Linked List

Self-referrential data type

It has one or more pointers that points to the same types of data-types.

```
Struct student
    {
      int rollno           ———— 4 byte
      char name[20]        ———— 20 byte
      struct student* next ———— 8 byte.
    }
```

int id; — size 4

int A[10] — 10 X 4

Self-referrential data type is used to implement the dynamic data structure.

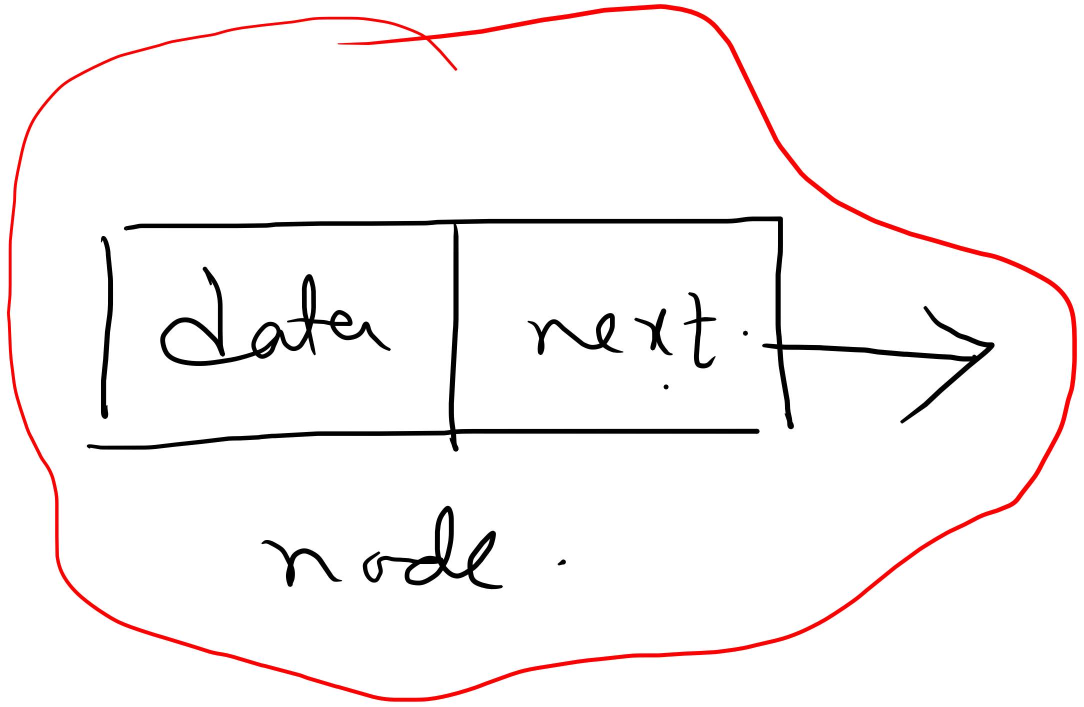
- linked list
- stack
- graph
- tree.

Linked list

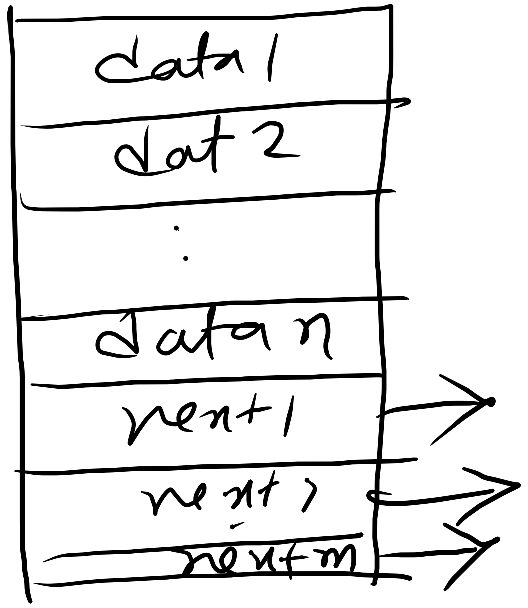
A ordered collection of finite, homogeneous data elements.

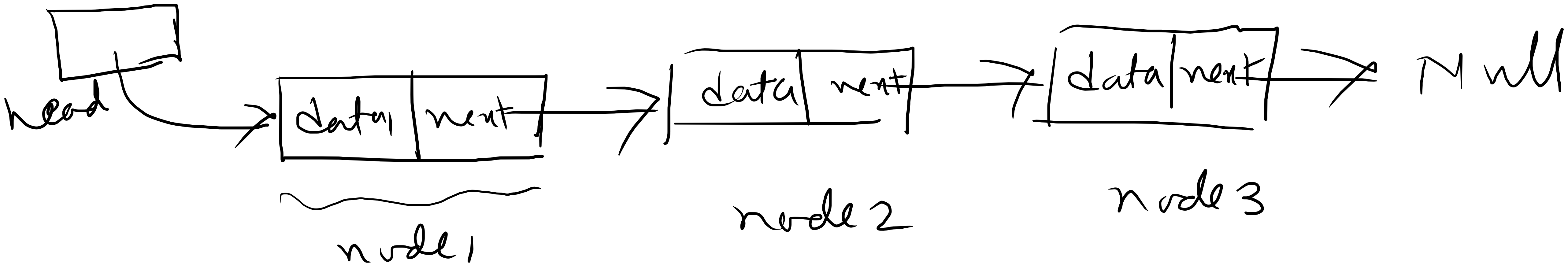
nodes

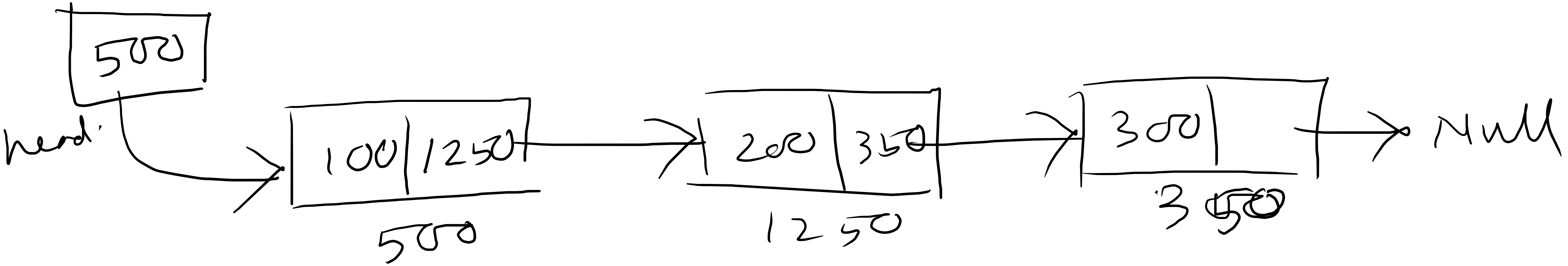
maintain a linear order



previous or next element can be defined. properly.

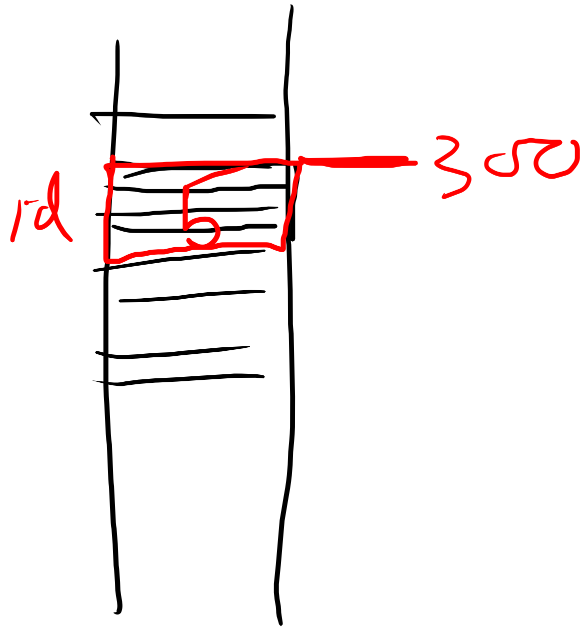



head
data
next
data
next
data
next
Null
node1
node2
node3

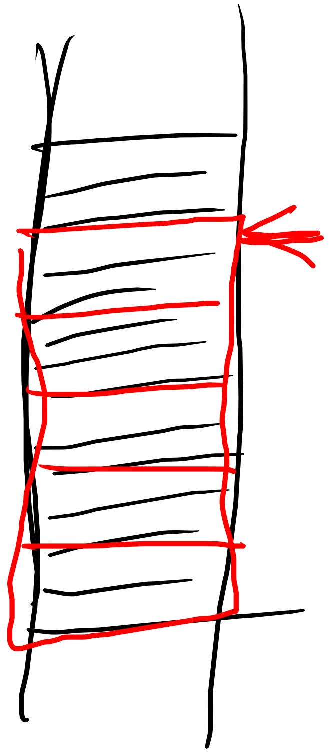

500
head
100
1250
200
350
300
Null
500
1250
350

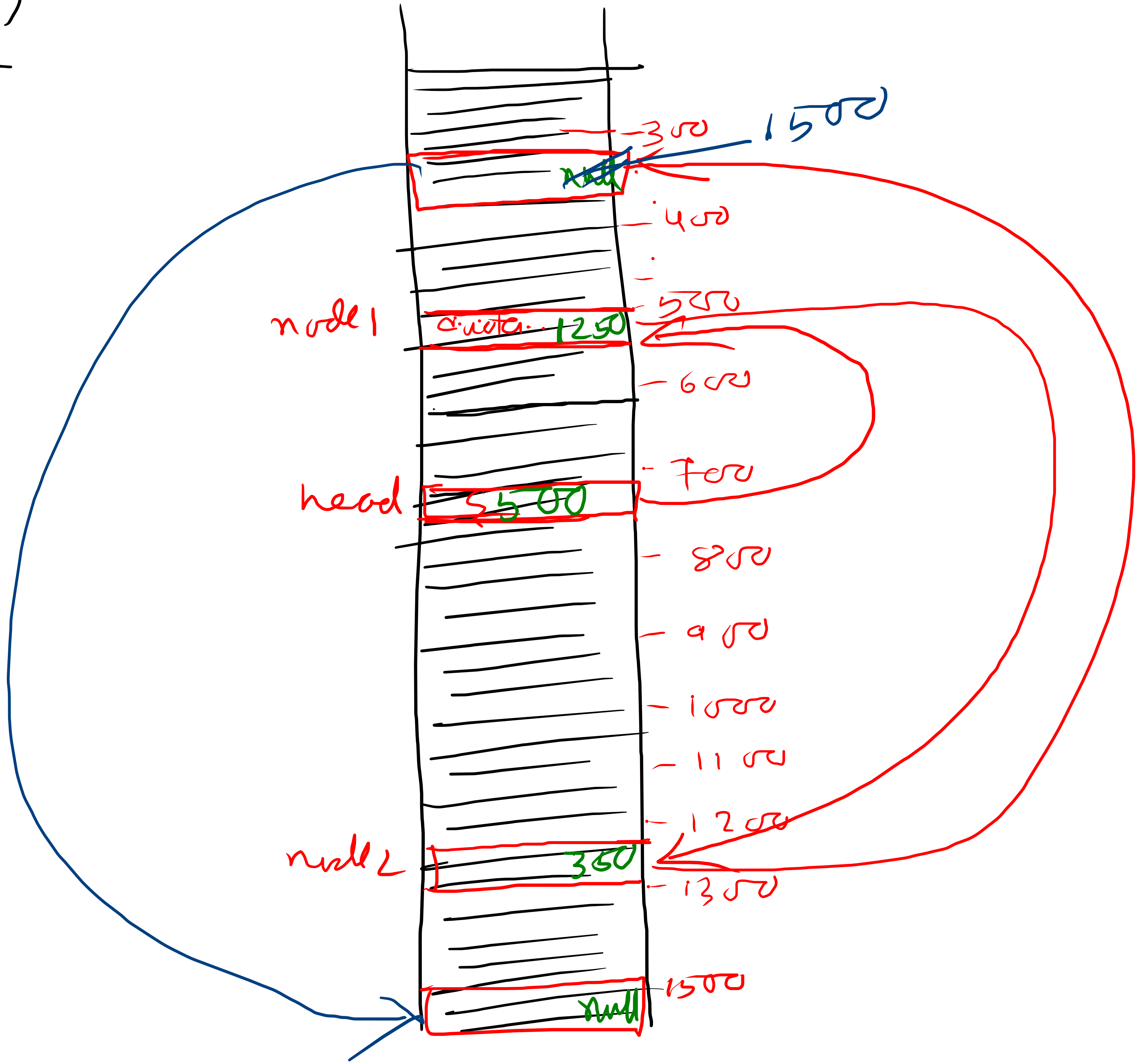
memory representation

int id=5



int A[5]





Possible operations performed on linked list

List-traversal (L) : Traversing element of the list L

List-count (L) : counting # elements in L

List_search (L, K) : searching for K in L

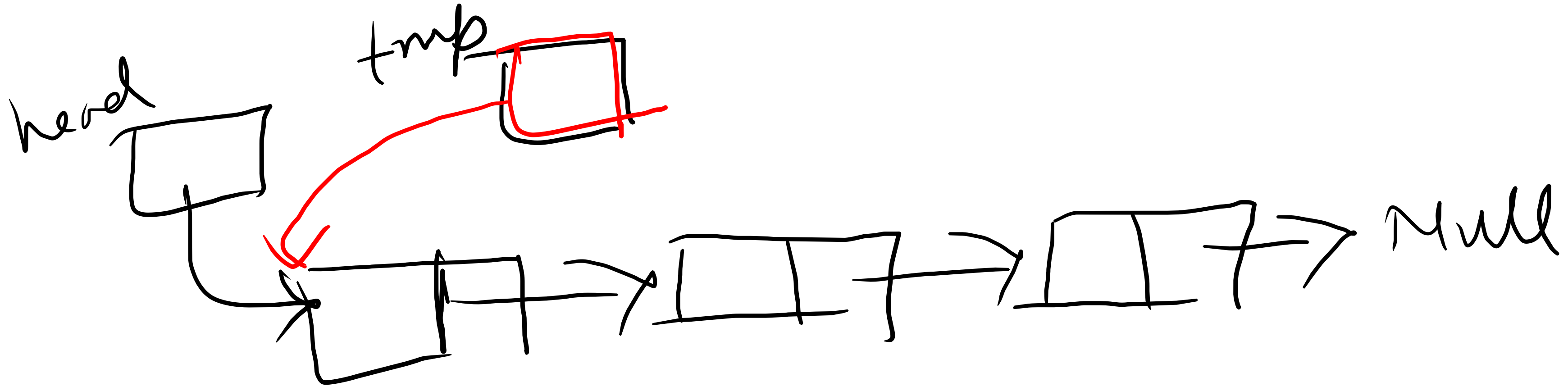
Insertion

- Insert_at-begining (L, K)
- Insert_at-the end (L, K)
- Insert_at_any-position (L, K, pos)

Deletion

- delete_at begining (L)
- " " end (L)
- " " any-position (L, pos)

Traverse:



```
List_traverse (L)

tmp = L.head
while tmp ≠ Null
    Process (tmp.data)
    tmp = tmp.next
```

Time: $O(n)$

Space: $O(1)$

count:

```
List_count (L)
tmp = L.head
count = 0
while tmp ≠ Null
        count = count + 1
        tmp = tmp.next

  return count.
```

Search

```
List-Search (L, K)
  tmp = L.head.
   while tmp ≠ Null
        if tmp.data == K
             return tmp

        tmp = tmp.next.
```

Time: $O(n)$

space: $O(1)$