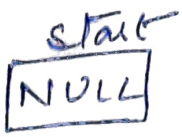
(b) Insertion in an empty list

Before insertion

start is NULL




After Insertion

T is the only node

start points to T

Link of T is NULL.

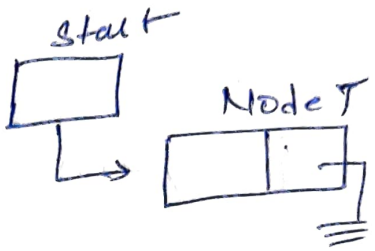



Fig: Insertion in an empty list

When the list is empty, value of start will be NULL. The new node that we are adding will be the only node in the list. Since it is the first node, start should point to this node and it is also the last node so its link should be NULL.

```
tmp->next = start;
start = tmp;
```

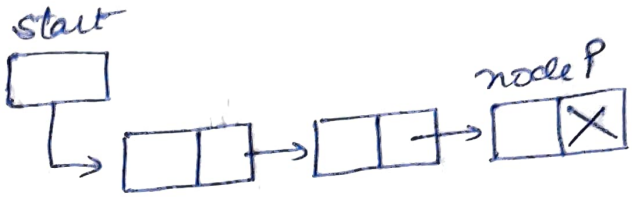
Since initially start was NULL, so we can write start instead of NULL.

(c) Insertion at the end of the list

We have to insert a new node T at the end of the list. Suppose the last node of list is node P, so node T should inserted after node P.

Before insertion

Node P is the last node
next of node P is NULL.
(link)




After insertion

Node T is last node
Node P is second last node
link of node T is NULL, link of P points to node T.

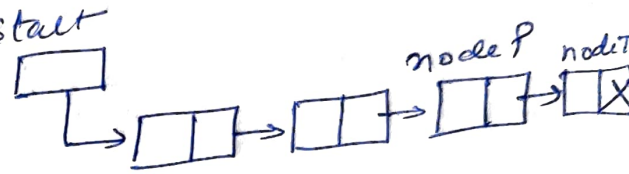



fig: Insertion at the end of the list

Suppose we have a pointer p pointing to the node P. There are two statements:

```
p -> next = tmp;
tmp -> next = NULL;
```

To traverse the list till the end to get the pointer p and then do the insertion.

```
p = start;
while (p -> next != NULL)
    p = p -> next;
```

b) Inserting a node at the End

Insert_last (start_item)

(1) [check for overflow]

if Ptr = NULL

then

Print "Overflow"

exit

else

Ptr = (node *) malloc(sizeof(node));

endif

(2) Set ptr → info = item

(3) Set ptr → next = NULL

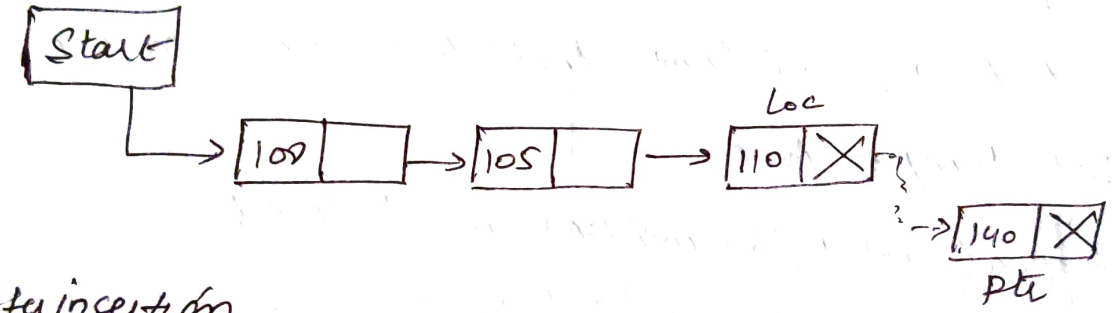
(4) if start = NULL and if then Set start = ptr;

(5) Set loc = start

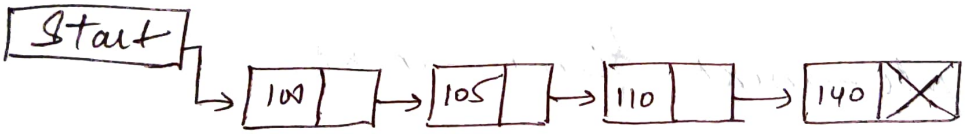
(6) Repeat Step 7 until loc → next != NULL

(7) Set loc = loc → next

(8) Set loc → next = ptr




After insertion




c) Inserting a new node at the specified position

Insert_location (start, item, loc)

(1) [check for overflow?]

if ptr = NULL

then

print "Overflow"

exit

else

ptr = (node *) malloc (sizeof (node));

endif

(2) Set ptr → info = item

(3) if start = NULL

then

set start = ptr

set ptr → next = NULL

endif

(4) Set I = 0, node *temp // Initialize counter and pointer

set temp = start

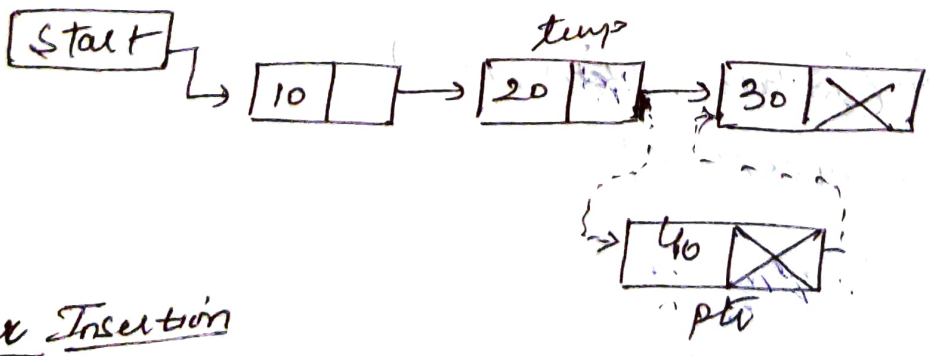
(5) Repeat Step (6) and (7) until I < loc

(6) set temp = temp → next

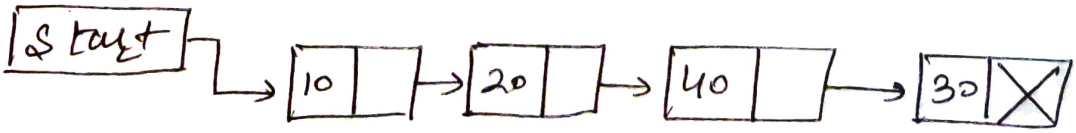
(7) Set I = I + 1

(8) Set ptr → next = temp → next

(9) Set temp → next = ptr




After Insertion

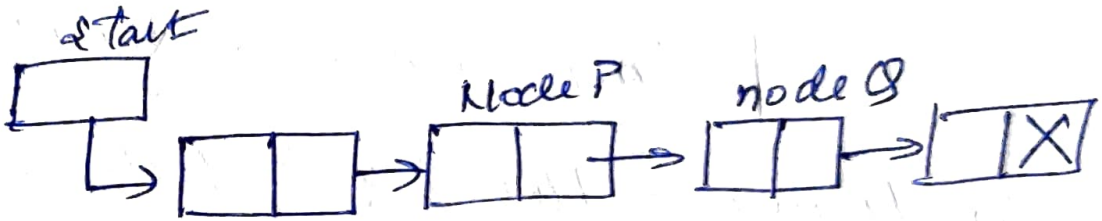


## (d) Insertion in between the list nodes

We have to insert a node T between node P and Q.

**Before insertion**

Node Q is after node P. Link of P points to node Q




**After insertion**

Node T is between node P and Q. Link of node T points to node Q. Link of node P points to node T.

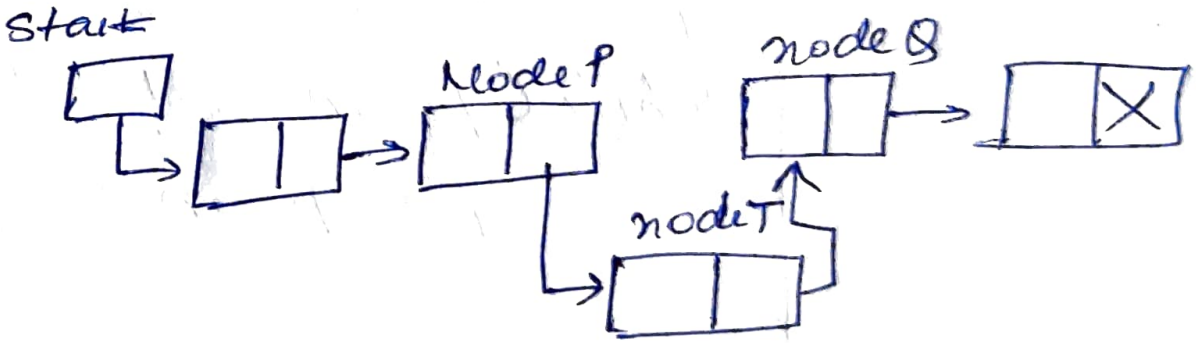



Fig: Insertion in between the list nodes

Suppose we have two pointers p and q pointing to nodes P and Q respectively.
The two statements written as:

```
tmp -> next = p -> next;
p -> next = tmp;
```

we cannot write here

```
tmp -> next = q;
```

Note:
The order of these two statements are very important
If the order reversed then

```
p -> next = tmp;
```

will lose the address of Q because the address of node Q is in p -> next.

Three cases of insertion in between the nodes:

a) Insertion after a node.
b) Insertion before a node.
c) Insertion at a given position.