Some students have a problem in understanding the difference between `while(p!=NULL)` and `while(p->link!=NULL)`, so I have written this article with full explanation. Please go through it, and feel free to post in the Q&A forum, if anything is not clear.

## FAQ 1 :

What is the difference between

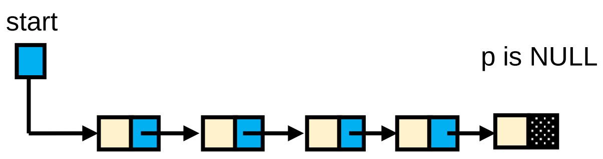
`while(p!=NULL)` ,

`while(p->link!=NULL)` and

`while(p->link->link!=NULL)` in linked lists?

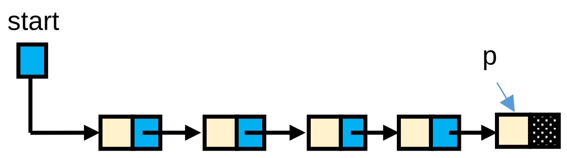
When to use which one?

```
p = start;
while(p!=NULL)
{   .........
  p = p->link;  }
```



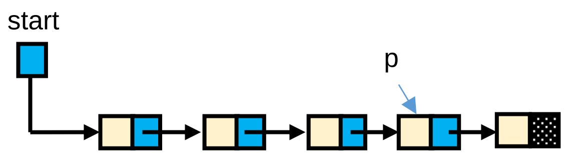
**After the loop terminates, p is NULL**

```
p = start;
while(p->link!=NULL)
{   .........
  p = p->link; }
```



**After the loop terminates, p points to last node**

```
p = start;
while(p->link->link!=NULL)
{   .........
  p = p->link;  }
```



**After the loop terminates, p points to second last node**

When the loop condition is `while(p!=NULL)`, the loop executes till p is not `NULL`, it terminates when p becomes `NULL`.

So after the loop terminates, p will be `NULL`. We use this condition when we want to visit each node of the list till the last node, for example when we have to print values in all nodes or when we have to search for a value in the list. This condition we have used in the previous lecture in traversal, where we had to access info of all the nodes starting from first node till the last node.

…………………………………………………………………………………..

When the loop condition is `while(p->link!=NULL)`, the loop executes till `p->link` is not `NULL`, it terminates when `p->link` becomes `NULL`. `NULL` is present in the link of last node, so after this loop terminates p will point to the last node.

We can use this loop when we want a pointer to last node of the list. We will use it when we have to insert a new node after the last node because in that case we need a pointer to the last node.

When you want to find pointer to last node, you don't want to stop the loop when p becomes `NULL`, you want to stop it when p points to the last node. So you have to write the condition `while(p->link!=NULL)`. If you write `while(p!=NULL)`, then when the loop stops, p will be `NULL`.

So write `while(p!=NULL)`, when you want to loop till p becomes `NULL`.
And write `while(p->link!=NULL)`, when you want to loop till p points to last node.

We have used the condition `while(p->link!=NULL)` when finding pointer to predecessor of a node with particular info.

```
p=start;
while(p->link!=NULL)
{
  if(p->link.info==x)
    break;
  p=p->link;
}
```

This is because in this case we want the loop to stop when p points to the last node. We don't want the loop to execute when p points to last node, because inside the loop we are checking `p->link->info`.

If we write `while(p!=NULL)`, then in the last iteration of the loop, p will point to the last node and `p->link->info` will give error because `p->link` is `NULL`.

………………………………………………………………………………………

When the loop condition is `while(p->link->link!=NULL)`, the loop executes till `p->link->link` is not `NULL`, it terminates when `p->link->link` becomes `NULL` and `p->link->link` will become `NULL` when p points to the second last node.

So after this loop terminates p will point to the second last node.

You can use this loop when you want pointer to second last node of the list. We will use it when we have to delete the last node because in that case we need a pointer to the second last node.

..............................................................................................

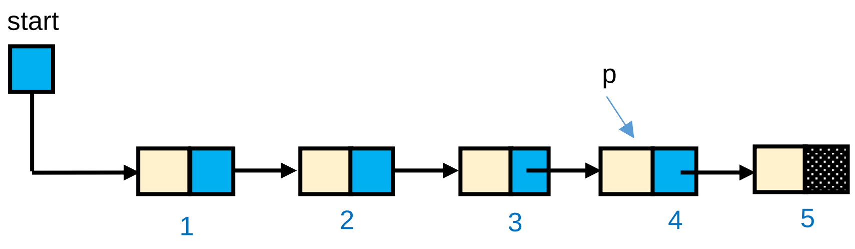
**FAQ 2:**

Here is a doubt that comes up in the lecture on insertion, it is better to clarify this point here only.

We have seen that this is the code for finding pointer to a node at particular position.

```
p=start;
for(i=1; i<k && p!=NULL; i++)
   p=p->link;
```

If k is 4, then after the above loop terminates, p will point to the fourth node.

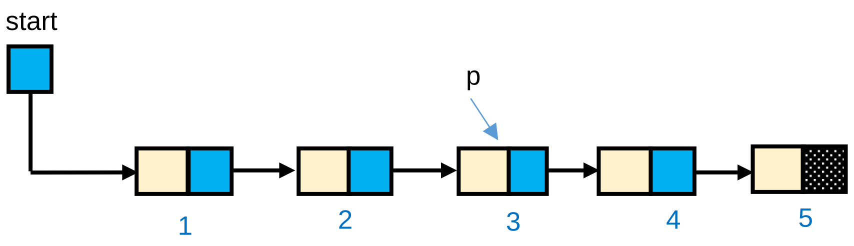


Now suppose we need reference to the predecessor of kth node, so if k is 4, then we need reference to third node.

For this we will just make a small change in the above code.

```
p=start;
for(i=1; i<k-1 && p!=NULL; i++)
   p=p->link;
```

We changed k to k-1 and this will give us pointer to predecessor of kth node. We will need this pointer when we will insert a new node at the kth position in the list.


start
p
1
2
3
4
5