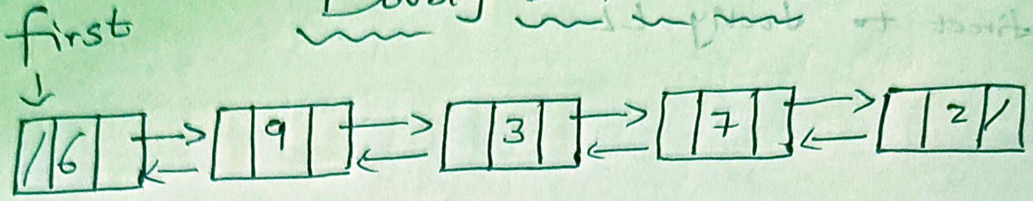
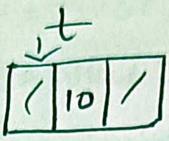
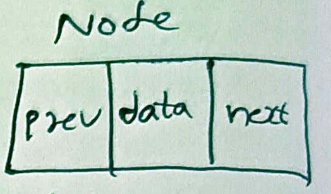
Doubly Linked List







```
struct Node *t;
t= new Node;
t->prev=NULL;
t->data = 10;
t->next=NULL;
```

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

Code pdf ✓

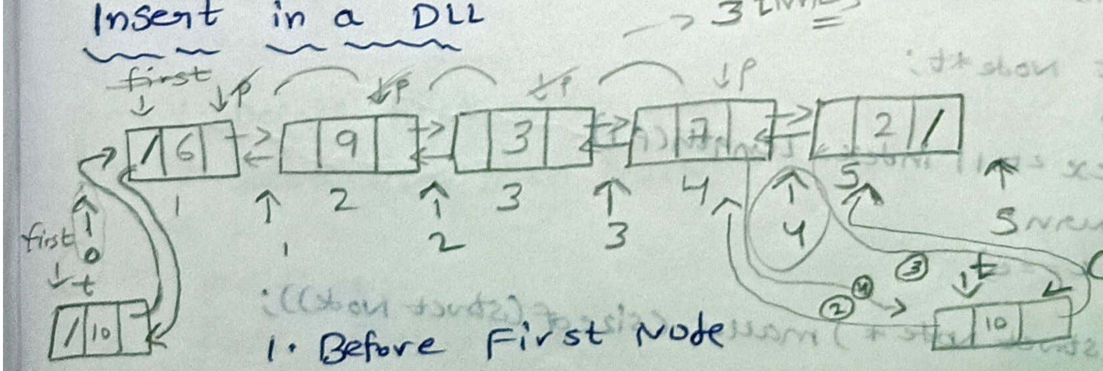
```
#include <stdio.h>          -> stdlib.h
struct Node
{
    struct Node *prev;
    int data;
    struct Node *next;
} *first = NULL;
void create (int A[], int n)
{
   struct Node *t, *last;
   int i;

   first=(struct Node *) malloc(sizeof(struct Node));
   first-> data=A [0];
   first-> prev=first-> next = NULL;
   last=first;

   for(i=1; i<n; i++)
   {
      t=(struct Node *)malloc (sizeof(struct Node));
      t-> data=A[i];
      t-> next= last ->next;
      t-> prev= last;
      last-> next=t;
      last=t;
   }
}
```

```
Void Display(Struct Node *P)
{
    while(P)
    {
      printf("%d ", P->data);
      P=P->next;
    }
    printf("\n");
}
int Length(struct Node *P)
{
  int Len=0;
  while(P)
  {
    len++;
    P=P->next;
  }
  return len;
}
int main()
{
    int A[]={10, 20, 30, 40, 50};
    create(A, 5);
    Printf("\nlength is : %d\n", length(first));
    Display(first);
    return 0;
}
```

Insert in a DLL



1. Before First Node

2. At any given Index.

```
Node *t = new Node;
  t->data =x;
  t->prev = NULL;
  t->next=first;
  first->prev = t;
  first = t;
```

→ for ① Before first Node.

② at any given index.

pos=4

```
  Node *t=new Node;
   t->data=x;
  for(i=0; i<pos-1;i++)
   {
     p=p->next;
   }
  t->next=p->next;
  t->prev= p:
   if(p->next)
   {
    p->next->prev = t;
   }
    p->next=t;
```

min(o(1))
max(o(n))

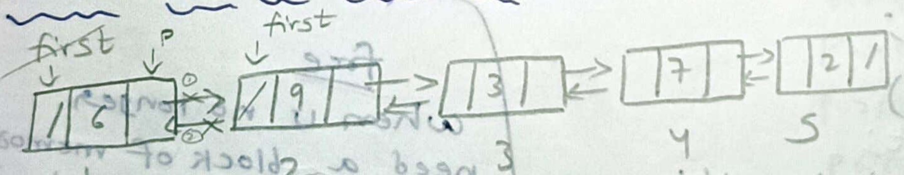
Code for insert

```
void  insert(struct Node *p, int index, int x)
{
   struct Node *t;
   int i;
  if (index <0 || index> Length(p))
      return:
  if(index ==0)
  {
    t= (struct Node *) malloc(sizeof(struct Node));
    t->data=x;
    t->prev= NULL;
    t->next = first;
    first -> prev= t;
    first = t;
}
else
{  for(i=0;i<index-1;i++)
    { p=p->next;
    }
   t=(struct Node*) malloc (Sizeof(struct Node));
   t->data=x;
   t->prev=p;
   t->next=p->next;
   if(p->next)
   {p->next->prev=t;
   }
   p->next=t;

 }
}

int main()
{  int A[]={10, 20, 30, 40, 50};
   create (A, 5);
   Insert(first) 2, 25 ;
   Display (first);
   return 0;
 }
```

Deleting from Doubly LL

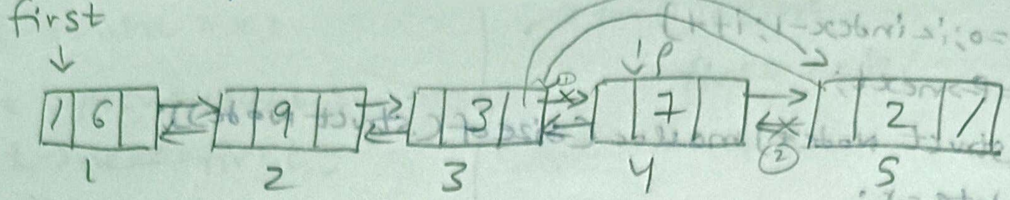


1. Delete first Node;
2. Delete from given index;

①
```
    p=first;
    first = first->next;
    x=p->data;
    delete p;
    if (first)   -> Point in next cage
    {
      first->prev = NULL;
    }
```

Constant time O(1)

② Pos=4



```
p = first;
for (i=0; i<pos-1; i++)
{
  p=p->next;
}
p->prev->next = p->next;
if(p->next)
{ p->next->prev=p->prev;
}
x=p->data;
delete p;
```

We cannot access null's nexts (or) prevs

so... if cond

Code for delete

```
int   Delete (struct Node *p, int index)
{
  ~~struct Node *q;~~
  int  x=-1, i;
  if(index<1 || index>Length(p))
     return -1;
  if (index==1)
  {     p = first;
    first=first->next;
    if(first)
    { first->prev=NULL;
    }
    x=p->data;
    free(p);
  }
  else
  {
   for(i=0;i<index-1; i++)
   {p=p->next;
   }
   p->prev->next=p->next;
   if(p->next)
   {p->next->prev=p->prev;
   }
```

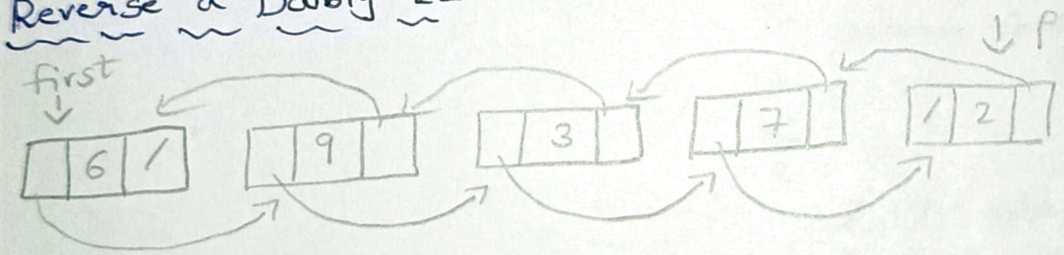
free

when u no longer need a block of memory that was allocated using malloc fn, u can use the free fn to deallocate that memory & make it available for other uses. ①

```
    x = p-> data;
    free(p);
  }
  return x;
}
int main()
{ int A[] = {10, 20, 30, 40, 50}
  create (A, 5);
  Delete (first, 1);
  Display (first);
  return 0;
}
```

o/p 20 to 50

Reverse a Doubly LL



```
p=first;
while(p)
{
   temp = p->next;
   p->next = p->prev;
   p->prev = temp;
   p = p->prev;
   if (p-> next==NULL)
        first=p;
}
```

&& P!=NU

first this can n write

second this

pdf /but SLL's.

Code ∝

→ code not in
pdf inplace of That
SLL's reverse code is There.