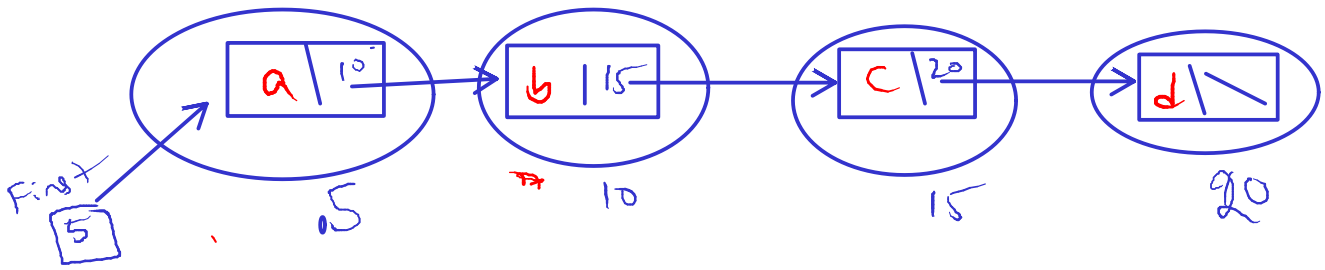
Pitfalls of SLL => Singly linked list



Struct node * temp = first

t = t -> link // Move forward

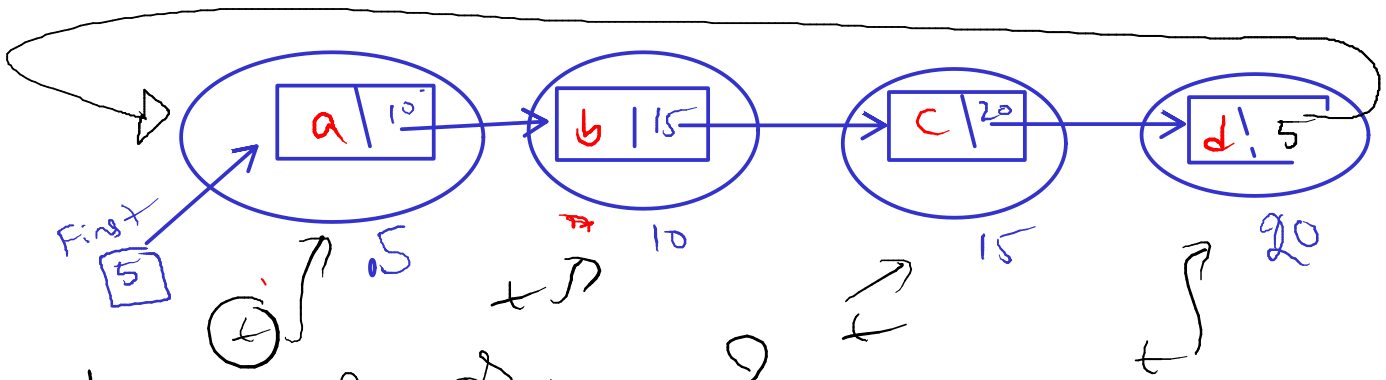
Pitfalls: ① Stepping backwards : Not possible.

OR

The addr of the predecessor : Not known

② The link part of the last node is Not utilized

Hence the sll has become circular linked list.
The link part of the last node is utilized by placing the address of the first node

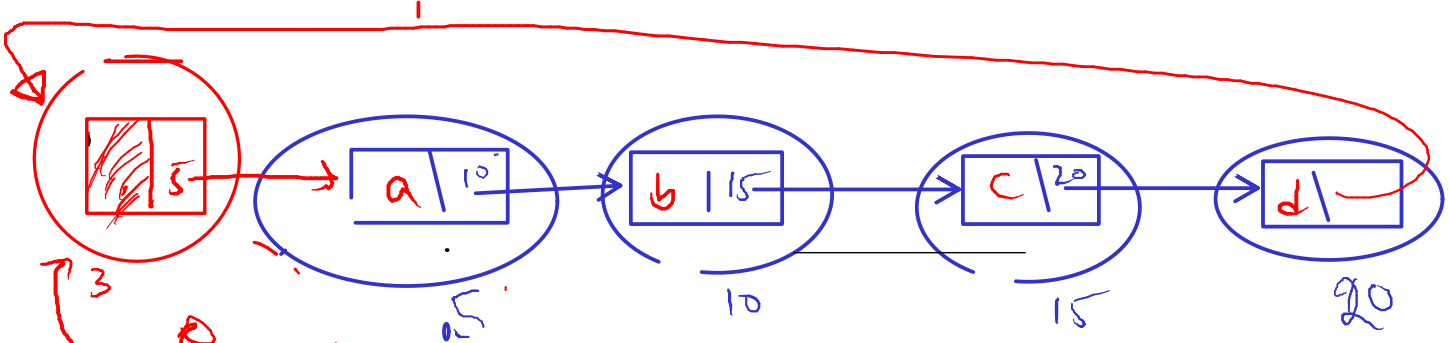


Disadv of CLL = ?

```
c = 0
t = first
while (t)
{   c++;
    t = t -> link;
}
```

SLL code, if applied on CLL, there is danger of falling into Infinite loop.

Hence the concept of **Head Node** is introduced



while (t) ⇒ SLL

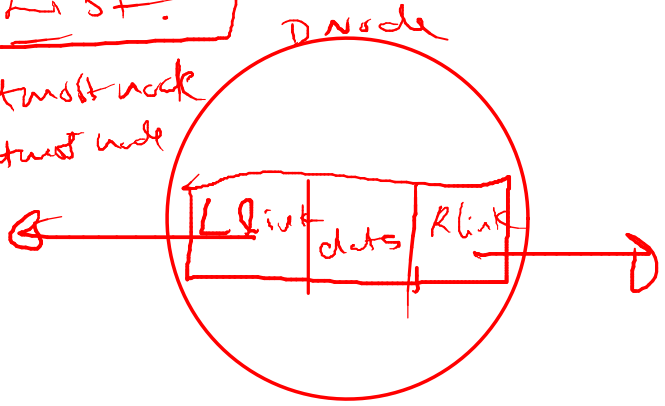
while (t != Head) ⇒ CLL

Doubly Linked List

L = holds addr of the leftmost node

R = holds addr of the Rightmost node

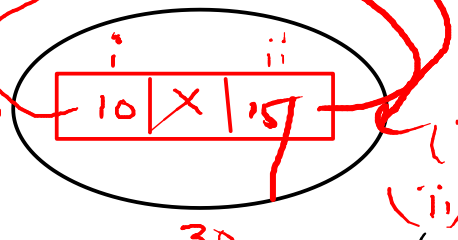
struct DNode




```
{
    struct DNode *Llink;
          char data;
    struct DNode *Rlink;
};
```

struct DNode *L = NULL;
*R = NULL;

Say 'M' is given some middle node and data



n → data = 'c'

(i) n → Llink = M

(ii) n^5 → Rlink = M^10 → Rlink^15

struct DNode * n = (struct DNode *) malloc(sizeof(struct DNode));

(iii) M → Rlink = n

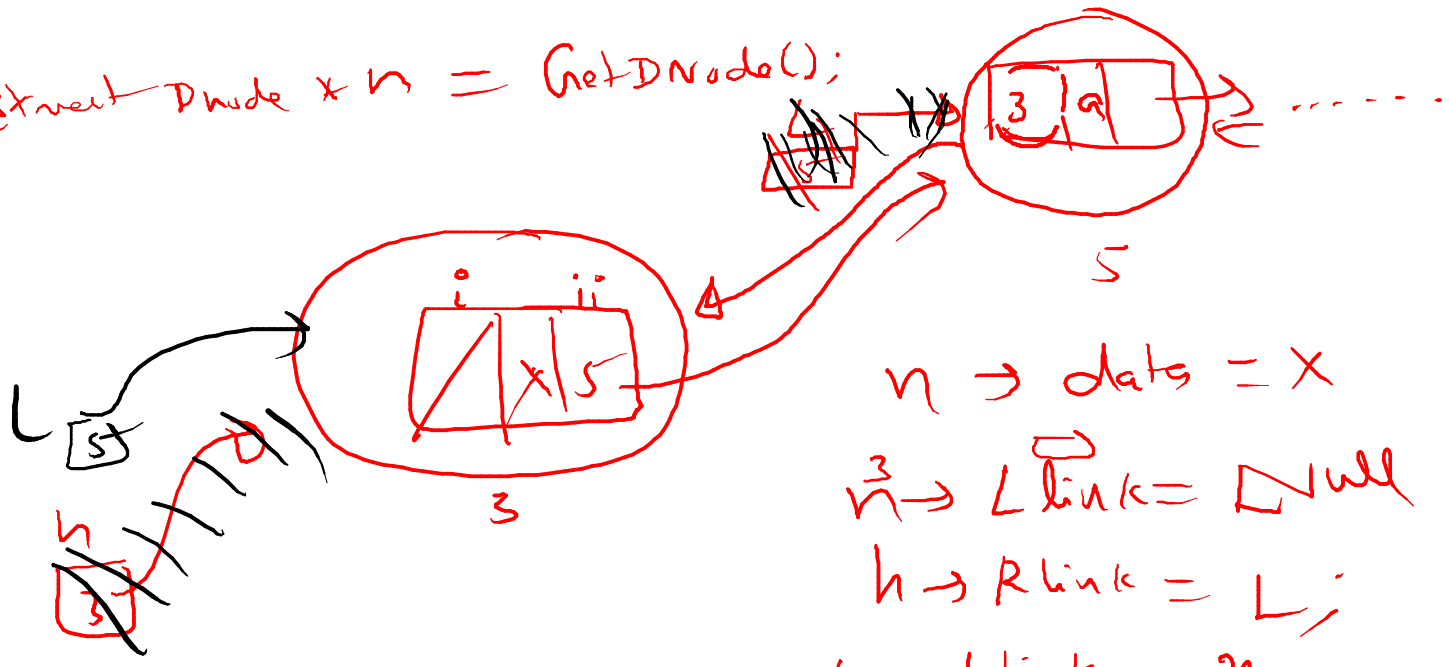
(iv) n^30 → Rlink → Llink = n

howmany link fields are getting updated for the following operations:

Operation	Leftmost	Rightmost	else
Insert	3 ✓	3	4 ✓
delete	1	1 ✓	2 ✓

Insert Leftmost

struct Dnode *n = GetDNode();



n → data = x

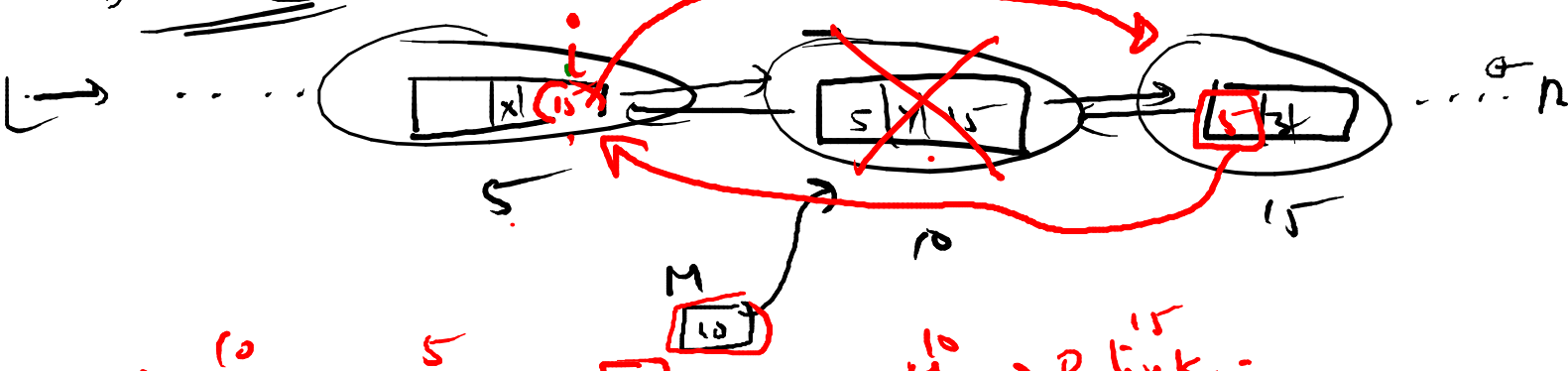
n → Llink = NULL

h → Rlink = L;

L → Llink = n

L = n;

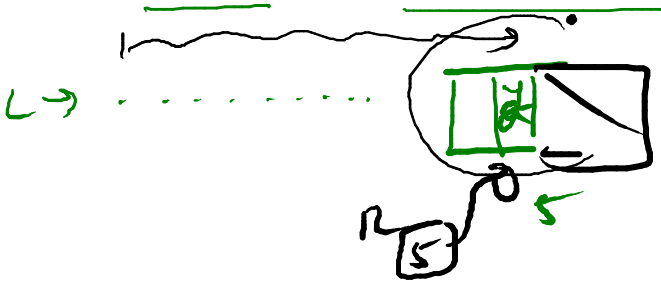
Delete M = address of mid node



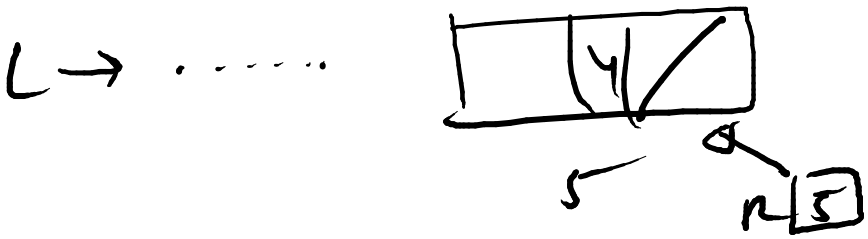
(i) M → Llink → Rlink = M → Rlink;

(ii) M → Rlink → Llink = M → Llink; free(M)

Delete Rightmost




```
R = R->Llink;
free(R->Rlink);
R->Rlink = NULL;
```




```
struct Node *t = R;
R = R->Llink
R->Rlink = NULL
free(t)
```

t is known as dangling pointers

} // out of Scope.