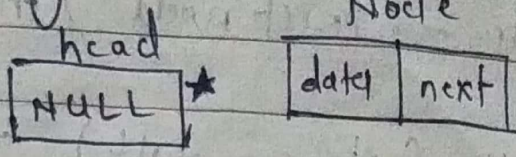
LinkList

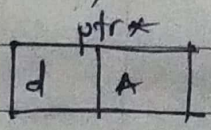
LinkList Operations

1) Insert at the beginning




```
int data
struct node* next
struct node* head;  (By default value stored Null in hea
Insert Method()
{
struct Node* ptr;  — temp variable which will store ad
int item;  — new item to add at beginning.
ptr=(struct node*) malloc (sizeof (struct node*));
```

Typecasting — memory allocation



ptr == NULL ⟶ overflow

but ptr == NULL ⟶ false then

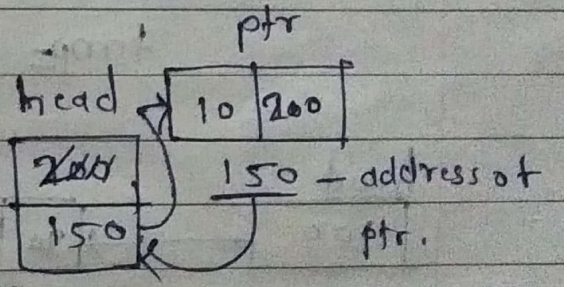
Read value in item

item = 10

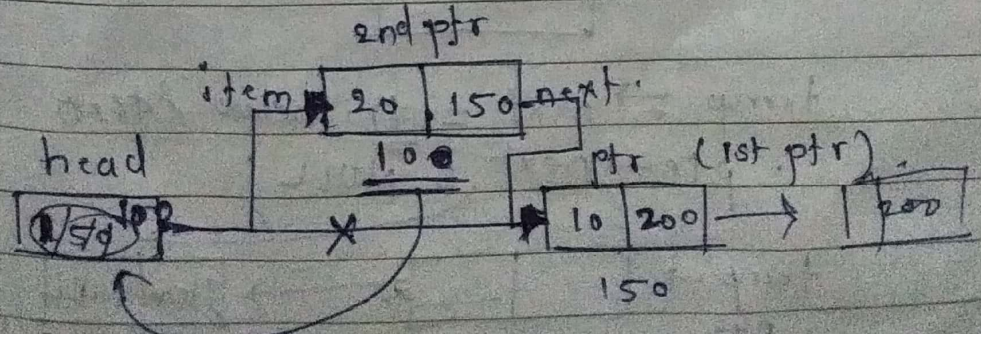
ptr → data = item

ptr → next = head

head = ptr

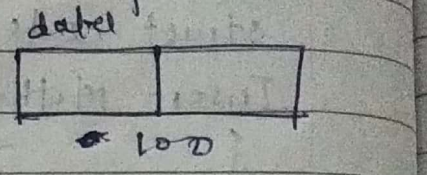


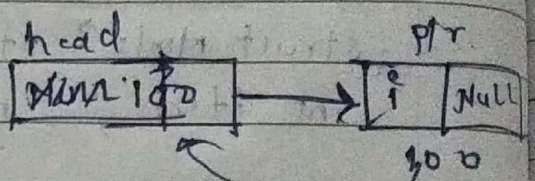
Again we call Insert function

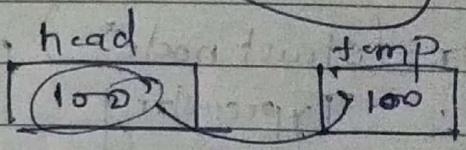


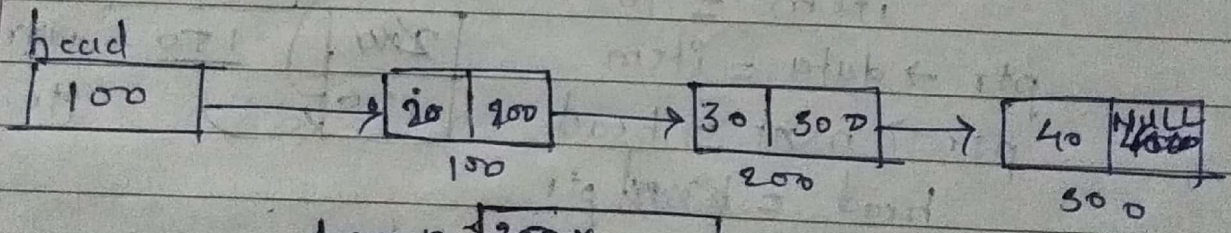
2) Insert at Last

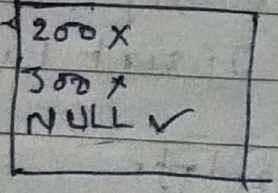
```
struct Node *ptr, *temp;
int item
ptr = (struct node*) malloc (sizeof (struct node*));
if (ptr == NULL) → overflow
else Take input
    ptr → data = item
    if (head == NULL)
    {
        ptr → Next = NULL
        head = ptr
    else
        temp = head
while (temp → next != NULL)
    temp = temp → next;
```





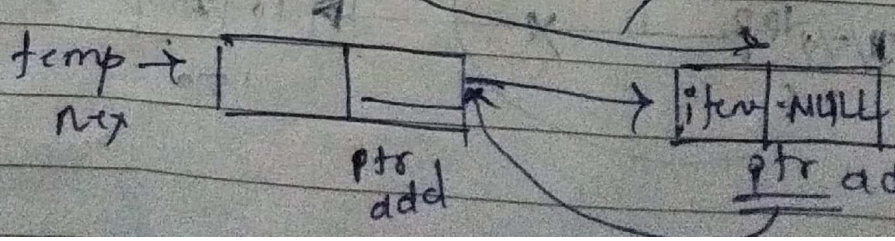


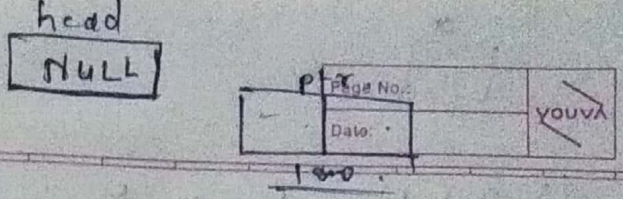




temp → next = ptr

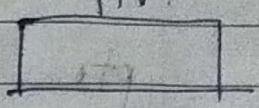
ptr → next = NULL



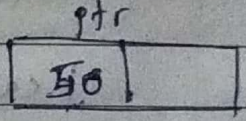


3) Random Insert ()

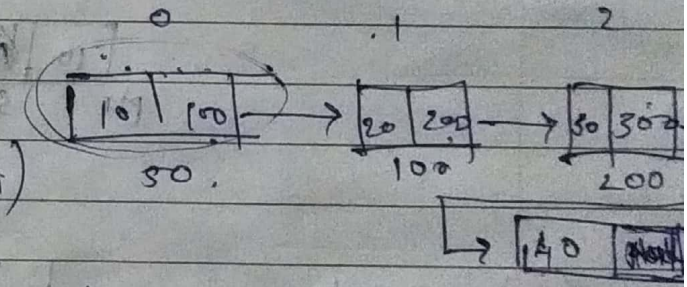
```
int i, loc, item
Struct node *ptr, *temp;
ptr = (struct node*) malloc (size of (struct node*));
```

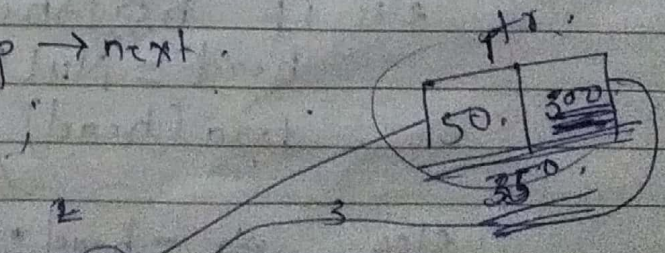


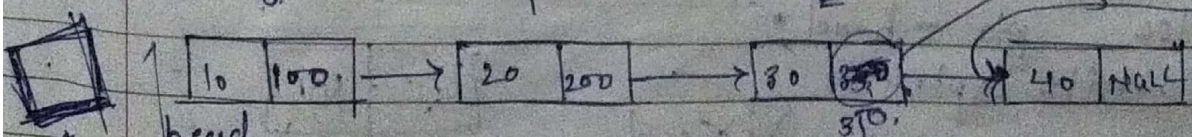

```
if ptr == Null → overflow
else
    Take input from user
        item = 50.
    ptr → data = item
```




```
then take the location after which
you want to insert data.
        location = 2;
    temp = head;
for (i=0; i<loc; i++)
{
    temp = temp → next;
    if (temp == Null)
    {
        print — cant insert
    }
}
{ ptr → next = temp → next.
  temp → next = ptr;
```







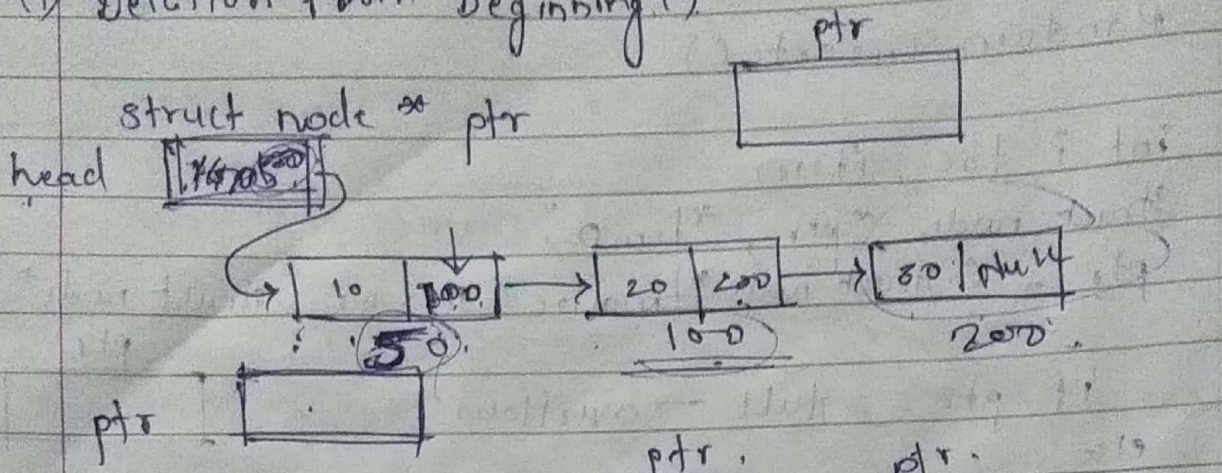
head

temp (100)

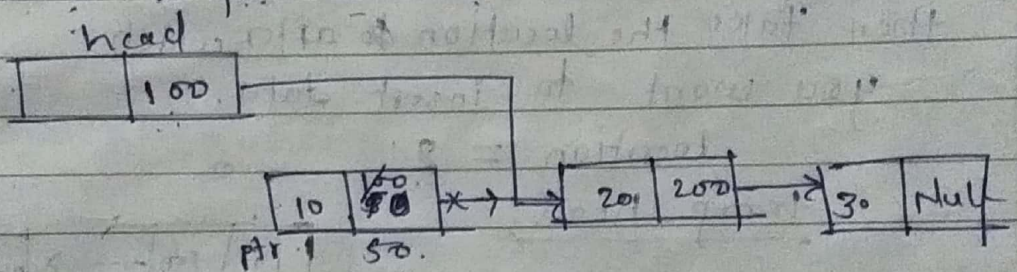
i = 0 temp temp → next = temp (100)

i = 1 temp = temp → next = temp (200).

(4) Deletion from Beginning ().



head = ptr → next . 50



(5) Delete from last ().

struct node *ptr, *ptr1;

if head == Null — empty
else if head->next == Null
head = Null;
free (head);

else ptr = head;
while (ptr -