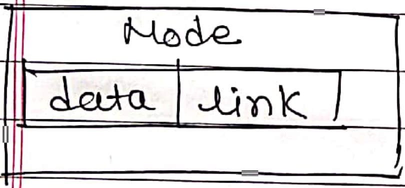
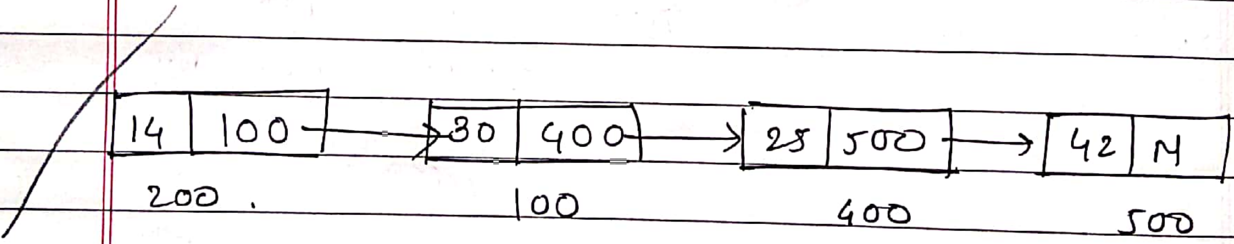
Unit IV

Linked list

linked list is a very common data structure often used to store similar data in memory While the elements of an array occupy contiguous memory locations. Those of the linked list are not consitrained to be stored in adjacent location. The individual elements are stored "Somewhere" in memory, rather like a family dispresed, but still bound together. The order of the elements is maintained by explicit links between them.



N stands for NULL



Linked list is the collection of nodes. each of which stores two items of information

— an element of the list
— and an link.

A link is a pointer or an address that indicates explicitly the location of the node containing the successor of the list element.

The data part of each node contain

The element & the link part is a pointer to the next node. The NULL in the last node indicates that this is the last node in the list.

Operation on linked list :-

There are several operations that can performed on linked list. These are

- Adding a new node at begining
- Adding a new node at end.
- Adding a new node at middle

It also performed dispeay the linked list operation & & delete the node.

Node structure

Structure is used to represent a node. The structure node contains 'data &' link' part. The variable 'P' has declared as a pointer to the 1st node in linked list. When (P== NULL) it means there is no node in linked list.

```
struct node
{ int data;
  node*link ;
}*p ;
```

Append :-

The append fuction deals with two situations these are

1) The node is being added to an empty list
2) The node is being added at the end of the existing list.

1) if (P = = NULL)

In the 1st case the above condition is check & if it is true then it's mean there is no node in the link list.
So 1st we allocate space for new node using 'new' operator.

```
temp = new node;
temp → data = num;
temp → link = NULL;
```

Lastly P is point to this node, since the 1st node added to the list

2) IP (P==NULL) is false i.e in secondcase
in this case 'temp' is made to point to the 1st node

```
temp = P;
```

Then using 'temp' we can traversed through the entire linklist using the statement.

```
while( tem → link != NULL)
    temp = temp → link;
```

when 'temp'' reaches at the end of the linked list. Once outside the loop a new node is formed and space is allocated for this.

```
r = new node ;
r → data = num ;
r → link = NULL ;
temp → link = r ;
```

3) Now adding a new node at the middle of linked list.
For this we use a for loop to reach to the specified location.

```
for( i=0 ; i<loc ; i++)
{ temp = temp → link
      if (temp == NULL)
      { cout<<"less element
                than loc" ;
        return;
      }
}
```

By using this we can reached to the specified location in the linked list now allocate the space for new node

```
r = new node ;
r → data = num ;
r → link = temp → link ;
temp → link = r ;
```

By using above code we add the new node 'E' at the specified location.

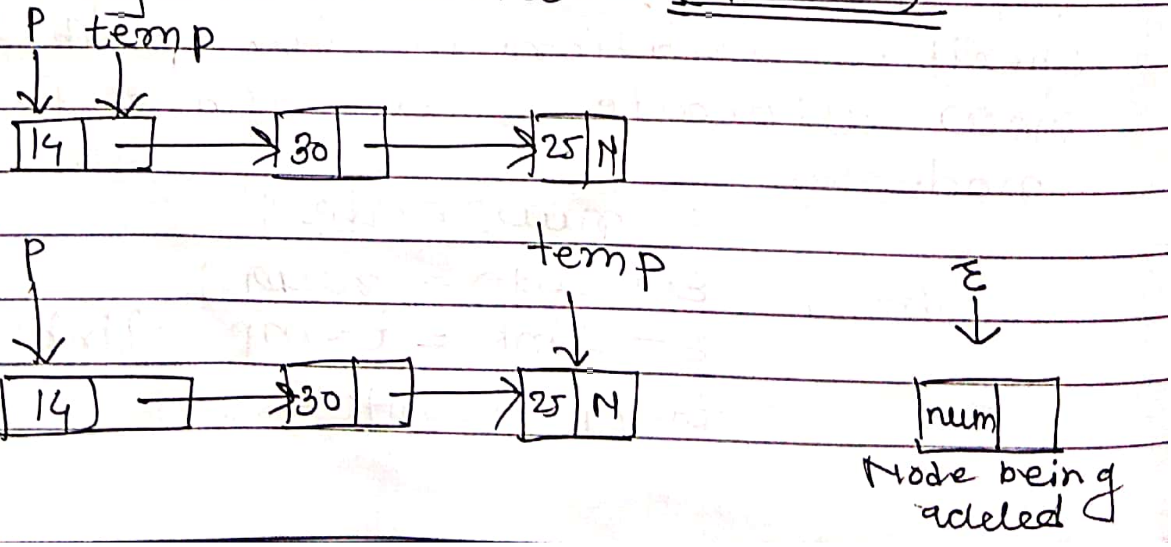
Display linked list :-

We can traveler the link list from beginning to end using the 'temp' pointer & print the 'data' part in the linked list. For that we use while loop.

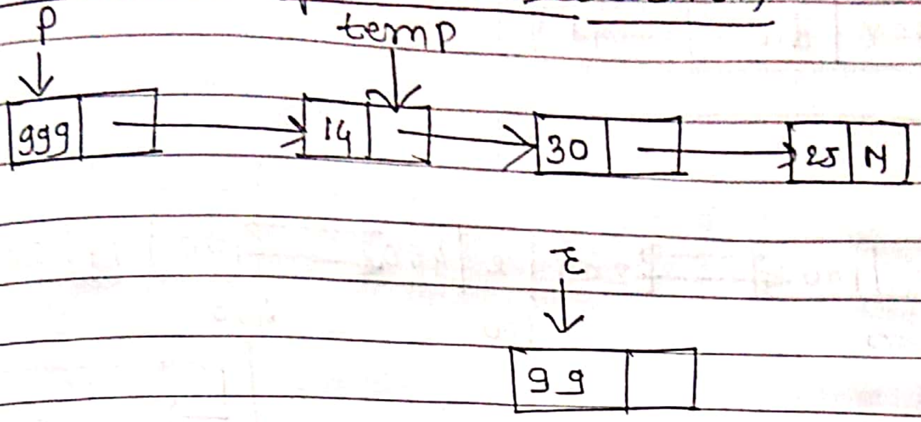
```
    node *temp = p
          while (temp != NULL)
          
             { cout << temp → data;
               temp = temp → link;
             }
```

These are operations performed on singly linked list.

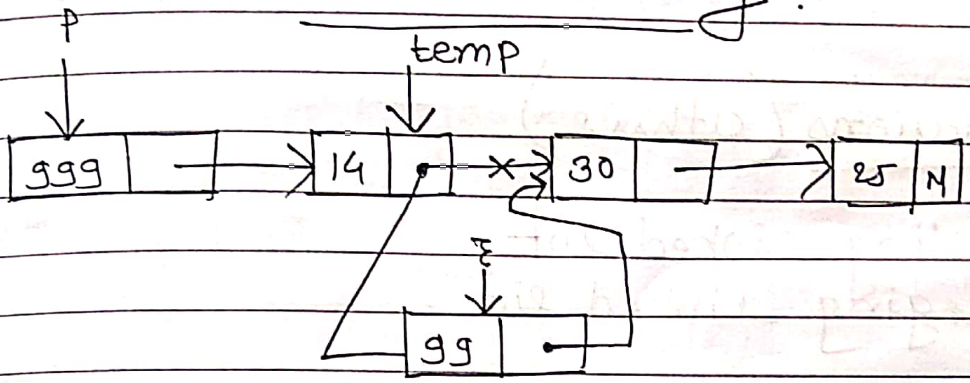
1) — adding data at end (append)



2) Add after specified location



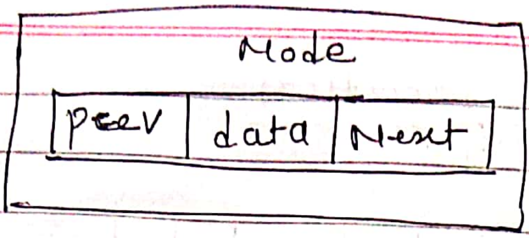
Before inserting.

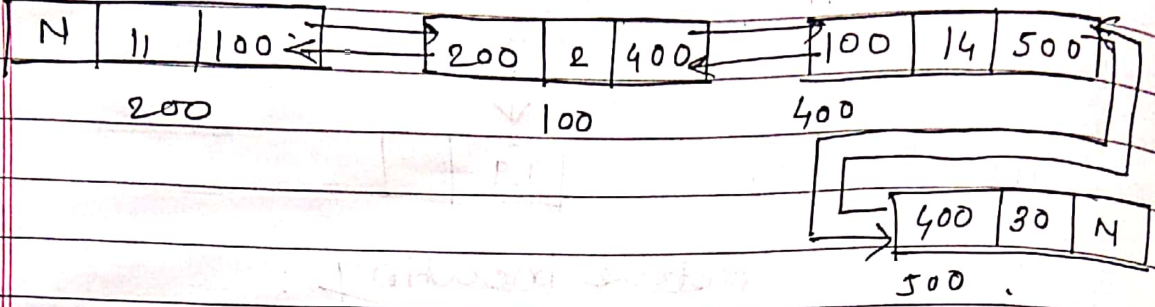


after inserting.

Doubly linked list :-

In singly linked list, the list provides the address of the next node only. It does not gives us the previous node location to overcome this in Doubly linked list the node can stand the location of next & previous node The node can be represented as follows -



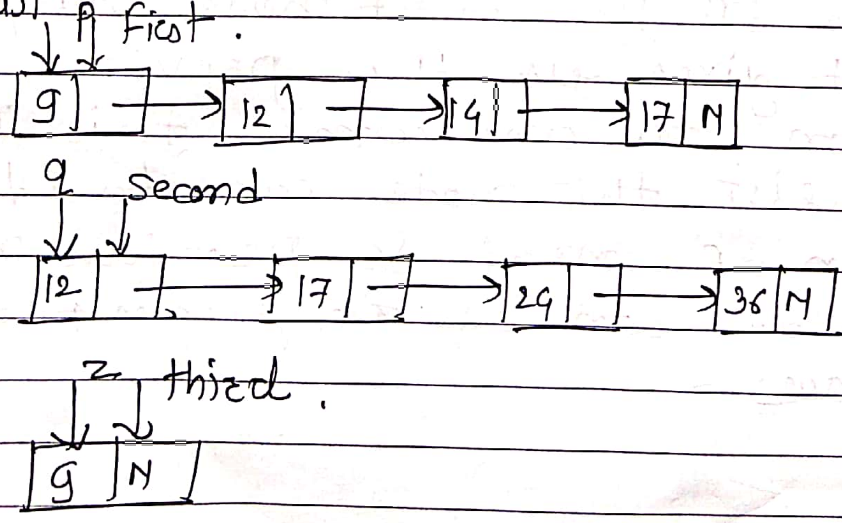


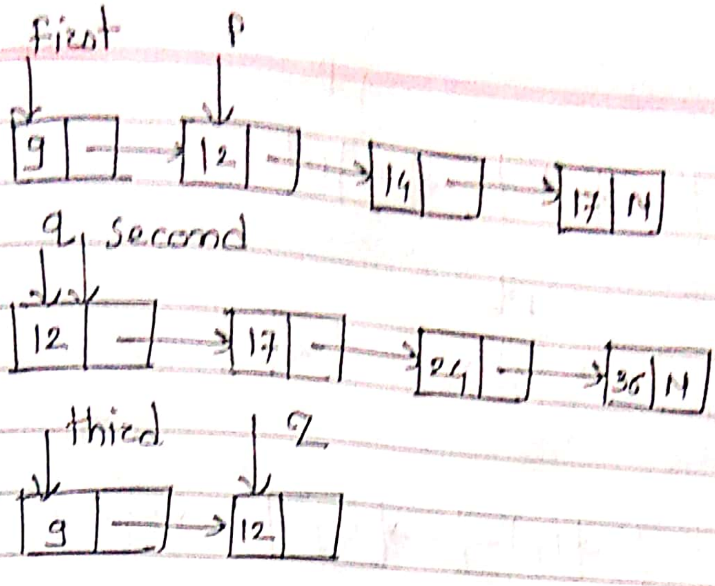
Operations (Others)

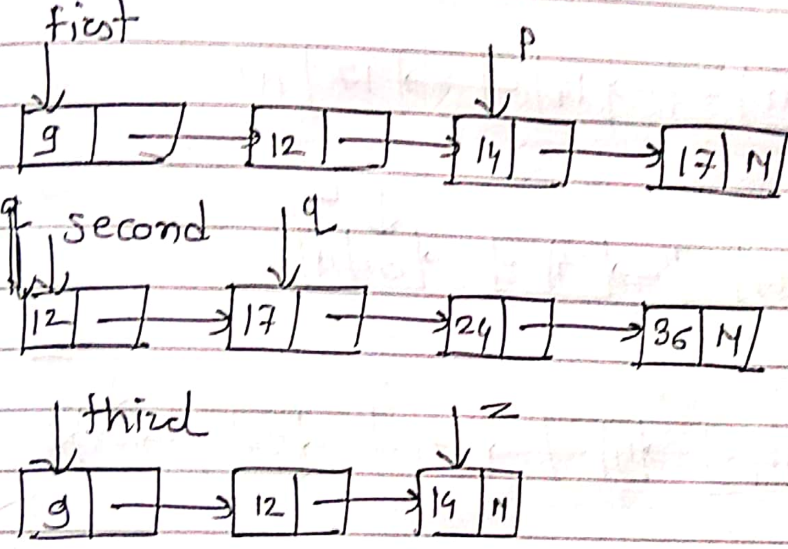
- Sorting linked list
- Merging linked list.

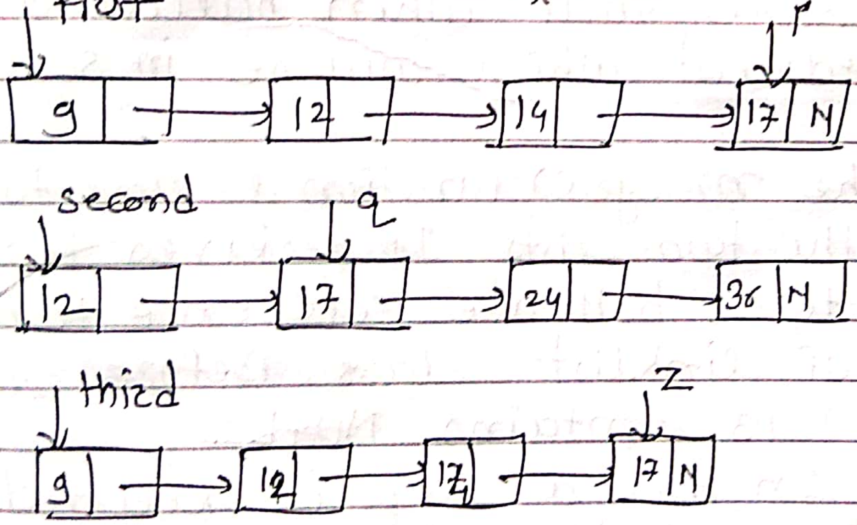
Merging linked list :-

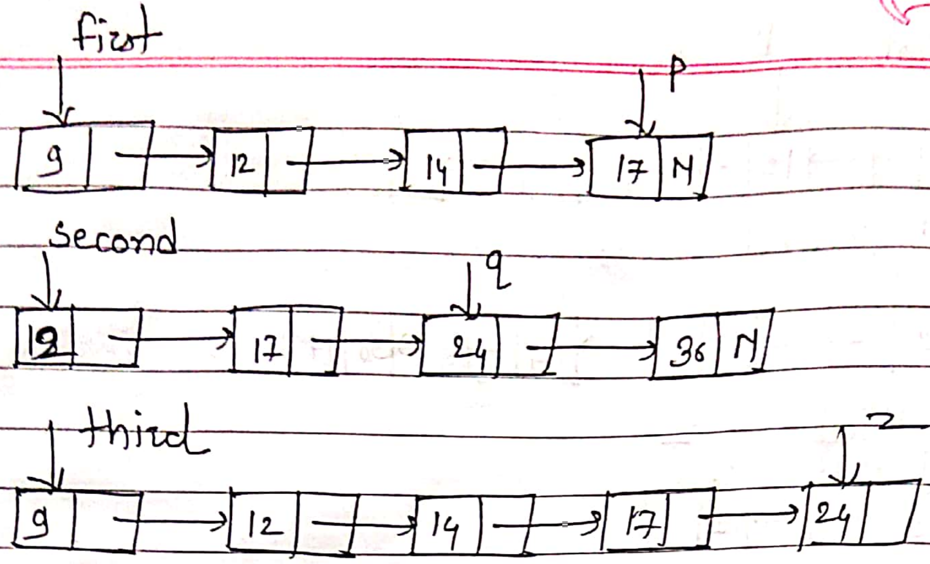
for merging linked list we required two lists which should be in accending order. So the final third list is required to merge the contentes of these two list

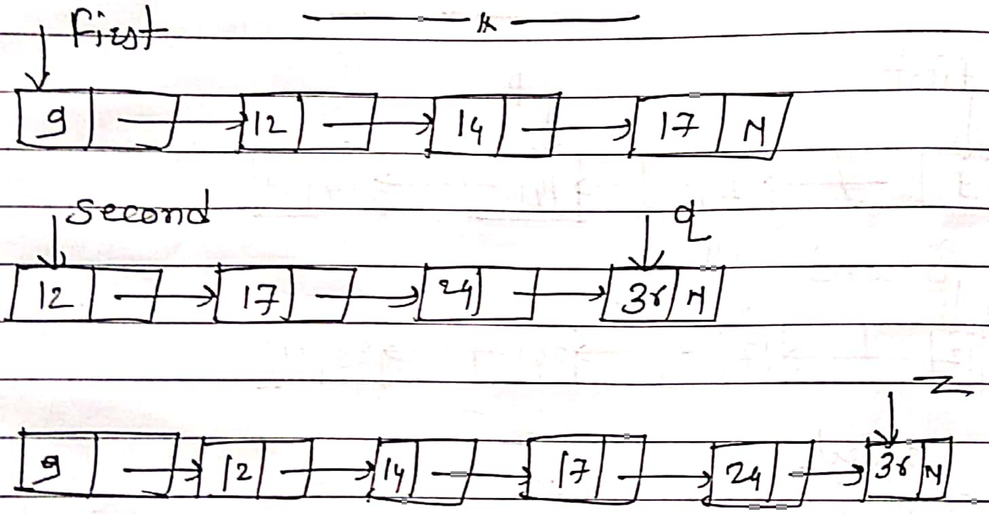



first
p
9
12
14
17
N
q
second
12
17
24
36
N
third
z
9
12


first
p
9
12
14
17
N
q
second
q
12
17
24
36
N
third
z
9
12
14
N


first
p
9
12
14
17
N
second
q
12
17
24
36
N
third
z
9
12
14
17
N





We use the add function which will make sure that when building the third list which will be in sorted order.

The merge() function is used to merge the two lists. It receives two parameters, both are reference to the object of linklist class. Before calling merge() l3 contains NULL.

In this a loop is executed to traverse the lists that are pointed by l1.p && l2.p. If end of any of the list is reached then the loop is terminated.

Data from both the list are compared & whichever is found to be smaller is stored in the data part of the 1st node of the merge list. The pointers that point to the merge list and to the list from where we copied the data are incremented appropriatly.

While compairing the data, if we find that the data of both the lists are equal then the data is added ony once to the merge list and pointer of l1 and l2 are incremented. this is done through the statements

```
if (l1.p→data == l2.p→data)
{
        z→data = l2.p→data;
        l1.p = l1.p→link;
        l2.p = l2.p→link;

}
```

If we reached end of the 1st list or second list the while loop is terminated. If we reach end of ony one list then the remaining element of the second list are dumped in the merged list as they are already in ascending order.