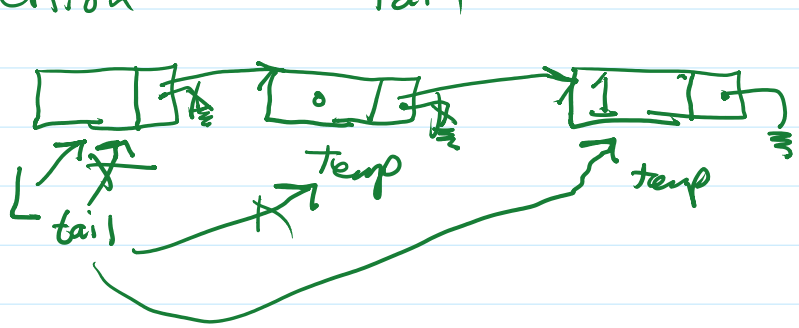
*Insertion



* check if last node

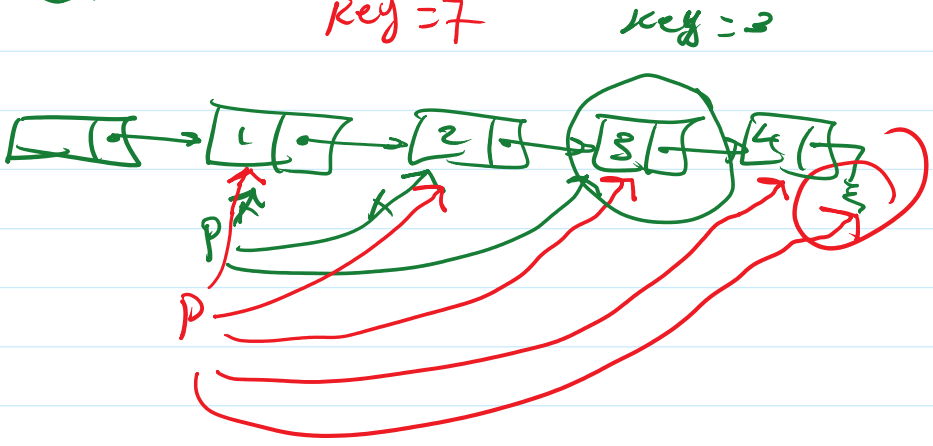
```
int isLast(Position P){
   return (P->next == Null);
}
```

* check if list is empty

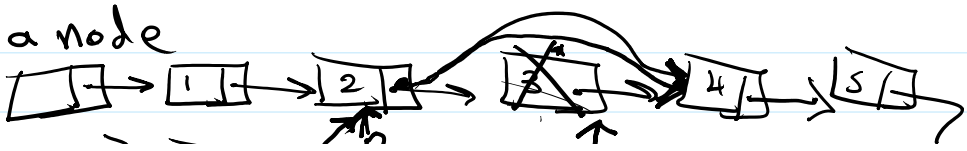
```
int isEmpty(List L){
   return (L->next == Null);
}
```

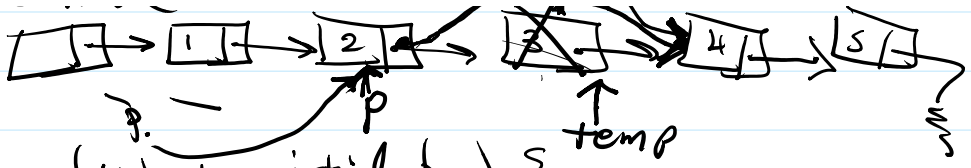
* Searching

```
Position Find(List L, int key){
   Position P=L->next;
   while(P!=Null && P->element != key){
      P = P->next;
   }
   return P;
}
```



*Delete a node

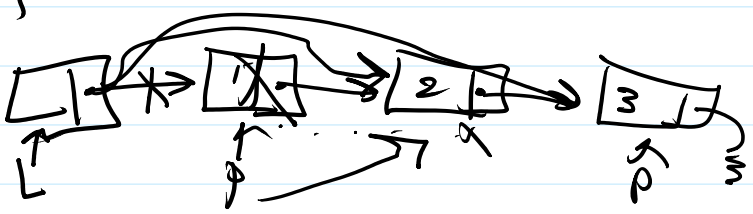





```
void delete (list L, int data) {
   Position p, temp.
     p = L;
     while (p->next != Null && p->next->data != data)
          p = p->next;
     if(p->next != Null) {
       temp = p->next;
       p->next = temp->next;
       free(temp);
     }
}
```

ايجاد العقدة السابقة

Best O(1)
worst O(n)
Avg O(n)



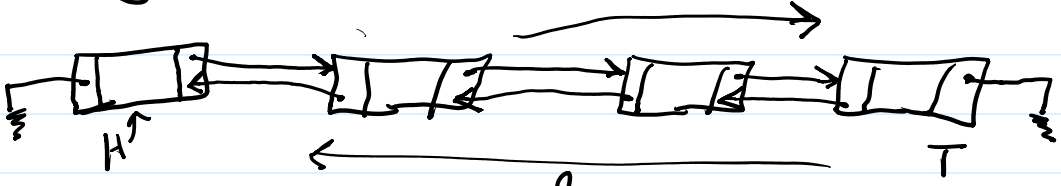
1000 node

```
void deleteList( List L) {
   Position p = L->next;
    while(L->next != Null) {
      L->next = p->next;
      free(p);
      p = L->next;
   } }
```

forward

* All concepts related to Singular Linked List.

* Doubly Linked List

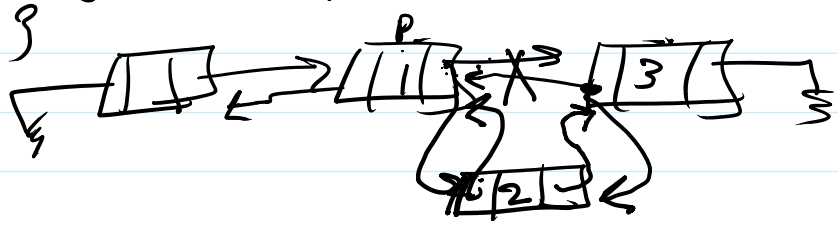


NB/ we can Access the node in two directions; forward & backward

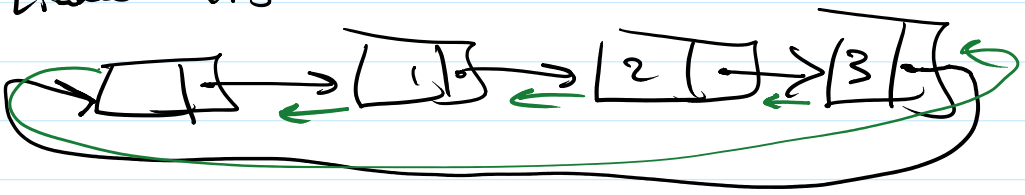
dis: extra space.

```
struct node{
    int element;
    Ptr next;
    Ptr previous;
```

```
void insert_DLL ( List L, Position p, int data){
    Position temp = (pos...) malloc( ___ ___ ___
    if (temp != NULL ){
        temp -> next = p -> next;
        p->next -> previous = temp.
        p->next = temp;
        temp -> previous = p;
    }
}
```



* circular Linked list



* circular Doubly linked