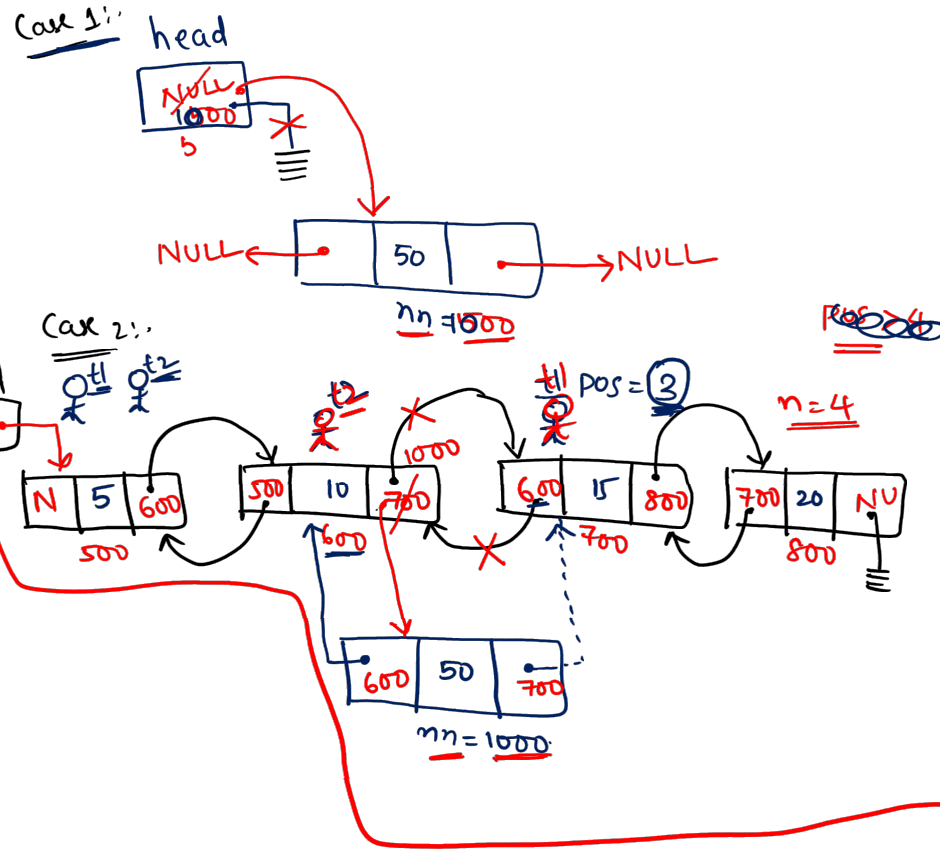
'insert at given position;. (pos)

n → no. of elements

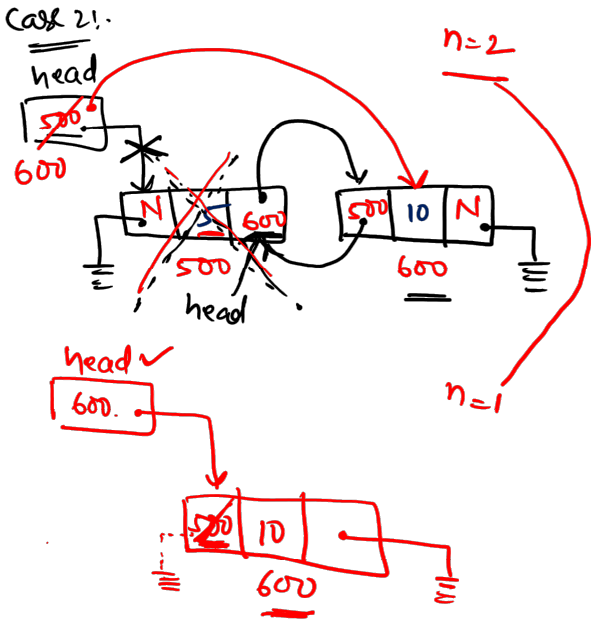
```
void insertatGP(int item, int pos)
{
struct Node *nn = (struct Node *)malloc(sizeof(struct Node));
   nn->data = item;

  if(head==NULL)          // Case 1:
  {
      nn->next = NULL;
      nn->prev = NULL;
       head = nn;
  }
  else   // case 2:
  {
     if (pos > n)
     { print("invalid position");
     }
     else
      {
```

```
struct Node *t1 = head;
Struct Node *t2;

for(i=1; i< pos ; i++)
{     t2=t1;
      t1 = t1->next;
}
     nn->next = t2->next
     nn->prev = t1->prev;
     t2->next = nn;
     t1->prev = nn;
```

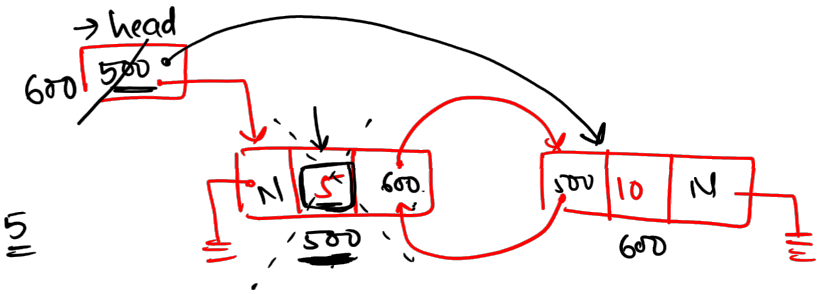
```
print("element inserted");
      n++;
}
```

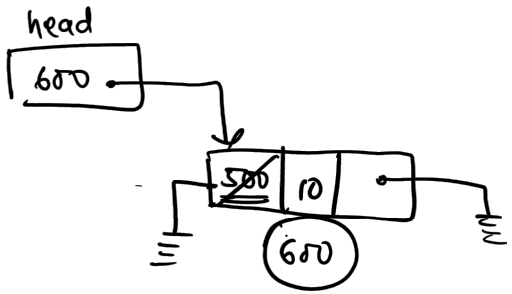


```
void delete at begin()
{
    if (head == NULL)
    {   printf("List is Empty - can't delete");
    }
    else
    {   printf("deleted element is %d", head->data");
        head = head->next;
        head->prev = NULL;
    } n--;
}
```



```
else
{
    printf("deleted element %d", head->data"); 5

        head = head->next;

        head->prev = NULL;

        n--; ||

}
```



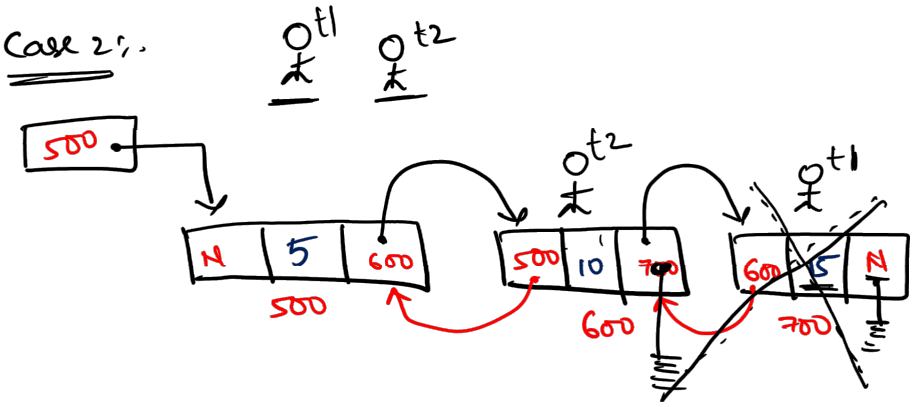


delete at end:- ✓

```
void delete at end( )
{
   if(head == NULL)
    {
      printf("List is Empty -> Can't delete");
    }

   else{
        struct Node *t1 = head;
        struct Node *t2;

     while(t1 -> next != NULL)
      {
          t2 = t1;
          t1 = t1 -> next;
      }
      t1 -> prev = NULL; // optional
      t2 -> next = NULL;

}
```

Case 1:.

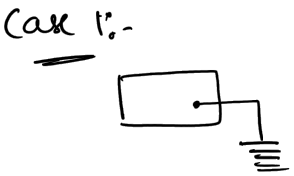
Case 2:.



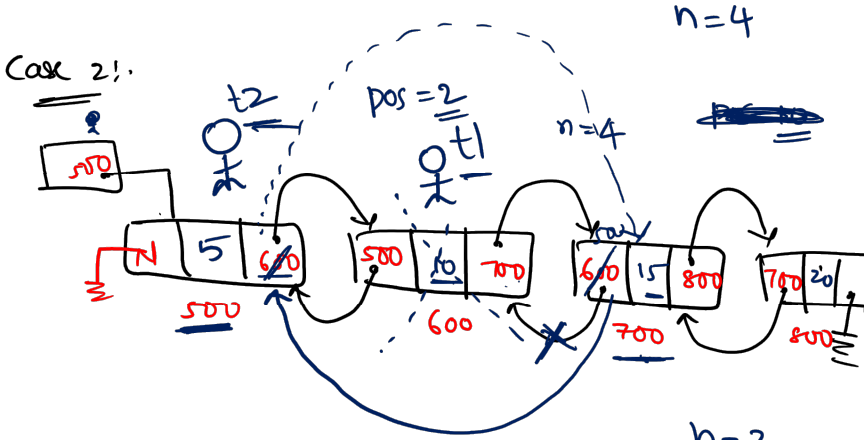
```
void delete at given position (int pos)
{
 if (head == NULL):
  { printf(" List is Empty - Can not delete");
  }
 else
 {
    if ( pos>n)
    { printf(" Invalid position - Can n't delete");
    }
    else
    {
       Struct Node *t1 = head;
       Struct Node *t2;
      for(i=1; i<pos; i++)
      {    t2=t1
           t1 = t1->next;
      }
```

```
      t2->next = t1 next
      (t1->next) -> prev = t2 ;

      n--;
    }
```

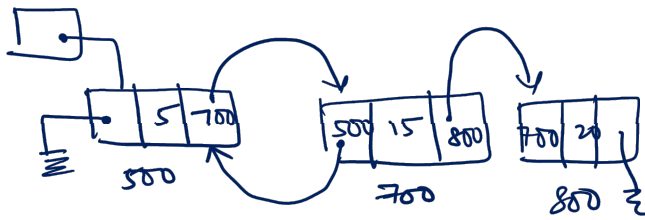
Case 1:-



Case 2:-

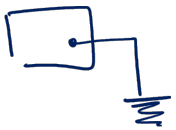


n=3

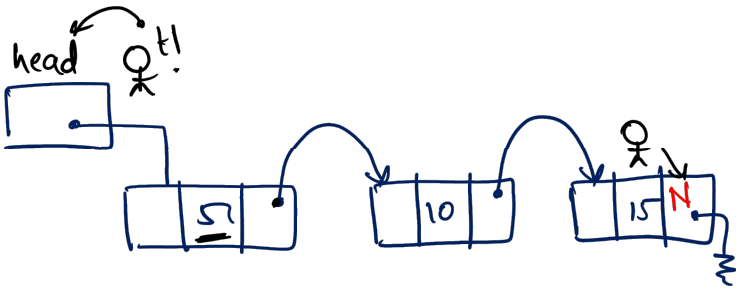


```
void display()
{
    if(head ==NULL)
    {
      print(" List is Empty");

    }
    else{
        struct Node *tl=head;

    while (tl->next !=NULL)
      {
        printf(" %d", tl->data);
          tl=tl->next;

      }
}
}
```

Case 1:-



Case 2:-



```
void search(int se)
{
    int status = 0;
    int pos = 0
    struct Node *tl = head;

    while( tl->next != NULL)
    {
        if( tl->data == se)
        {
            printf("element found %d", pos);
            status = 1;
        }
        tl = tl->next;
        pos = pos+1;
    }
    if(status == 0)
    {
        printf("not found");
    }
}
```