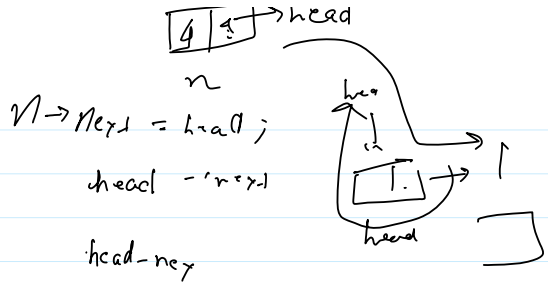
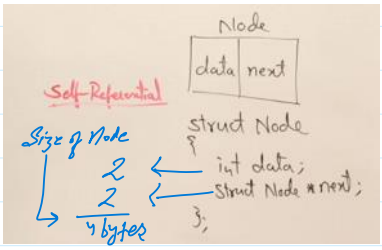
LINKED LIST

19 April 2022 21:21





Struct is support in C & also C++.
In C++ instead of struct it is class.

In class everything default is private &
" Structure " " " is public.

LL Created

* Node is created inside the heap.
* To create LL we need a pointer.

```
Struct Node *p;
p = (StructNode*)malloc(sizeof(StructNode));
```

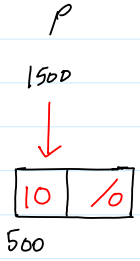
(StructNode*) — Typecaste; sizeof(StructNode) — Size; ; — use in

p → Is in stack

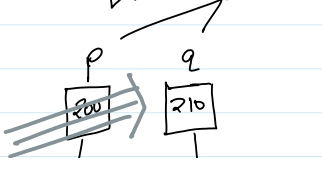
Instead we use

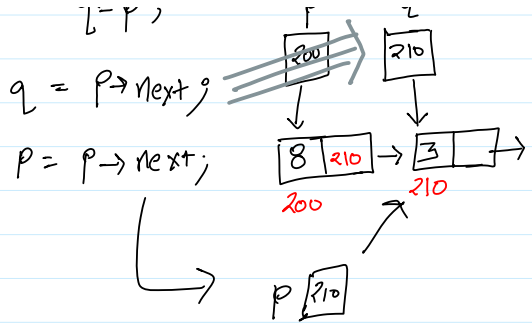
```
p = new node
p -> data = 10;
p -> next = 0;
```

0 — null

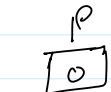


```
Struct Node *p, *q;
q = p;
q = p->next;
```



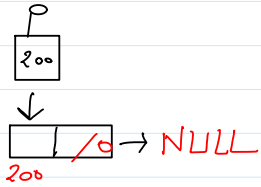



```
Struct Node *p=NULL;
```




```
if (p==NULL)
if (p==0)
if (!p)
```

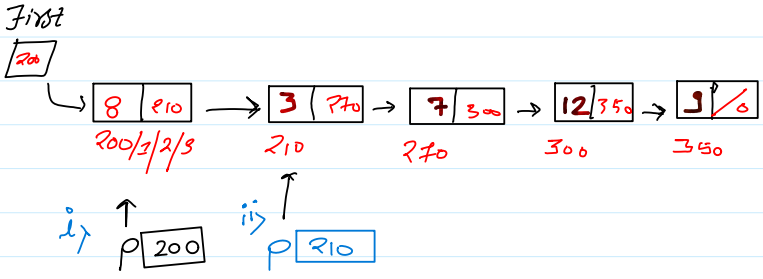
```
if (p->next==NULL)
```




```
if (p->next !=NULL)
```

Traversing

Lets start with first node



i) p 200

ii) p 210

iii) `p = p->next;` → It works repeatedly

So we use while loop

```
while (p!=NULL){
    cout << p->data;
    p=p->next;
}
```

// 8 3 7 12 3

```
while (P != NULL) {
    cout << p->data ;
    P = P->next ;
}
```

So we use while loop

// 83 + 129

Creating a LL

```
#include <iostream>
using namespace std;
class node {
    public :
    int data ;
    node* next ;
};
void create (node *n) {
    // CREATE
     node *first = new node;
     node *second = new node;
     node *third = new node;
     first -> data = 10 ;
     first -> next=  second ;
     second-> data  = 11;
     second-> next  = third;
     third -> data  = 13;
     third -> next  = NULL;
     node *temp ;
     temp = first ;
     // traverse
     while (temp != NULL){
         cout<<  temp->data<<" ";
         temp = temp-> next ;
     }
}
int main(){
    node *head    ;
    create (head) ;

    return 0 ;
}
```

Creating

Creating

Traversing

Creating LL

```
#include <iostream>
using namespace std;
class node
{
public:
    int data;
    node *next;
};
node* create(int A[], int n)
{
    int i;
    node *t, *last;
    node *first = new node;
    first->data = A[0];
    first->next=NULL;
    last = first ;

    for ( i = 1; i < n; i++)
    {
        node* t = new node;
        t->data = A[i];
        t->next=NULL;
        last->next = t;
        last = t ;
    }
    return first;
}
void Display (node* p){
  while (p != NULL)
  {
     cout<< p->data << " " ;
     p= p->next ;
  }
}

int main()
{
```

```
#include <iostream>
using namespace std;
class node
{
public:
    int data;
    node *next;
};
void create(node** first,int A[], int n)
{
    int i;
    node *t, *last;
    *first = new node;
    (*first)->data = A[0];
    (*first)->next=NULL;
    last = *first ;

    for ( i = 1; i < n; i++)
    {
        node* t = new node;
        t->data = A[i];
        t->next=NULL;
        last->next = t;
        last = t ;
    }
}
void Display (node* p){
  while (p != NULL)
  {
     cout<< p->data << " " ;
     p= p->next ;
  }

}

int main()
{
    int A[] = {3, 5, 7, 10, 15};
    node* first;
    create(&first,A, 5);
    Display(first);
```

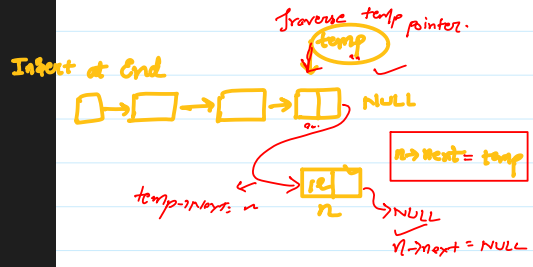
```
}
}

int main()
{
    int A[] = {3, 5, 7, 10, 15};
    node* first = create(A, 5);
    Display(first);
    return 0;
}
```

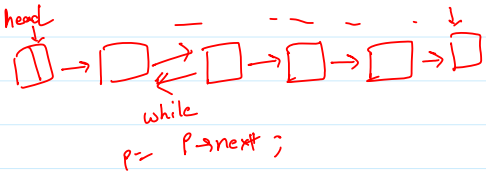
```
{
    int A[] = {3, 5, 7, 10, 15};
    node* first;
    create(&first,A, 5);
    Display(first);
    return 0;
}
```

Insertion.

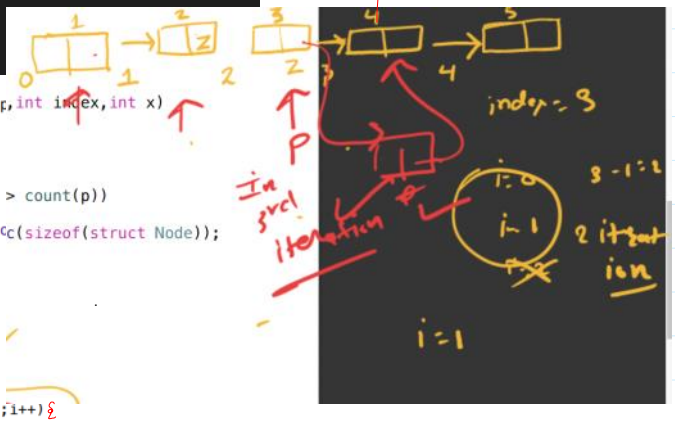
InsertAtFront

```
#include <iostream>
using namespace std;
class node
{
public:
    int data;
    node *next;
};
void create(node** p , int A[], int n)
{
    int i;
    node *t, *last;
    *p = new node;
    (*p)-> data = A[0];
    (*p) -> next=NULL;
    last = *p ;
    for ( i = 1; i < n; i++)
    {
        node* t = new node;
        t->data = A[i];
        t-> next=NULL;
        last->next = t;
        last = t ;
    }
}
void Display (node* p){
  while (p != NULL)

  {

      cout<< p-> data<<" ";
      p= p->next ;
  }
}
void insertathead (node **p , int val ){
        node *n = new node ;
        n->data = val ;
        n->next = (*p) ;
            (*p) = n ;
    }


void insertatend(node **t, int val)
{
    node *temp = *t;
    node *n = new node;
    n->data = val;
                // temp = *t ;
    n->next = NULL;
    while ((*t)->next != NULL)
    {
        (*t) = (*t)->next;
    }
    (*t)->next = n;
    *t = temp;
}


int main()
{
    int A[] = {3, 5, 7, 10, 15};
    node* first ;
    create(&first, A, 5);
    insertathead(&first,1) ;

    Display(first);
    return 0;
}
```



```cpp
#include <iostream>
using namespace std;
class node
{
public:
    int data;
    node *next;
    node(int val)
    {
        this->data = val;
       this->next = NULL;
    }
};
void create(node* &n)
{

     n = new node(1);
    node *second = new node(4);
    node *third = new node(8);
    n->next = second;
    second->next = third;
    third->next = NULL;
    node *temp;
    temp = n;
    int c = 0, sum = 0;
    int max = INT_MIN;
    while (temp != NULL)
    {
        cout << temp->data << " ";
        c++;
        if (max < temp->data)
        {
            max = temp->data;
        }

        sum = sum + temp->data;
        temp = temp->next;
    }
    cout << endl
         << c << endl
         << "sum LL = " << sum;
    cout << endl
         << "maximum = " << max;
}

void count(node *p ){

    node* temp = p ;
    int m = 0 ;

   // p=start ;
    while (temp != NULL){
        m++ ;
        temp = temp-> next ;
    }
    cout <<endl<<" m = "<< m ;
}
int main()
{
    node *head;
    create(head);
     count(head) ;

    return 0;
}
```




```cpp
void Insert(struct Node *p,int index,int x)
{
    struct Node *t;
    int i;

    if(index < 0 || index > count(p))
        return;
    t=(struct Node *)malloc(sizeof(struct Node));
    t->data=x;

    if(index == 0)
    {
        t->next=first;
        first=t;
    }
    else
    {
        for(i=0;i<index-1;i++)
            p=p->next;
        t->next=p->next;
        p->next=t;

    }

}
```

InsertAtPostionLL.cpp

```cpp
#include <iostream>
using namespace std;
class node{
public:
    int data;
    node *next;
    node(int val)
    {
        this->data = val;
        this->next = NULL;
    }
};
void create(node *&p, int A[], int n){
    p = new node(A[0]);
    p->next = NULL;
    node *t, *last;
    last = p;
    for (int i = 1; i < n; i++)
```

Create function

```
    p = new node(A[0]);
    p->next = NULL;
    node *t, *last;
    last = p;
    for (int i = 1; i < n; i++)
    {
        node *t = new node(A[i]);
        t->next = NULL;
        last->next = t;
        last = t;
    }
}
void insertatpos(node *&p, int index, int
val){
    node *t = new node(val);
    node *temp;
    temp = p;
    if (index < 0 || index > 5)
        return;
    if (index == 0){
        t->next = p;
        temp = t;
    }
    for (int i = 0; i < index - 1; i++){
        temp = temp->next;
    }
    t->next = temp->next;
    temp->next = t;
}
void display(node *p){
    while (p != NULL){
        cout << p->data << " ";
        p = p->next;
    }
    cout << "\n";
}
int main(){
    int A[] = {3, 4, 5, 6, 7};
    node *first;
    create(first, A, 5);
    display(first);
    insertatpos(first, 3, 11);
    display(first);
    return 0;
}
```

Create function

Insert at pos function

Display function

Searching

LINEAR SEARCH

Function For Searching

```
// Searching
node* search(node* p , int key){

    while(p != NULL){
        if(key==p->data) {
            return (p);
        }
        p=p->next ;
    }
    return NULL;
}
// RECURSIVE SEARCHING
node* Rsearch(node* p , int key){
    if(p==NULL)
    return NULL ;
    if(key==p->data)
    return (p) ;

    return search(p->next, key ); <--Same-->
}
int main(){
    int A[] = { 11, 22, 13, 54, 50 };
    node *head ;
    create(head, A, 5);
    cout<<"ADDRESS = "<<search(head, 54);
    cout<<endl<<"ADDRESS = "<<Rsearch(head, 54);
```

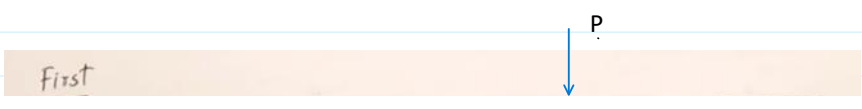
Improving linear search (found in less time)

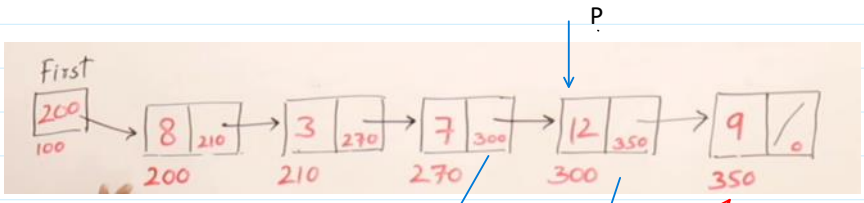
i) Transposition
It is a method where we change the value to previous value

ii) Move to Head
It means the key have broad in the beginning.

- Here transposition is not preferable because in linked list movement of data is not preferable. Movement of node is preferred.
So that's why move to head is preferable.

```
struct Node * LSearch(struct Node *p,int key)
{

    while(p!=NULL)
    {
        if(key==p->data)
            return p;
        p=p->next;
    }
    return NULL;

}
```





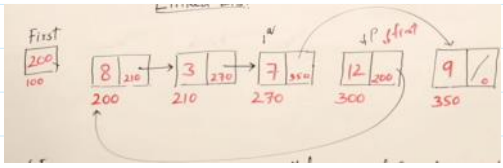
Lets KEY is 12 whose address is 300

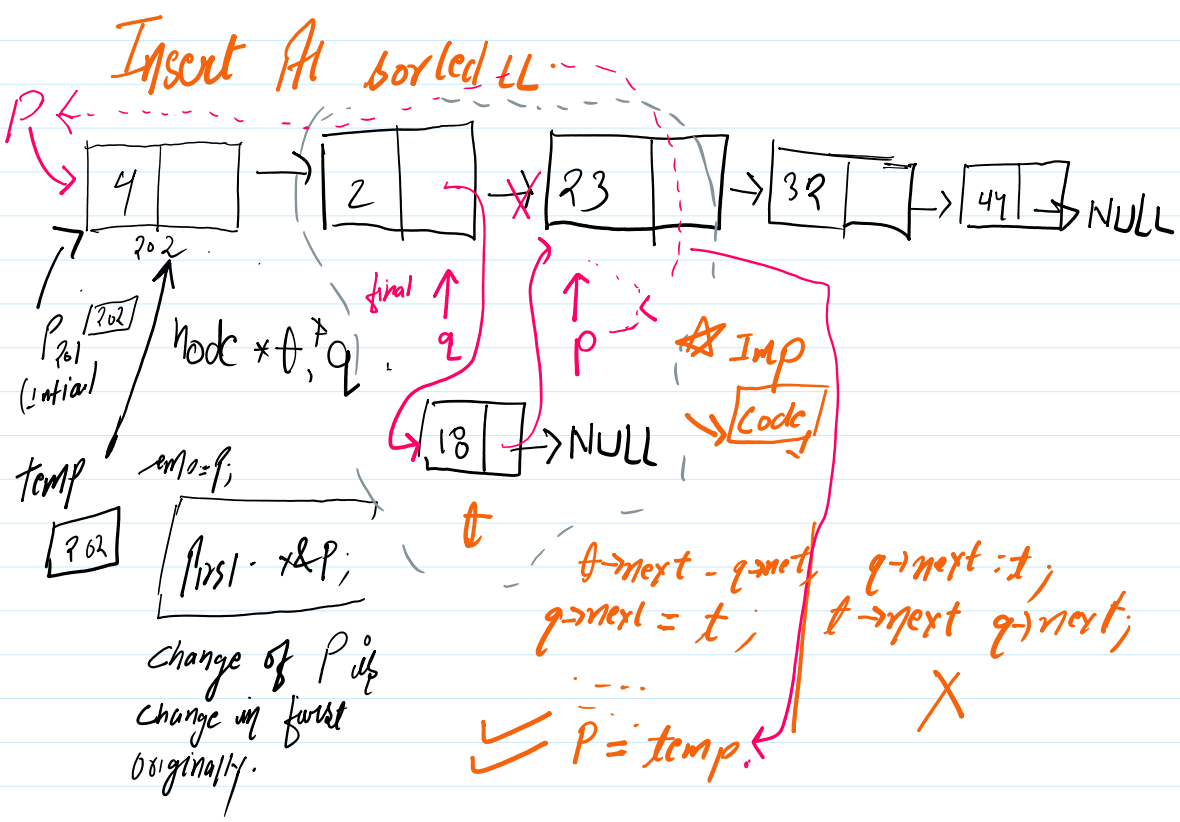
We have to bring this node in beginning.

We have to modify this previous node also.
and linked to this node_____.

According to this , we need pointer at previous node.
How to get pointer in previous node ?
Ans) Let make node pointer.

```
Node* p, *q;  //p is at first node and q follow p .
Node* temp;
Temp = p;
While(Till p traverse to NULL){
    If(key == p->>data) {
        q->next = p->next;
        P->next = first;
        first = p;
    }
      q=p;
    P=p->next;
}
```



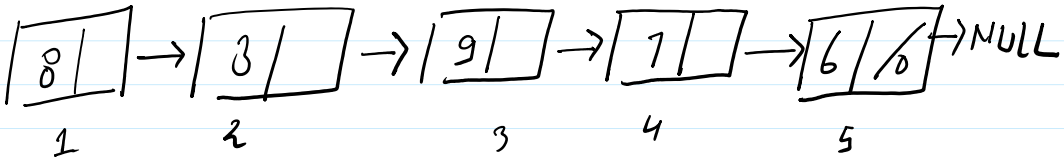


Coding

```
#include<iostream>
using namespace std;
class node{
```

```
    public :
    int data ;
    node* next ;
    node(int val ){
       this -> data = val ;
        this->next = NULL ;
    }
};
void create(node* &p ,int a[], int n){
    p = new node(a[0]);
    p->next =NULL ;
    node *t, *lst ;
    lst = p ;
    for (int i = 1; i < n; i++){
        t = new node(a[i]) ;
        t->next = NULL ;
        lst->next = t ;
        lst = t;
    }
}
void Display(node *p){
    while (p != NULL){
        cout<<p->data<<" " ;
        p=p->next;

    }
}
void insertinsortedLL(node *&p , int key  ){ // 4  12  23  32  44  // 18 ->4 12 18
        node *t , *q ;                        //    q   p
        t = new node (key) ;
        t->next = NULL ;
        node* temp ;
        temp = p ;
        if(p==NULL){
            p=t ;
        }
        else{
            while (p && p->data < key )
            {
                q=p;
                p=p->next ;
            }
            if(p==temp){
                t->next = p ;
                p = t ;
            }
            else{
                t->next = q -> next ;
                q->next = t ;
            }
      p = temp ;
        }
}
int main(){
    int arr[] = { 4,12, 23,32 ,44 } ;
    node* first ;
    create(first , arr , 5) ;
    Display(first);
    cout<<"\n";
    insertinsortedLL(first, 18);
    Display(first);
    return 0 ;
}
```

DELETION "LL"



* when we allocate node in linked list or occupy space in memory

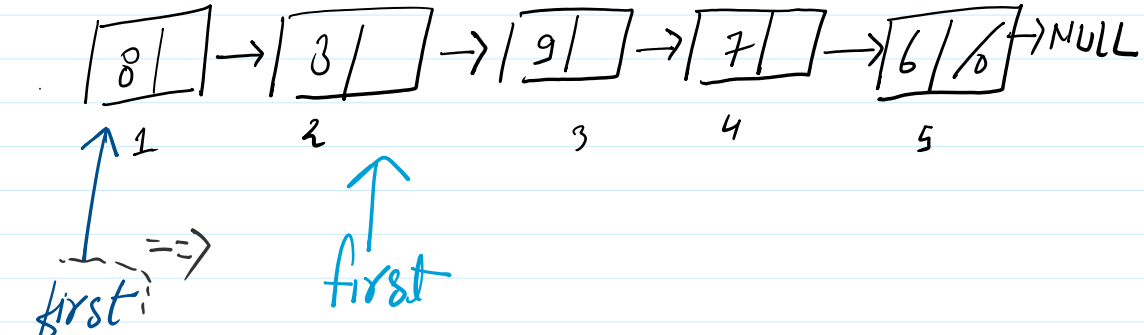
* When we allocate node in linked list or occupy space in memory so after doing Deletion of node it necessary to deallocate the that node.

Deletion Type :-

i) Delete first node

ii) Delete node at given position.

⇒ Delete 1st node.



---> Remove this first & jump into (first->data = 3) 2nd adjacent node.

After this.

We need ~~free/Deallocate~~ delete initial first pointing

Code ⇒

```
node; // first->data
void delete(node *p)
    node * temp;
    temp = p;
// delete first node
    if(p == temp){
      p = p->next;
      return;
```

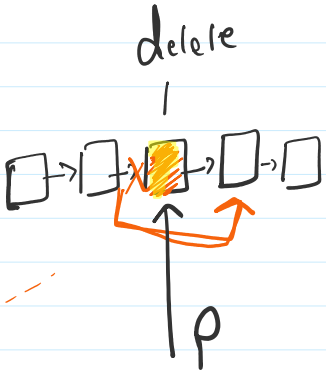
while(p->data == key)

```
    p = p->next;
    return;
  }
  else {
```

⇒ Delete node at given position.

```
void delete (node *p)
  node *temp;
  temp = p;
while (p->data == key) {
  p = p->next;
  if (p == temp) {
    p = p->next
    delete (temp);
  }
  else {
    Node *t;
    t = p->next;
    p->next = (p->next)->next;
    delete (p);
  }
  p = temp;
}
```

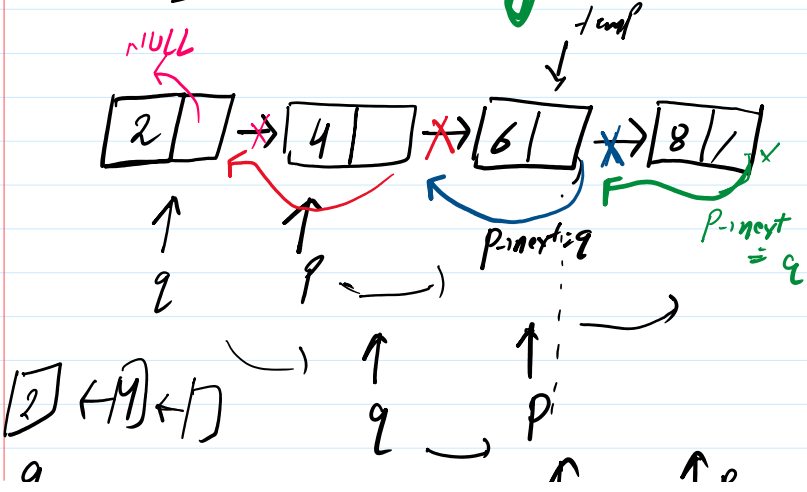
// Here we only delete node pointing by temp & shift p position.

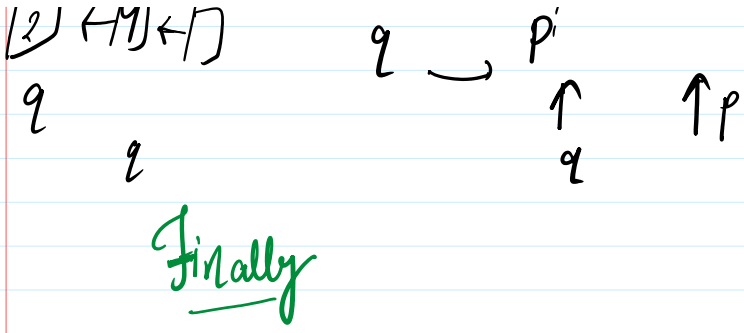


Remove Duplicates

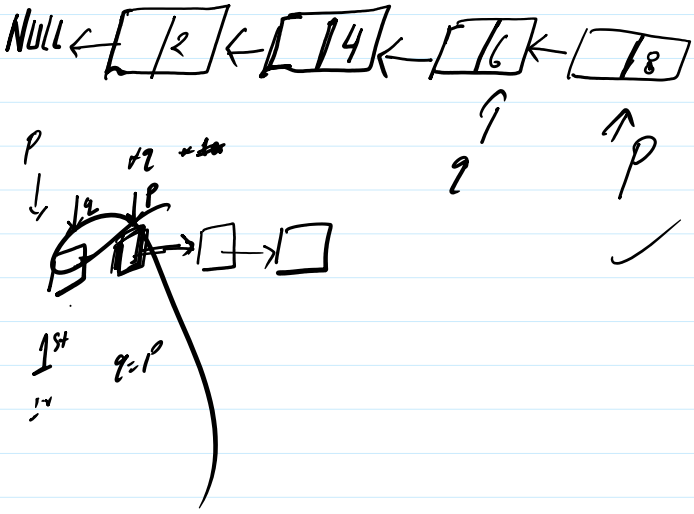
```
Node *p=first;
Node *q=first->next;

if(p->data!=q->data)
{
    p=q;
    q=q->next;
}
else
{
    p->next=q->next;
    delete q;
    q=p->next;
}
```

Reversing

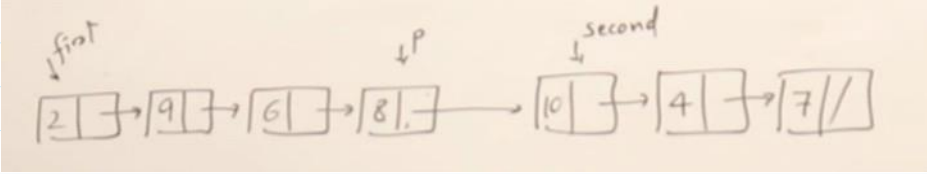




Finally

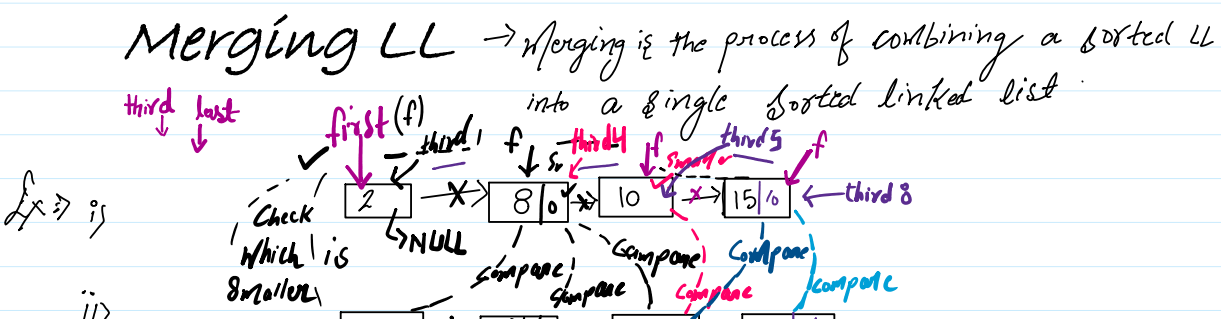


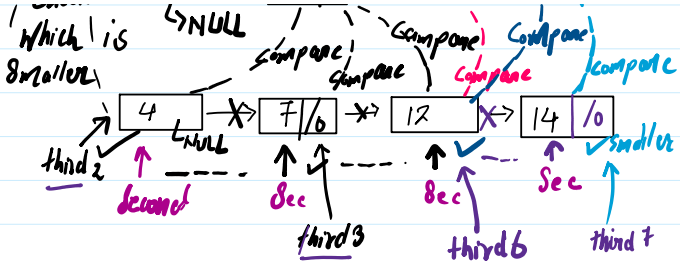
Concatenating LL

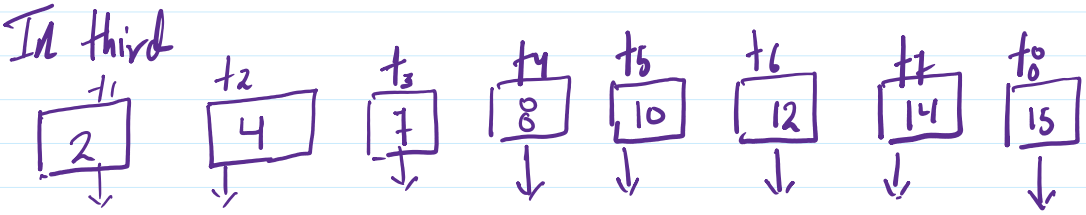


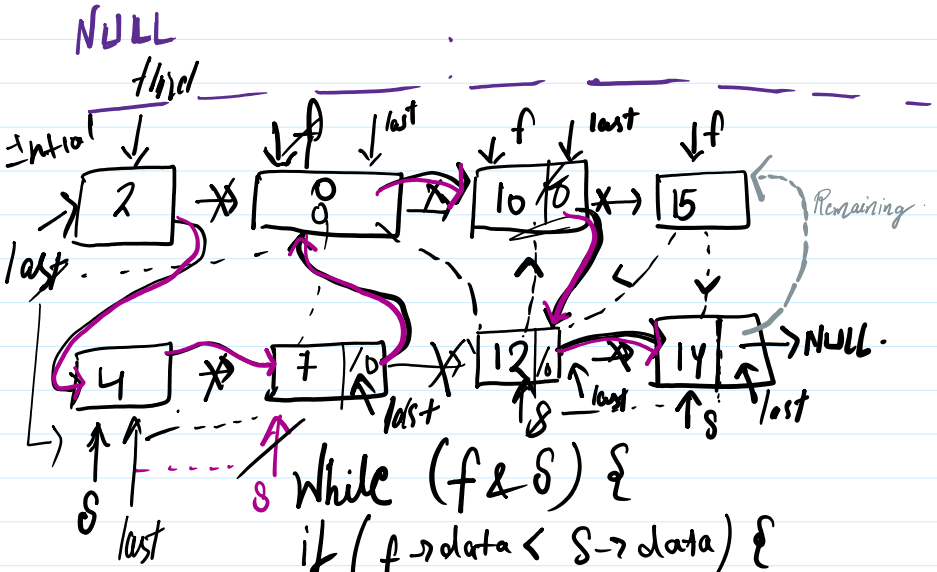

```
p = first;
while (p -> next != NULL)
{
    p = p -> next;
}
p -> next = second;
second = NULL;
```

```cpp
#include<iostream>
using namespace std ;
class node {
    public :
    int data ;
    node* next ;
    node(int val ){
        this->data = val ;
        this->next = NULL ;
    }
};
void create (node* &p , int a[],  int n){
     p = new node(a[0]);
     p->next = NULL ;
     node *last ;
     last  = p ;
     for (int i = 1; i <n ; i++)
     {
         node* t = new node(a[i]);
         t->next = NULL ;
         p->next = t ;
         p = t ;
     }
     p=last ;
}
void create2(node* &p , int a[] , int n ){
    p= new node(a[0]);
    node* last ;
    p->next = NULL ;
    last = p ;
    for (int i = 1; i < n; i++)
    {
        node *t = new node(a[i]);
        t->next = NULL ;
        p->next = t ;
        p=t ;
    }
     p= last ;
}
void display(node *p){
    while(p!=NULL){
        cout<<p->data <<" ";
        p=p->next ;
    }
}
void concatenate(node* &fis , node* &sec ) { // done
    node* temp ;
    temp = fis ;
        while(fis->next!=NULL){
            fis = fis->next ;
        }
        fis->next = sec ;
         fis = temp  ;
}
int main (){
    int A[] ={50,45,34 ,3, 2} ;
    int B[] = {33,4 ,56, 4 };
    node * head1  ;
    node* head2 ;
    create(head1 , A ,  5) ;
    create2(head2 , B , 4) ;
    concatenate(head1 , head2) ; // DONE
    display(head1);
    cout<<"\n" ;
    //display(head2);
    return 0 ;
}
```










```
While (f && s) {
    if (f->data < s->data) {
        last->next = f;
        last = f;
        f = f->next;
        last->next = NULL;
    }
    else {
        last->next = s;
        last = s;
        s = s->next;
        last->next = NULL;
    }
    if (f != NULL) {
        last->next = f; }
    else { last->next = s; }
```

Time Complexity in merging
Let m ≤ n
$\theta(m+n)$

Function Merge.

```
void merge( node* &fis , node* &sec , node* &thir){
         node *temp , *last ;
         temp = fis ;
        //  int c ;
          // Intialization ;
            if(fis->data < sec-> data){
                thir = last = fis ;
                fis = fis->next ;
                last->next = NULL ;
            }
            else{
                thir = last = sec ;
                sec = sec->next ;
                last -> next = NULL ;
            }
        while(fis && sec ){
            if (fis->data < sec->data){
                last->next = fis;
                last = fis ;
                fis = fis ->next ;
                last->next = NULL ;
            }
            else {
                last -> next =  sec ;
                last = sec ;
                sec=sec->next ;
                last ->next= NULL ;
            }
        }
        if(fis!=NULL){
        last->next=fis;
        }
        else {
        last->next= sec;
        }
}
```

```
merge(head1, head2, third ) ; // DONE
display(third);
```

Calling in int Main.