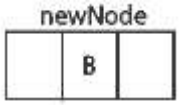
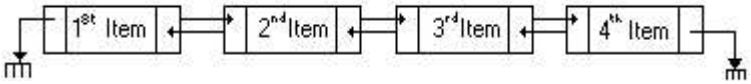
Lecture 8

This lecture is about Double Linked List. We will talk about Insert and delete operation with regard to double linked list. Let us first start with definition of double linked list.

Definition:
A double linked list is defined as a collection of nodes in which each nodes has three parts: data, llink, rlink. Data contains the data value for the node, llink contains the address of node before it and rnode contains the address of node after it.



This is how a doubly linked list looks like:



Note that
1. Each node point to previous node and next node.
2. Since there is no node before first node, so first node's llink contains null.
3. Similarly, for the last node. There is no node after last node, hence last node's rlink contains null.

Here is the data type of node for Double Linked List

```
typedef stuct node
{
        int data;
        node *llink;
        node *rlink;
}
```

node * temp; // temp is a pointer of type node.

If temp is a pointer of type node, then
1. To refer data part : temp->data
2. To refer llink : temp->llink
3. To refer rlink : temp->rlink

Next, we will do 4 things
1. Build a double linked list
2. Print a double linked list.

3. Insert a node in a double link list
4. Delete a node from a double link list

Building a double Linked List

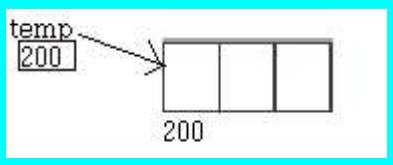
Login behind building a double linked list is similar to building a single linked list. Only difference is, in single link list, each node contains the address of node after it, whereas in double link list, each node contains the address of node after it and before it.

Step1: Initialize ans, head, temp, temp1 to null

```
node *head = null;
node *temp = null;
node *temp1 = null;
char ans='y';
```

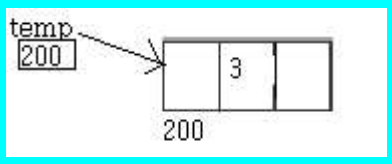
Step2: Create a node, call that node temp or store the addres of that node in temp.

```
temp =(node*) malloc(sizeOf(node));
```



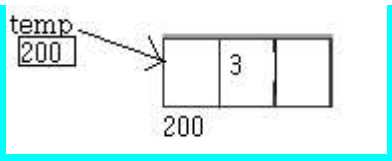
Step3: Ask the user to enter data value for the node.

```
printf("Enter value");
scanf("%d",temp->data);
```



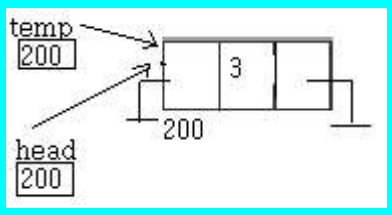
Step4: Since this is the first and last node, hence there is no node before and after it. Therefore, store null in llink and rlink.

```
temp->llink = null;
temp->rlink = null;
```



Step5: Store the address of first node in head, because head always contain the address of first node.

head=temp;

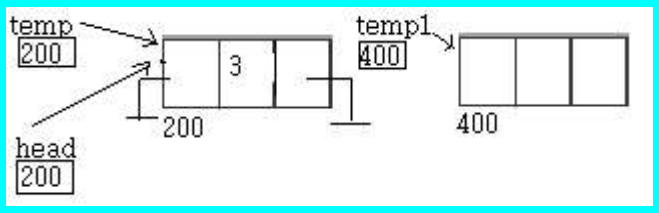


Step6: Start a loop, which repeats itself as long as condition ans=='y' remains true

while (ans == 'y') {

Step7: create a node and store the addres of this node in temp1.

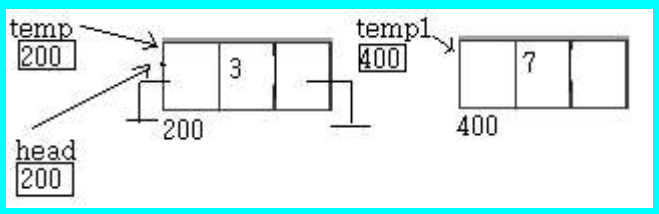
temp1= (node *)malloc(sizeOf(node));



Step8: Ask the user to enter data value in temp1->data

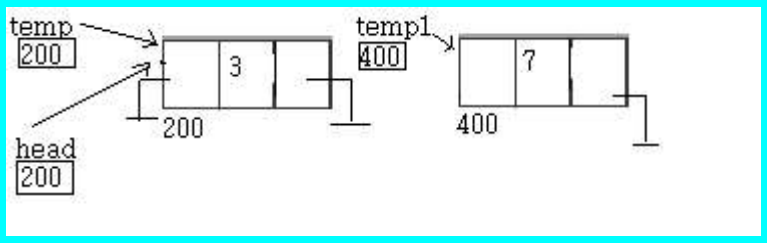
printf("Enter value");

scanf("%d",temp1->data);



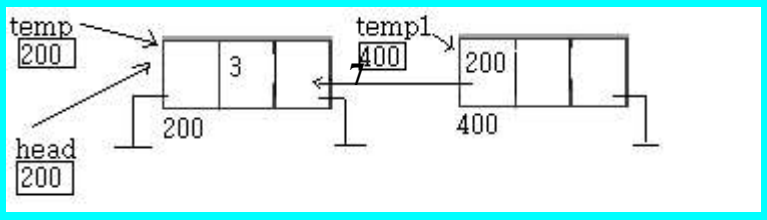
Step9: We want temp1 to be the second node and also the last node.
If temp1 is going to be last node, this means, tem1->rlink should contain null.

temp1->rlink = null;



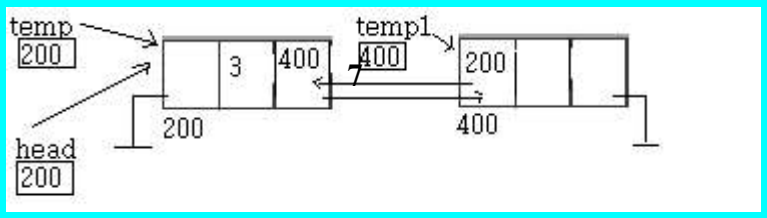
Also node before temp1 is temp, hence temp1->llink should contain the address of temp.

temp1->llink = temp;



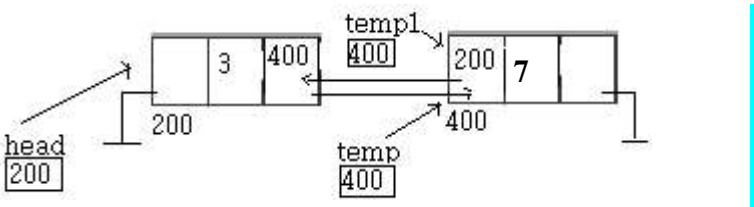
Step10: Still we are not done, because temp->rlink is still null. Now, temp->rlink should contain the address of temp1.

temp->rlink = temp1;



Step11: Move temp to the last node, i.e. Store address of temp1 in temp.

temp = temp1;



Step12: Ask the user if he wish to add more node.

```
        printf("Do you wish to continue");
        scanf("%c",&ans);
}
```

Print a doubly linked list

Also, printing a doubly linked list is similar to printing a single linked list. This is your assignment.

Insertion in a Double linked list

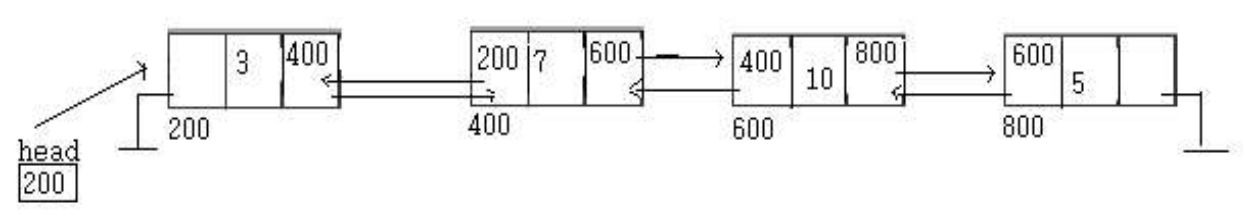
Like we did insertion in single linked list, same we will do here. We will take three cases

1. Insert a node in the beginning
2. Insert a node in the end
3. Insert a node in between

Let us start with the first case.

Insert a node in the beginning

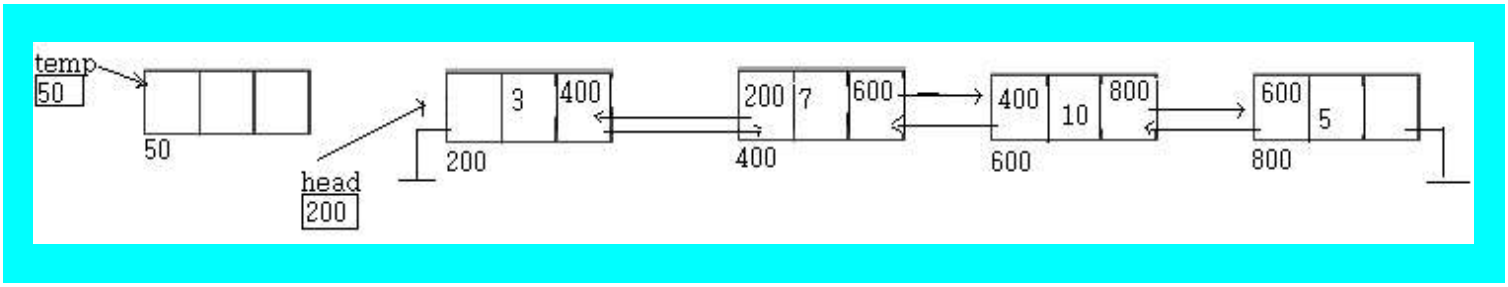
We assume that we have a double linked list, with head point to the first node.



If we want to insert a node in the beginning, that means, we want to insert before first node .

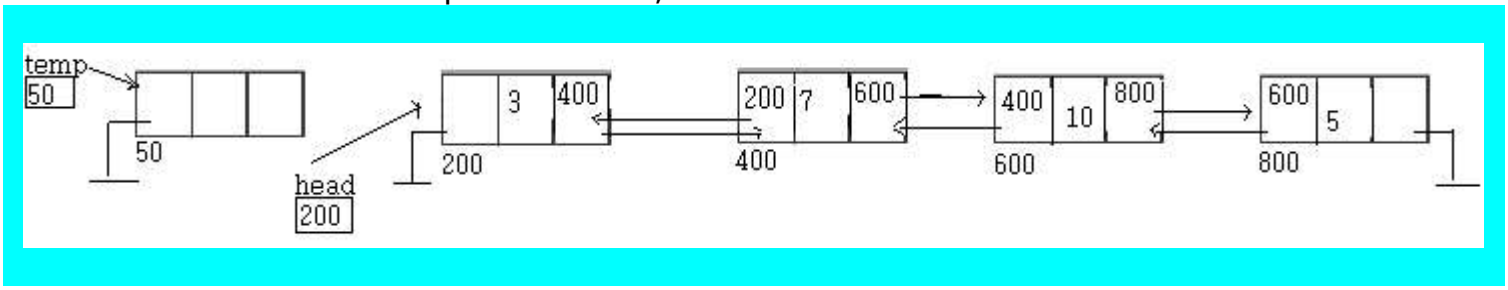
Step 1: Create a new node, store the address of this node in temp.

```
temp = (node *)malloc(sizeOf(node));
```



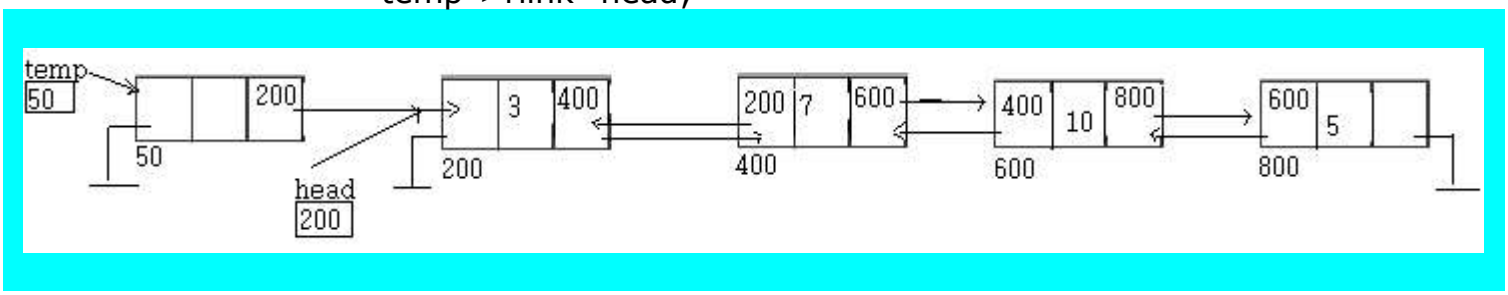
Step2:Since temp is going to be first node, that means, there is no node before temp. Hence store null in temp->llink.

temp->llink=null;



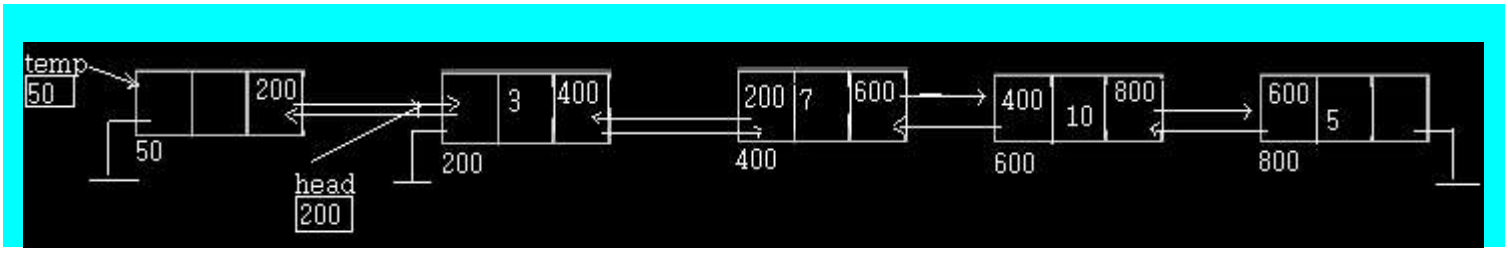
Step3: If temp is first node, then temp->rlink should store the address of current first node (which is head).

temp->rlink=head;



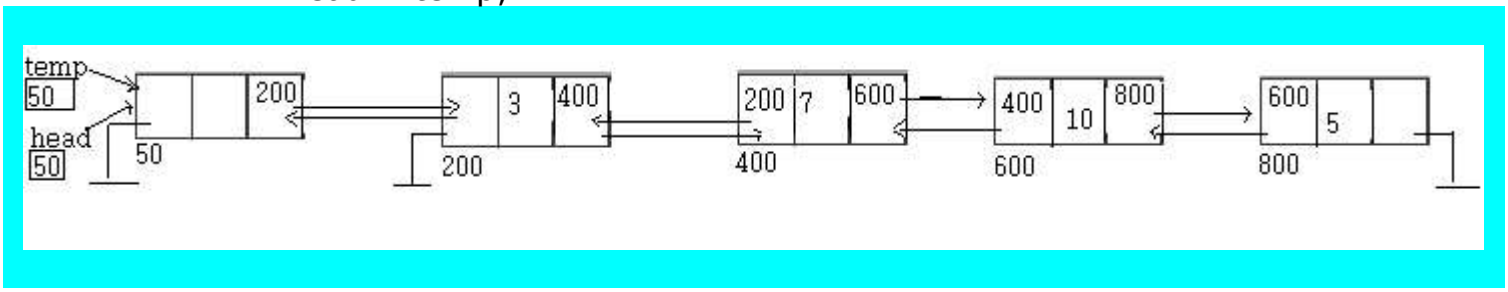
Step4: Make current first node's llink (i.e. head->llink) store the address of temp.

head->llink = temp;



Step5: This is the last step, as head should always point to the first node, so change the address stored in head. Head currently stores 200, it should store 50.

head = temp;

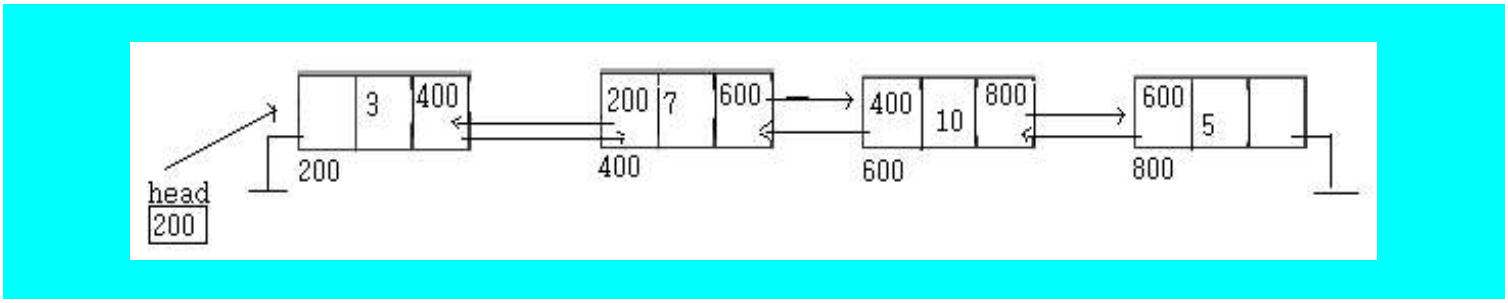


So, we are finished with insertion in the beginning.

Next topic is insertion in the end.

Insert a node in the end

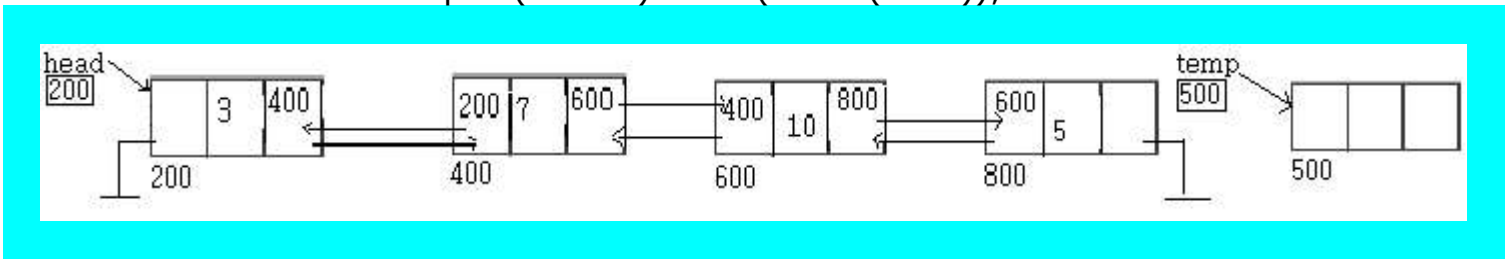
We assume that we have a double linked list, with head pointing to the first node.



Insert a node in the end means, to insert a node after the last node.

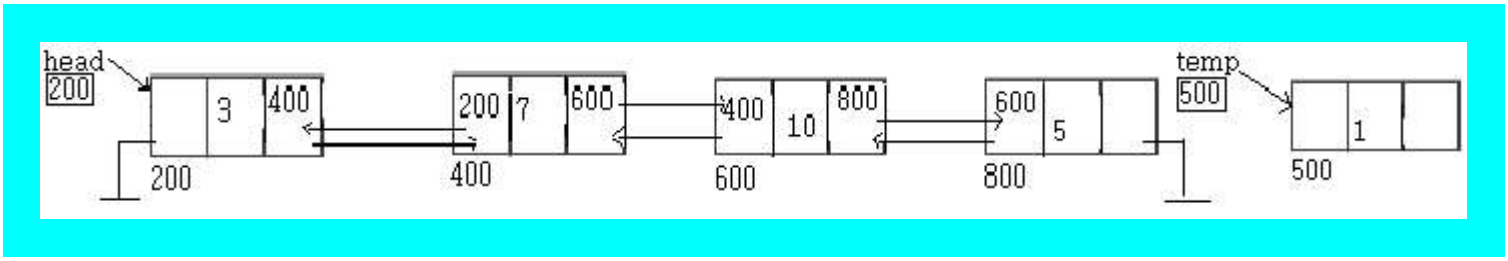
Step1: Create a new node, store the address of this node in temp.

temp = (node *)malloc(sizeOf(node));



Step2: Ask the user to enter value in temp->data.

printf("Enter data");
scanf("%d",temp->data);

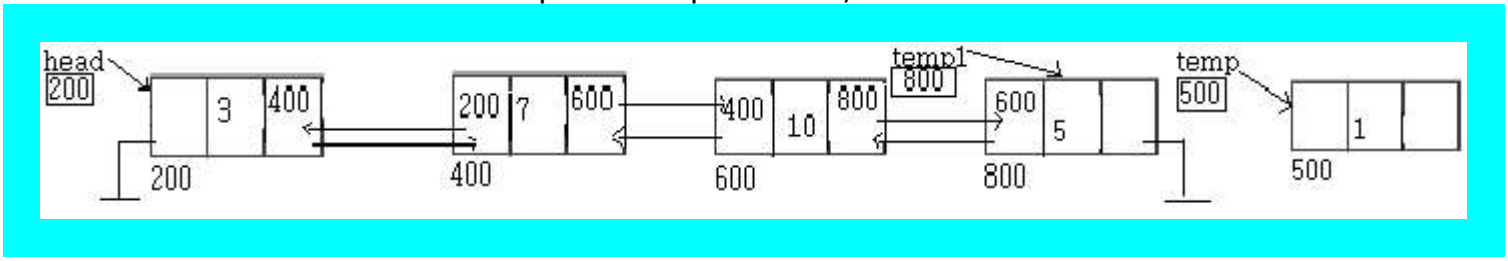


Step2: We need a variable that points to the last node, or we need temp1 point to the last node. For that, we will first store address of first node in temp1.

```
temp1 = head;
```

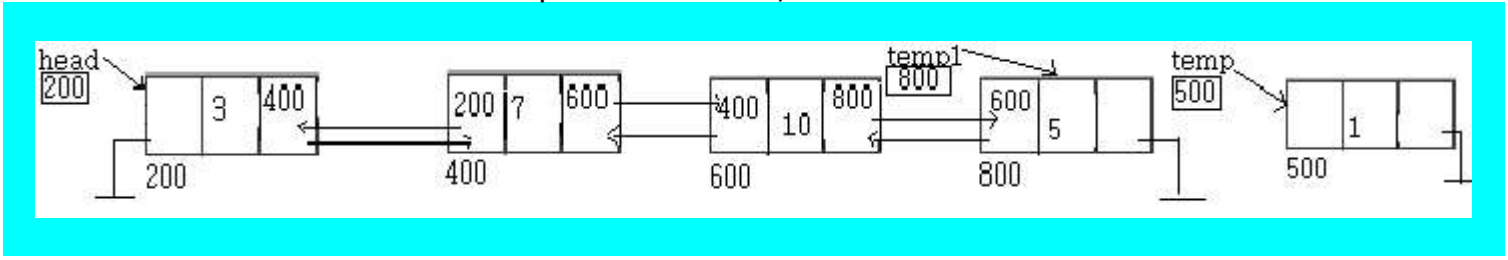
Step3: Now move temp1 to the last node.

```
while (temp1->rlink != null)
    temp1 = temp1->rlink;
```



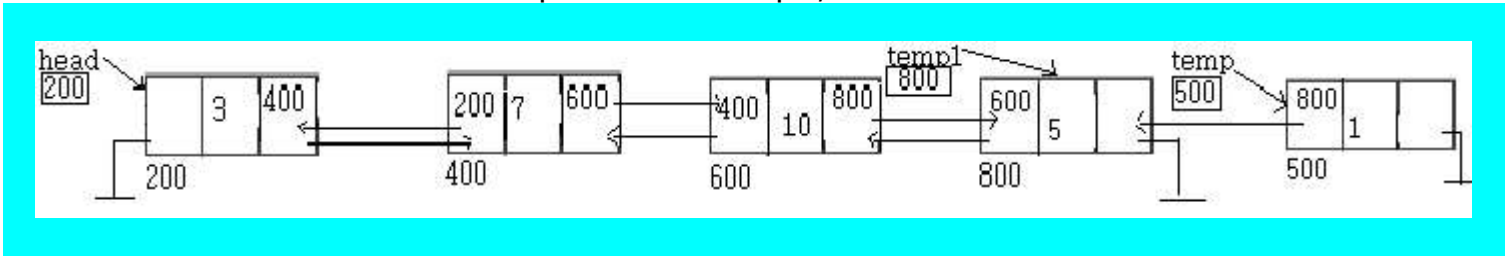
Step4: We want to make temp as the last node, that means temp->rlink should contain null.

```
temp->rlink = null;
```



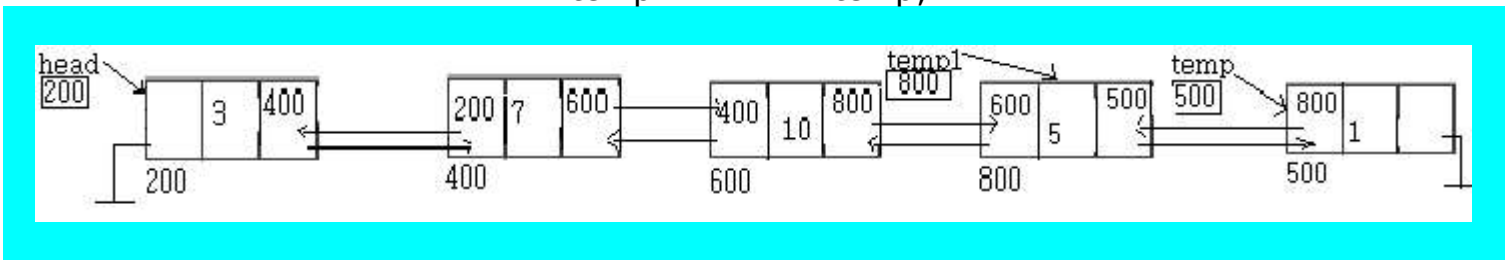
Step5: because temp is going to be last node, so temp->llink should store the address of temp1.

```
temp->llink = temp1;
```



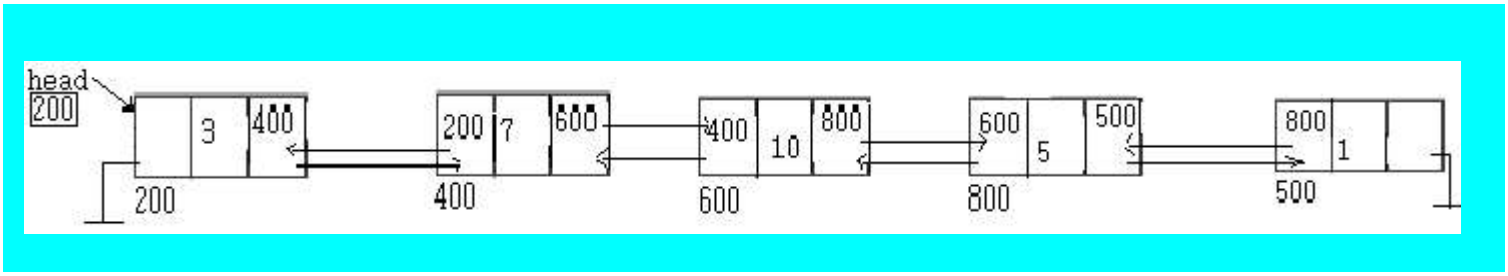
Step6: This is the last step, now temp1->rlink should store the address of temp.

temp1->rlink = temp;



Insert a node in between

This is last case of insertion.
We assume that we have a double linked list with head pointing to the last node.



To insert a node in between, we need to ask the user after which node he wish to insert.
Infact now there are two cases

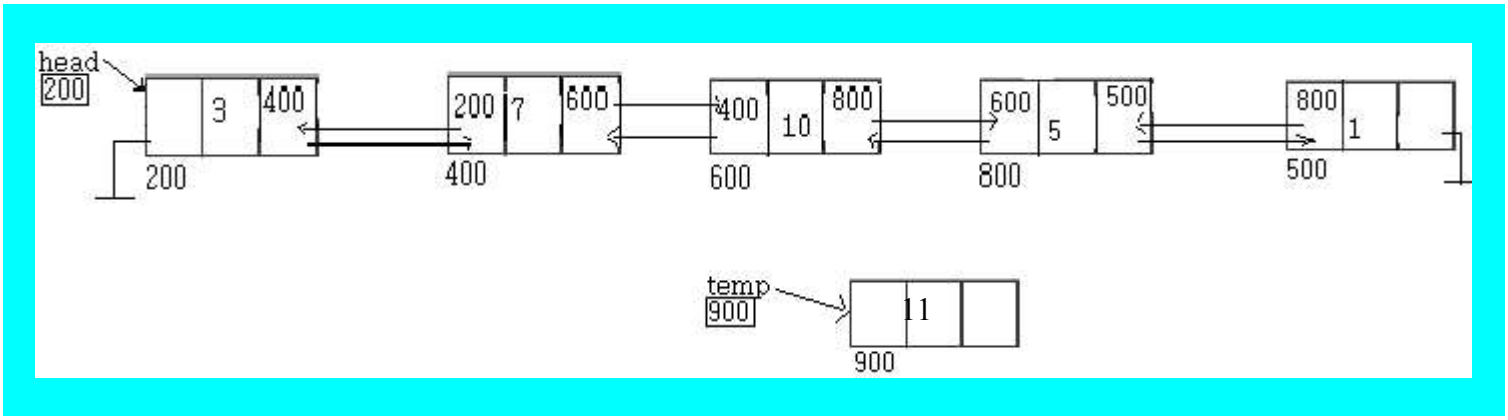
1. Insert before a node
2. Insert after a node

I will do insert after a node, and Insert before a node is your assigment.

Insert after a node

Step1: Create a node, store the address of this node in temp.

```
temp = (node *)malloc(sizeOf(node));
printf("Enter value");
scanf("%d",temp->data);
```



Step2: Ask the user after which node he wish to insert.

```
printf("Enter value of node after which you want to insert");
scanf("%d", &val);
```

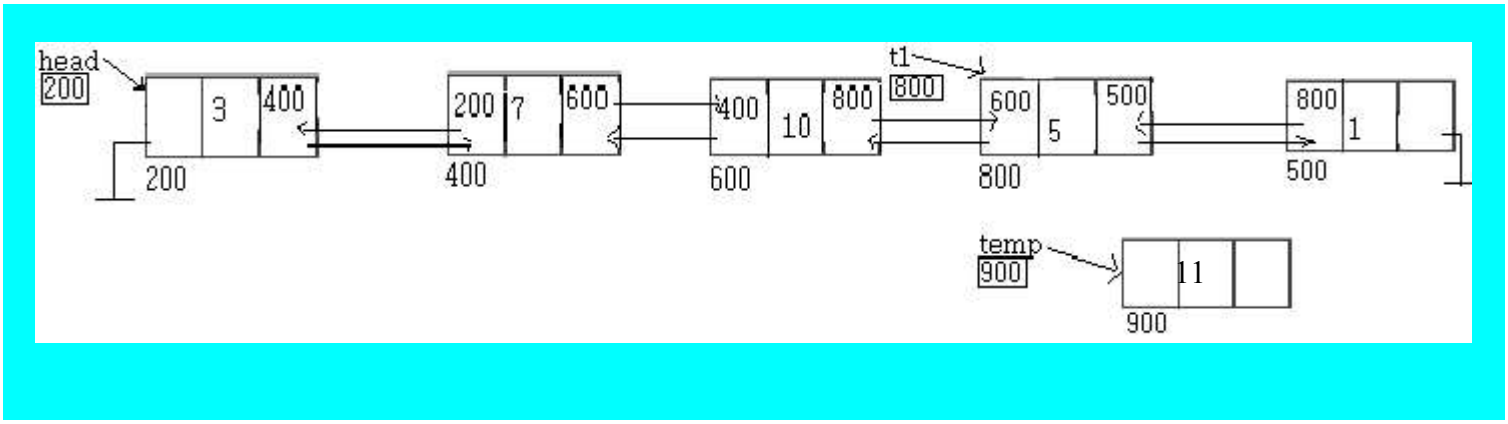
Step3: Suppose user enter 5, so val=5. This means, we want to insert a node after node with value 5.

Make t1 point to first node.

```
t1 = head;
```

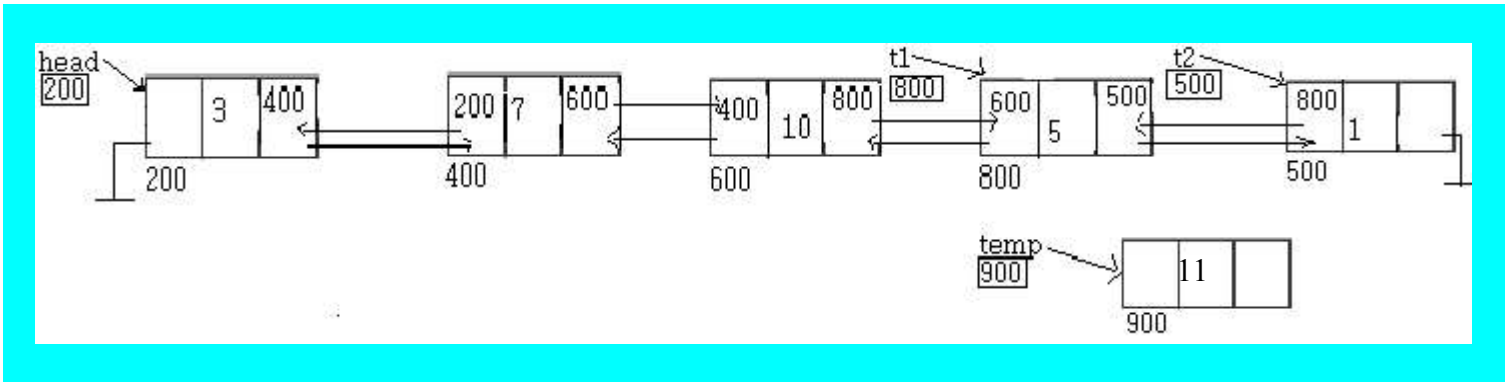
Move t1 to node with value 5.

```
while(t1->data != val)
        t1 = t1->rlink;
```



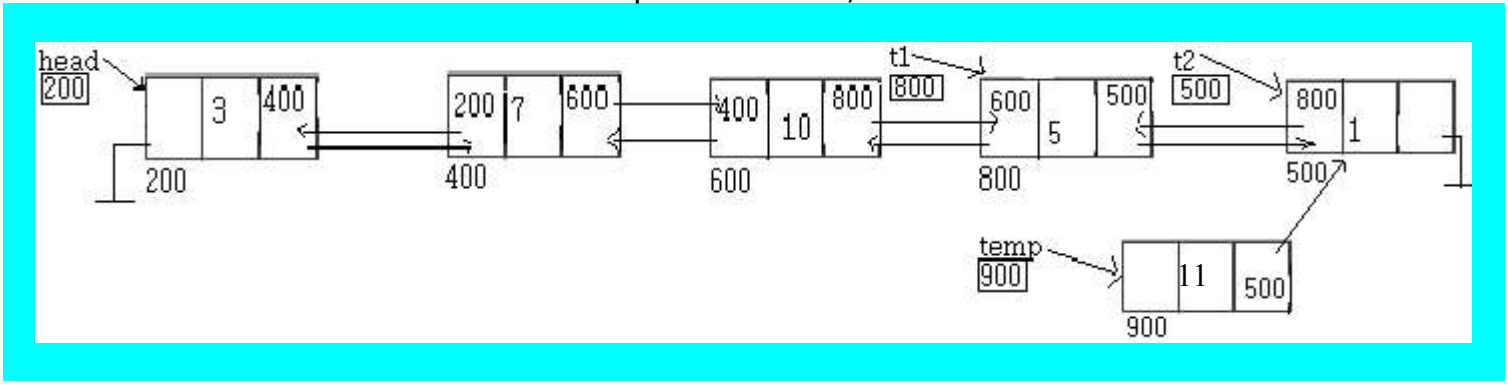
Step4: Make t2 store address of node after t1.

t2 = t1->rlink;



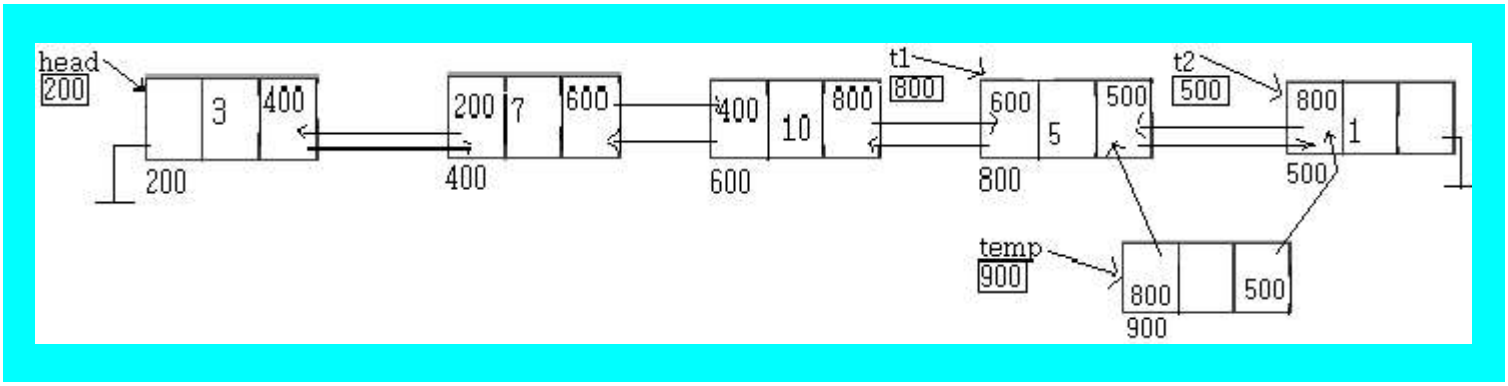
Step5: Now everything is in place, we only need to change addresses.
temp->rlink should contain 500.

temp->rlink = t2;



Step6: And, temp->llink shoud contain 800.

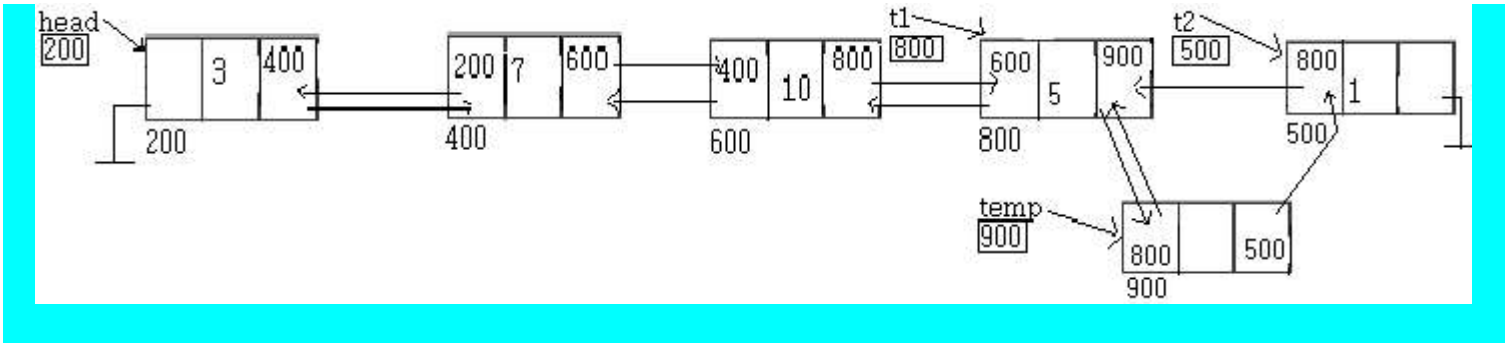
temp->llink = t1;



Step7: Store 900 in t1->rlink.

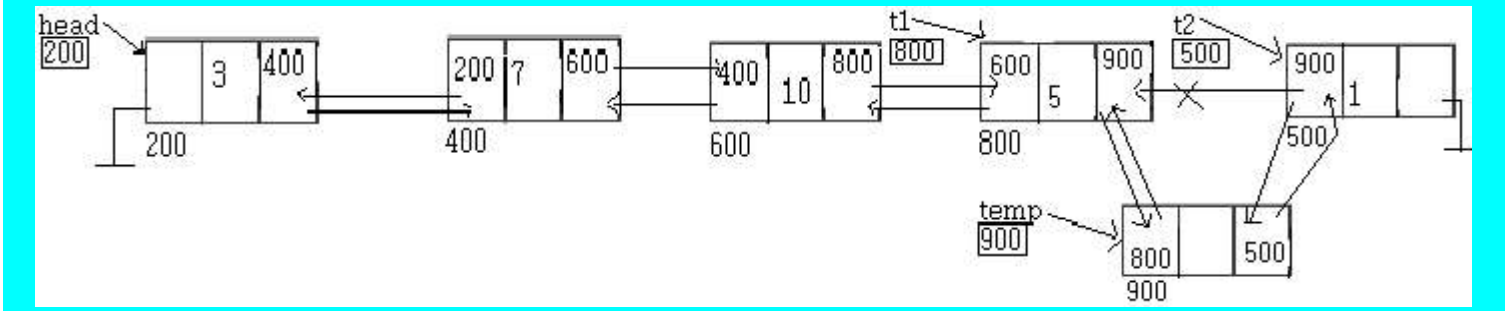
t1->rlink = temp;

Finally, we have this

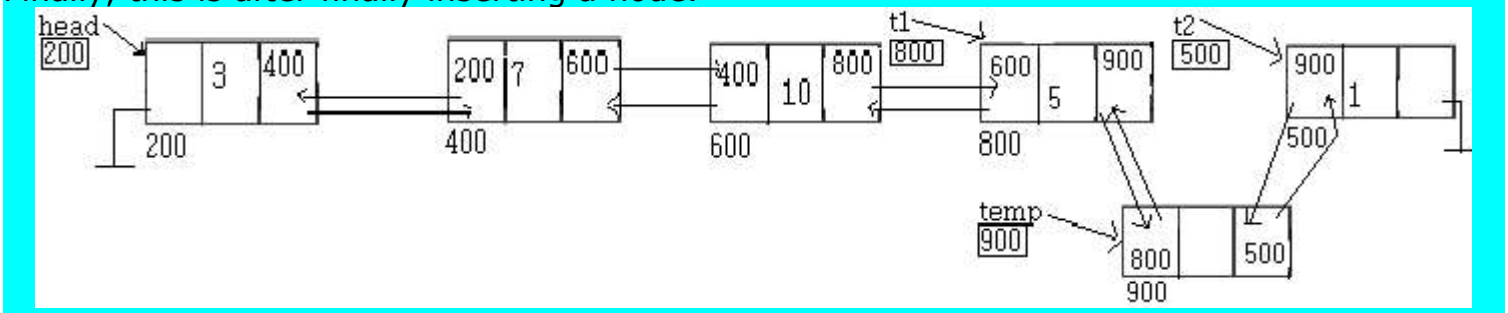


Step8: Store 900 in t2->llink.

t2->llink = temp;



Finally, this is after finally inserting a node.



Deletion in a Double Linked List

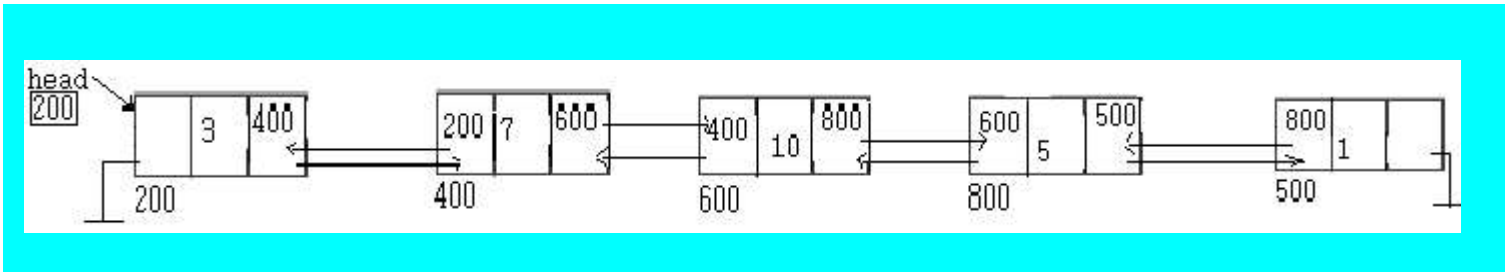
After inserting a node, we will learn how to delete a node from a double linked list. Logic for insertion and deletion of nodes in double linked list is similar to single linked list.

Again there are 3 types of deletion possible
1. Delete first node
2. Delete last node
3. Delete anywhere

We will start with Delete first node.

Delete first node

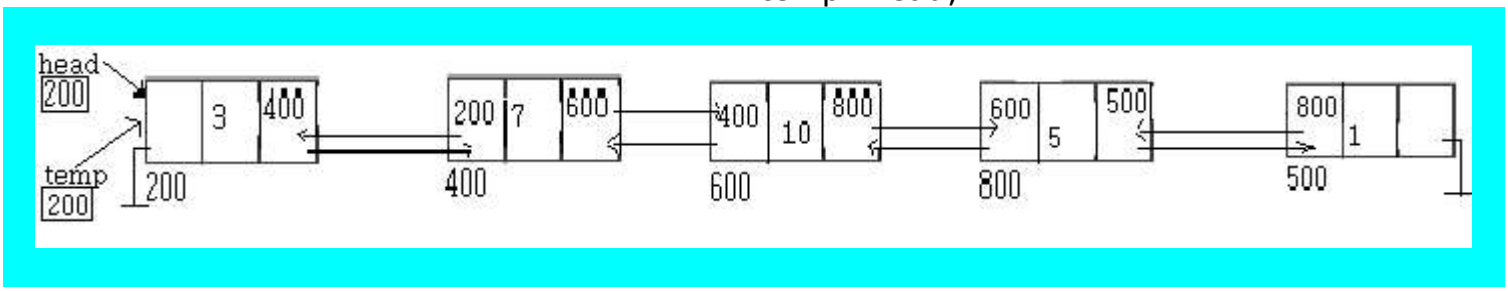
The name of topic itself defines what we want to do. We want to delete first node of double linked list.
We assume that we have a linked list, with head points to the first node.



We want to delete node pointed by head.
If we delete first node, second node (whose address is 400) will become first node.

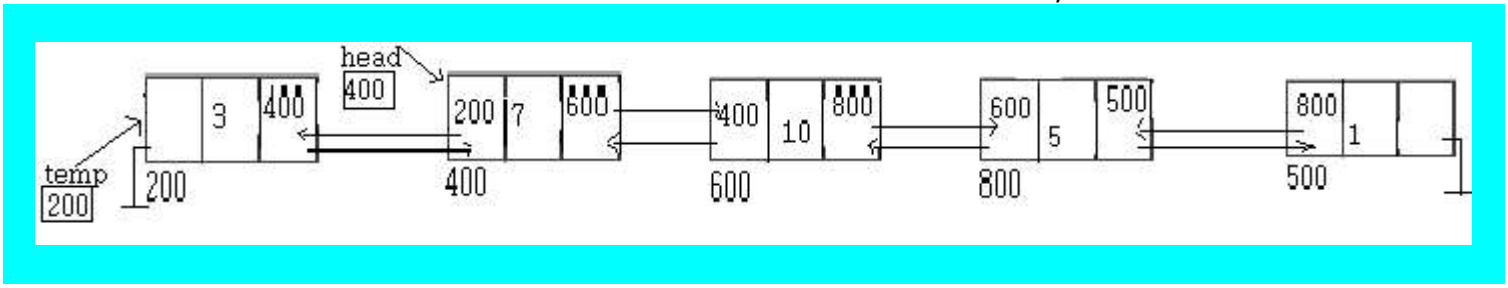
Step1: Make temp point to the first node.

temp=head;



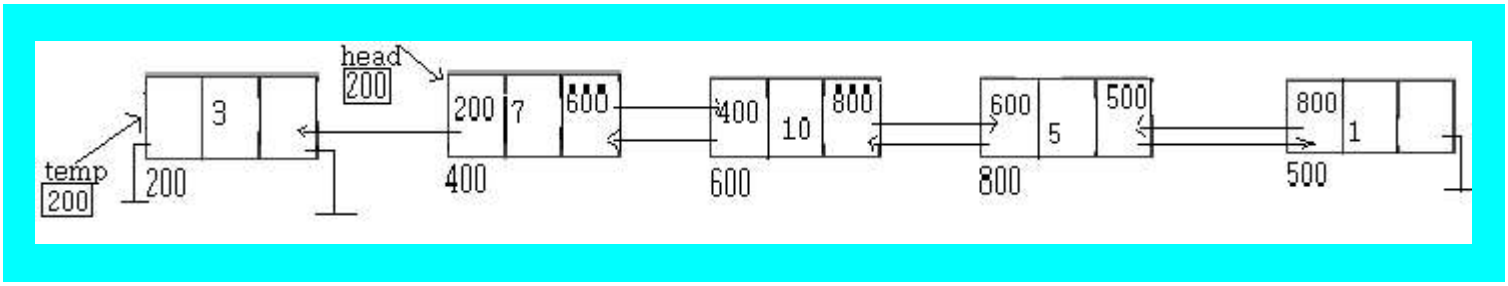
Step2: Move head to second node.

head=head->rlink;



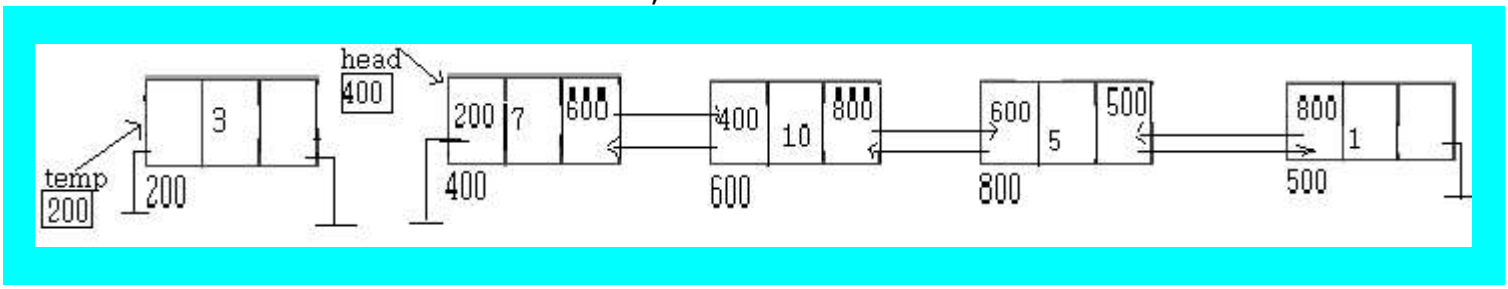
Step3: Since we want to delete temp, temp->rlink should contain null.

temp->rlink = null;



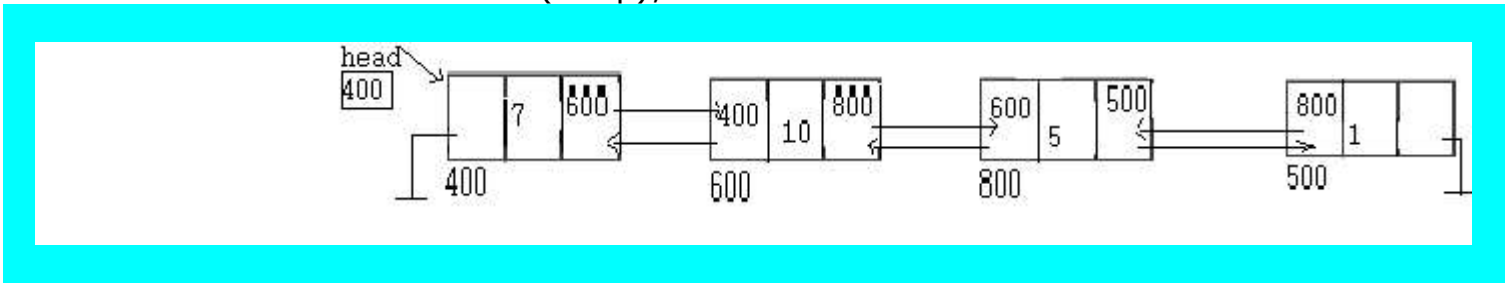
Step4: Finally make temp isolated from the double link list. Store null in head->llink;

head->llink=null;



Step5: Now temp is completely isolated from the doubled link list. We can delete temp now.

free(temp);

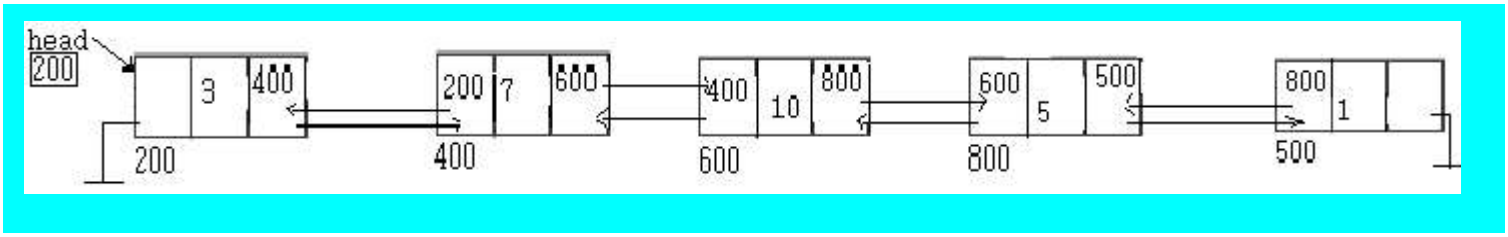


Delete last Node

So, Delete frist node was pretty simple, You wont be disappointed for delete last node also. This one is also simple.

Infact one of the advantage of double link list over single link list is that, insert and delete operations on double link list is easier compared to same operations on double link list.

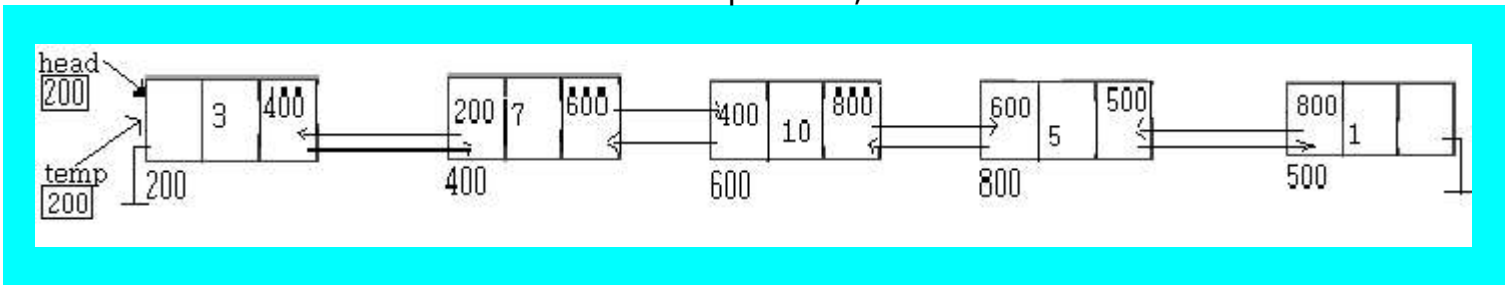
So, here we go. We assume that we have a double link list, and head points to the first node.



We want to delete last node of this double link list i.e. We want to delete node with address 500.

Step1: Make temp point to first node.

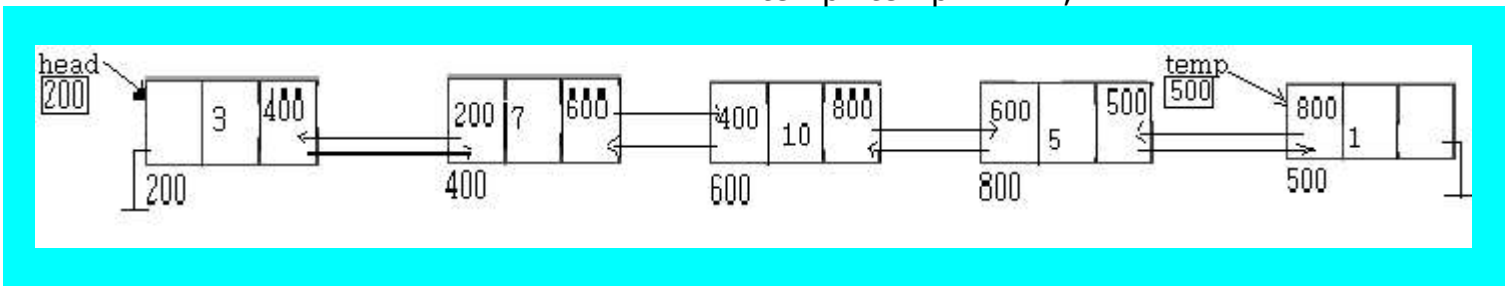
temp=head;



Step2: Move temp to the last node.

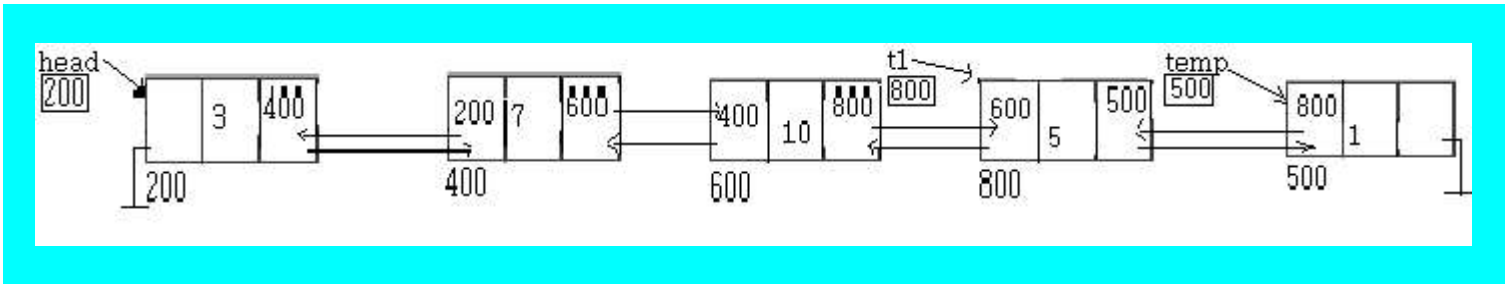
while (temp->rlink != null)

temp=temp->rlink;



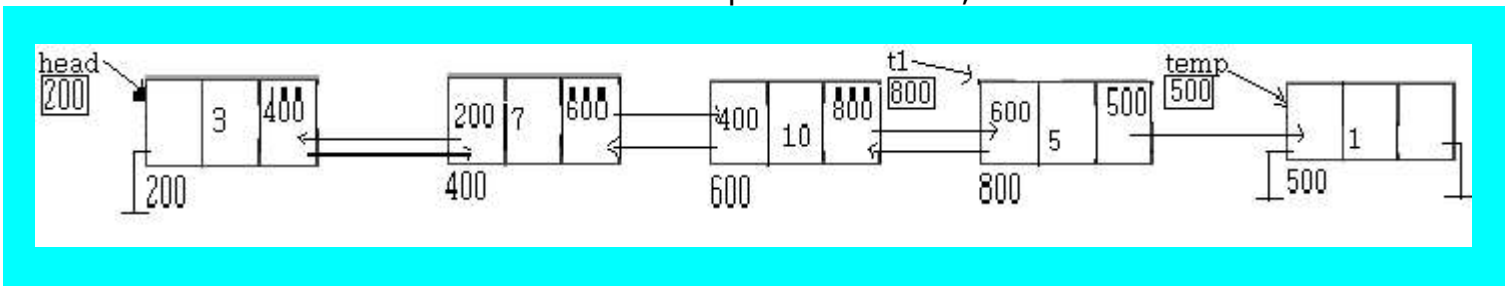
Step3: Make t1 point to node before temp.i.e. Store 800 in t1.

t1=temp->llink



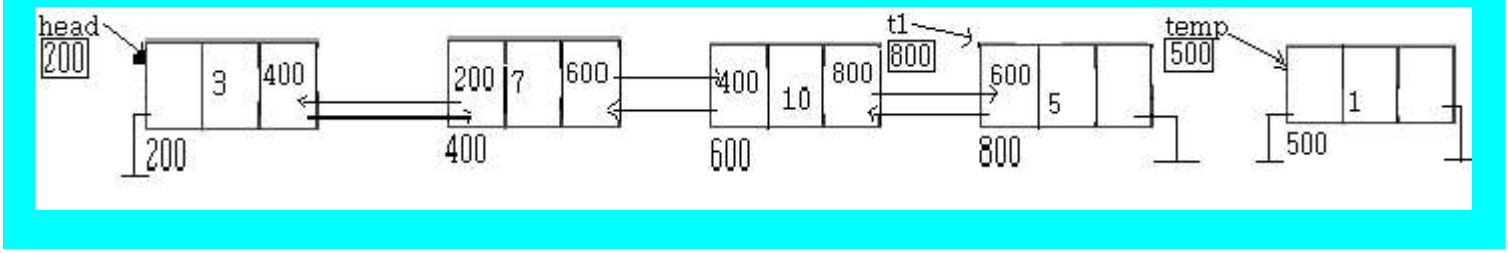
Step4: We need to store null in temp->llink.

temp->llink = null;



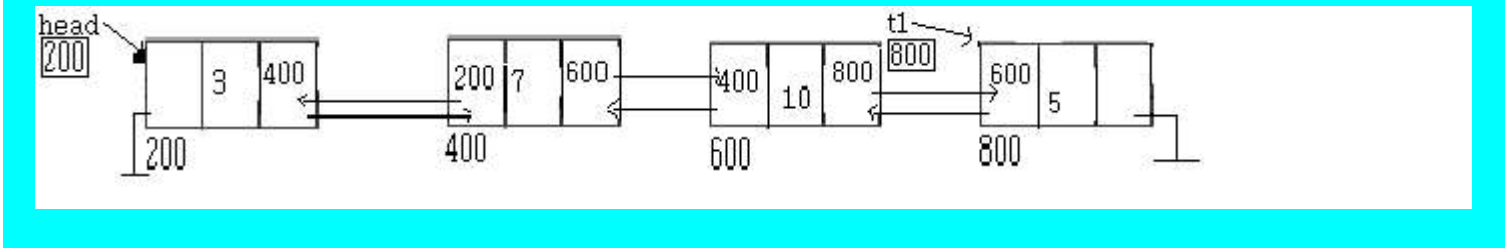
Step5: Store null in t1->rlink, thus isolate temp from double link list.

t1->rlink = null;



Step6: Now, temp is completely isolated from double link list.

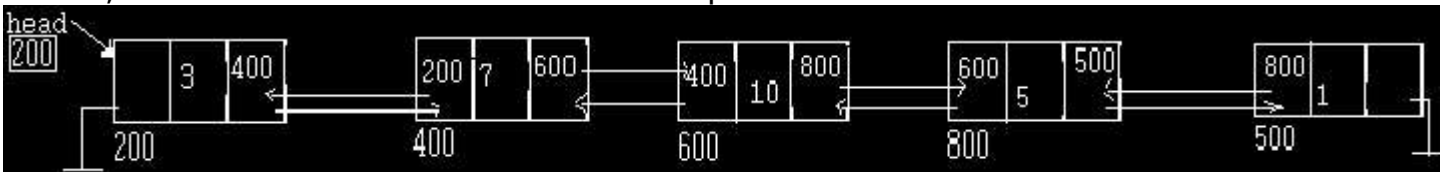
free(temp);



Delete anywhere

Delete anywhere means, delete a node which user want to delete. We will ask the user which node he want to delete.

As usual, we have a double link list and head points to the first node.

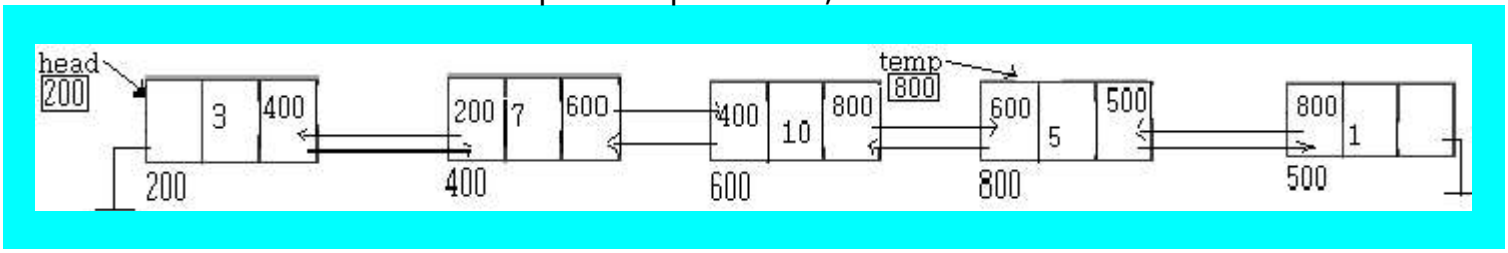


Step1: We will ask the user which node he want to delete. User will enter the value of node, he want to delete.

```
printf("Enter the value of node you wish to delete");
scanf("%d",&val);
```

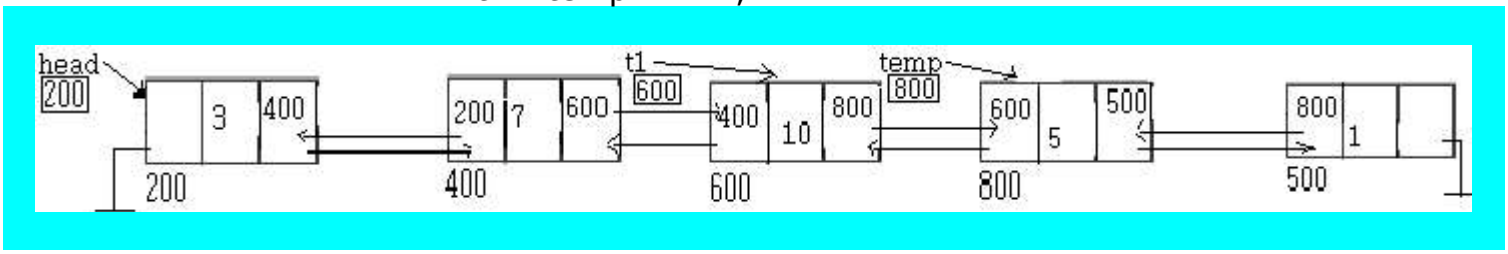
Step2: Suppose user enter 5, so val=5. That means, we have to delete node with data value 5. We will make temp point to node with value 5.

```
temp = head;
while(temp->data != val)
        temp = temp -> rlink;
```



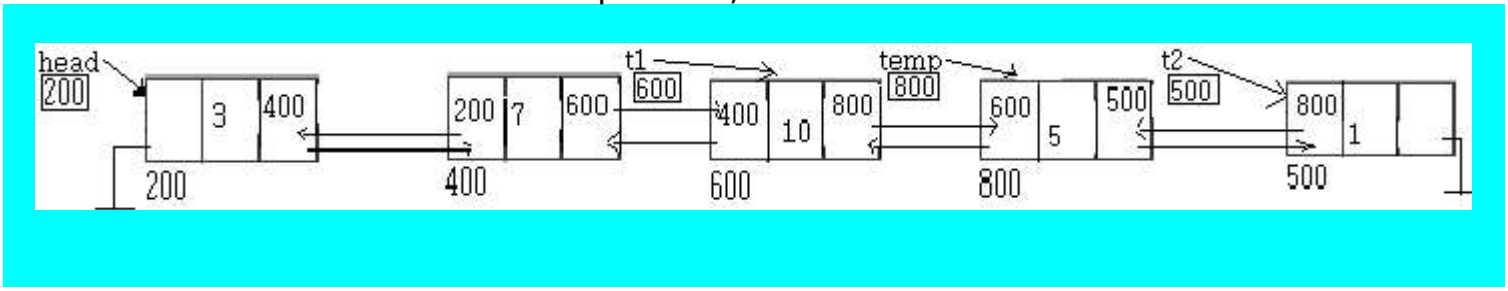
Step3: Make t1 point to the node before temp. i.e. t1 will store the address 600.

```
t1= temp->llink;
```



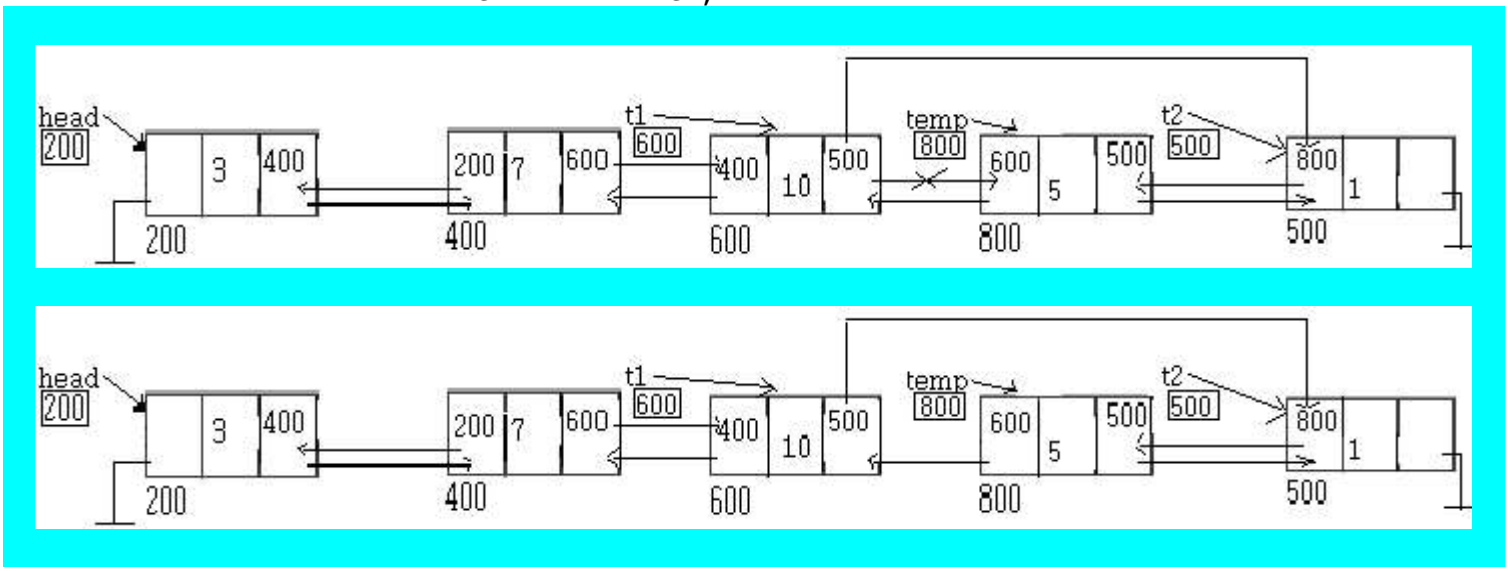
Step4: Make t2 point to the node after temp i.e. t2 will store the address 500.

```
t2=temp->rlink;
```



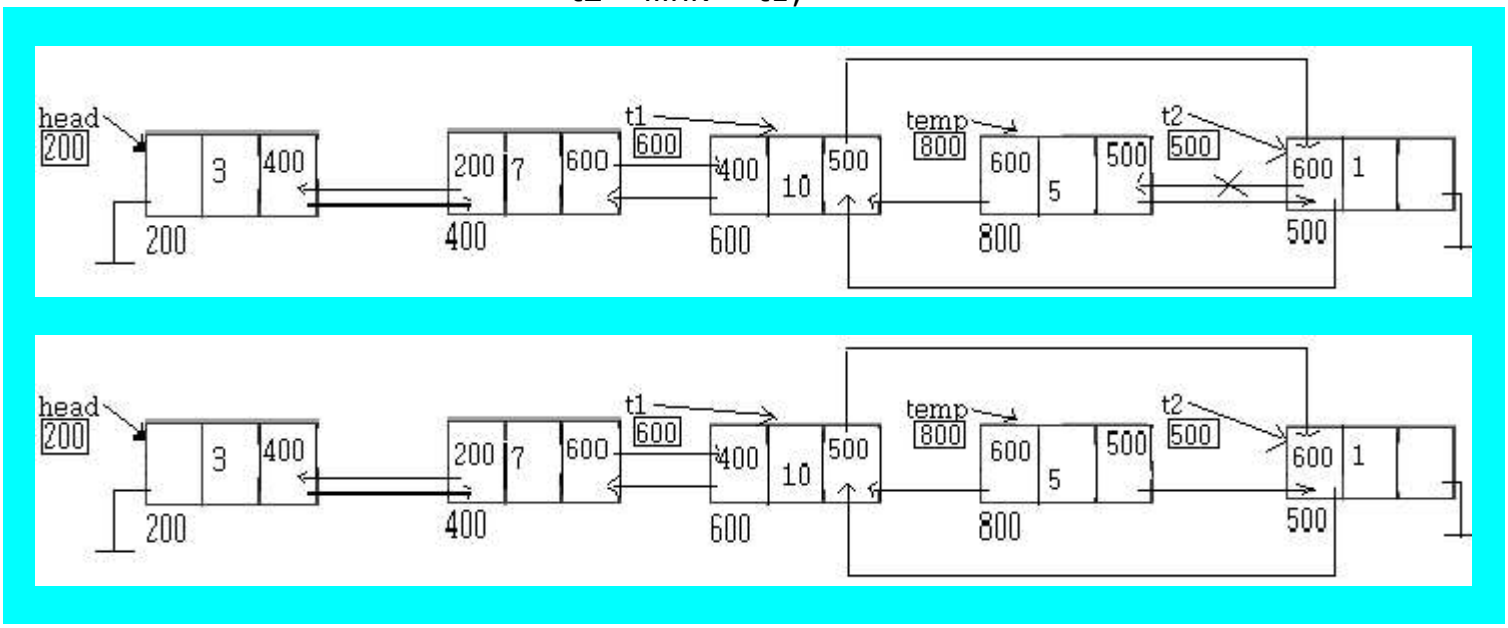
Step5: We want to delete temp, that means, t1->rlink should store the address of t2. i.e t1->rlink should store 500.

t1->rlink = t2;



Step6: Now, t2->llink will store the address of t1. i.e. t2->llink should store 600.

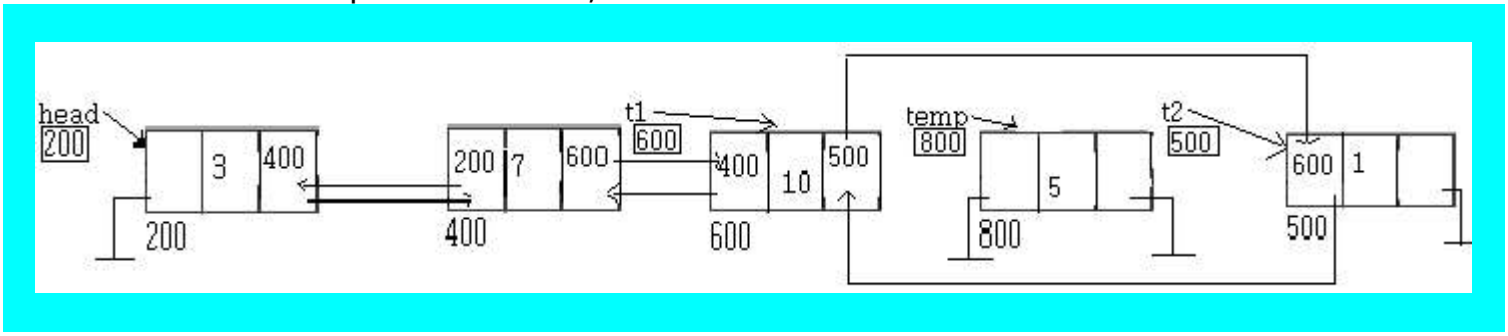
t2->llink = t1;



Step7: Now, temp is abandoned by the double link list, only temp is holding this double link list from left and also from right.

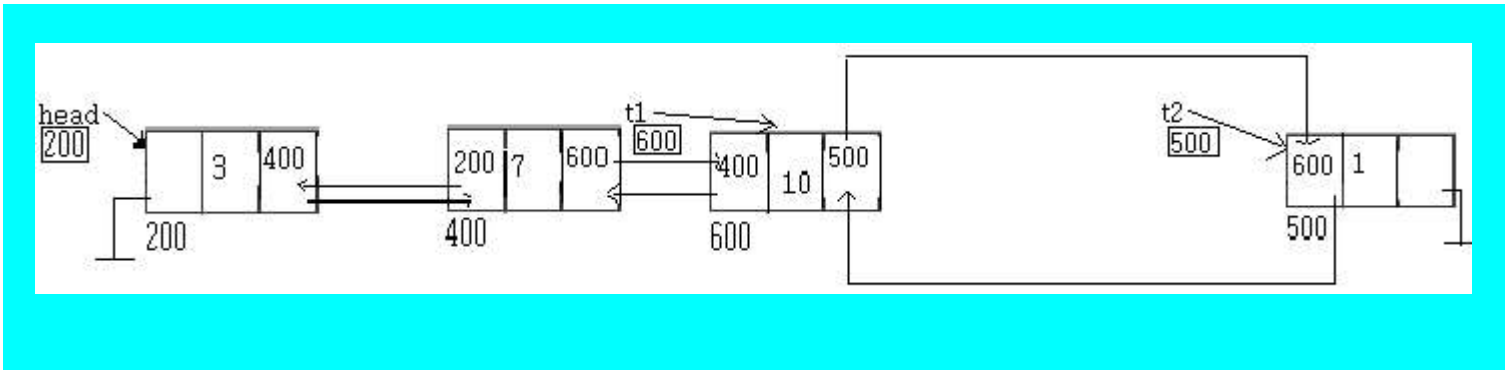
We want temp->llink to store null and also temp->rlink to store null, thus completely isolating temp from double link list.

```
temp->llink= null;
temp->rlink = null;
```



Step8: We are almost finished with our job. Just delete temp now.

```
free(temp);
```



This finishes the operation insert and delete in a double link list.