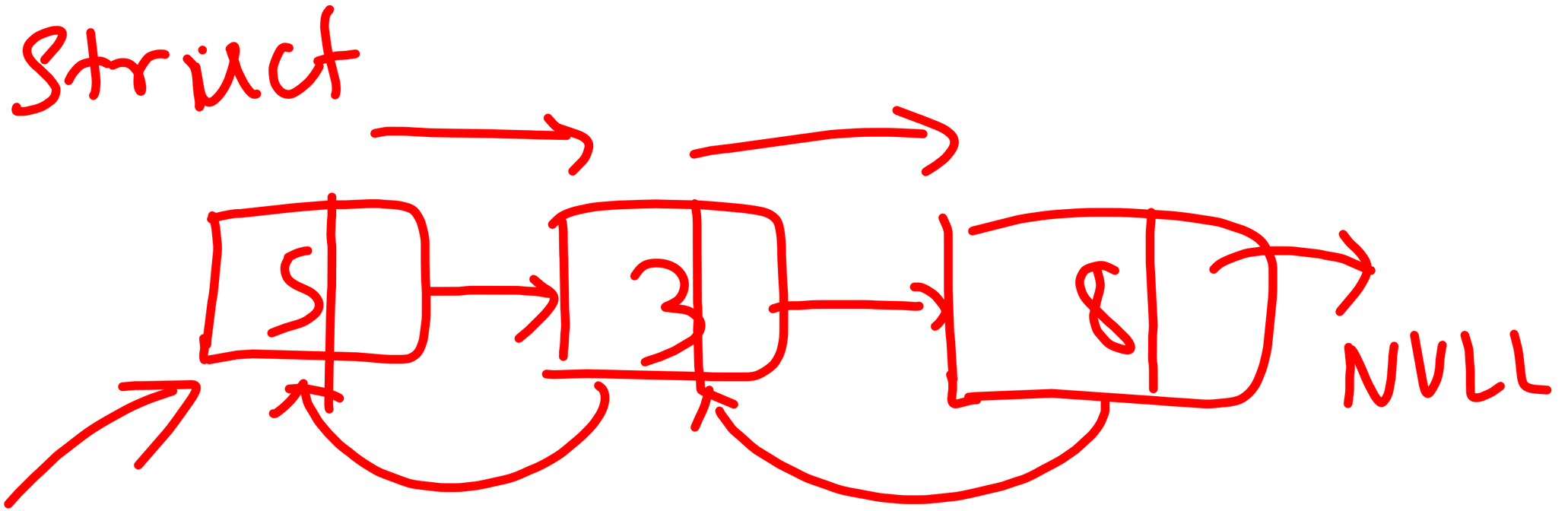
L = [ ]

L.append(x).

len(L)

$L_3 = L_1 + L_2$,



head Typedef struct node{ //Node

Node* head; int Val;

struct node* next;

}

1. Creat_list ( ){

Node * head = NULL;

Return head

}

2. Add_element(Node** head, int x){

if* head == NULL;

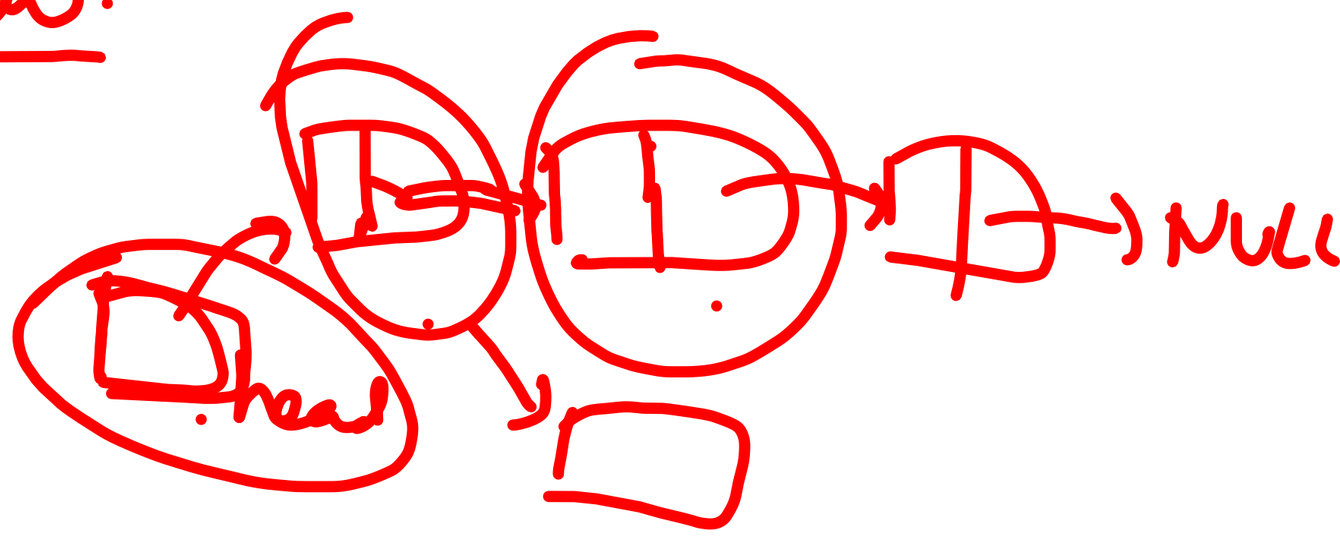
Node* new = (rear_node( ) .

(* new). val = x ; new→val = x ;

new→Next = NULL

* head = new.

}



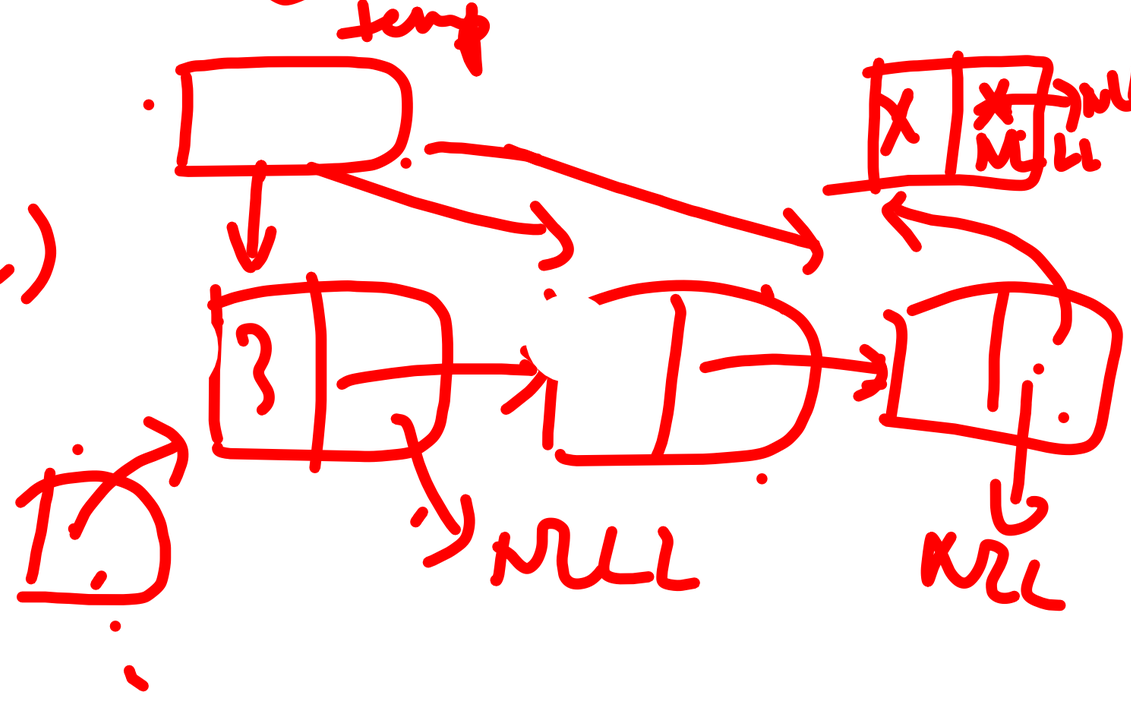
```
if head != NIL {
    temp = head;
    while (*temp).next != NULL)
        temp = (*temp).next
    }
    Node* new = (Node*)malloc()
    (*new).val = x
    (*new).next = NIL
    (*temp).next = new
}
```

① Move to Last

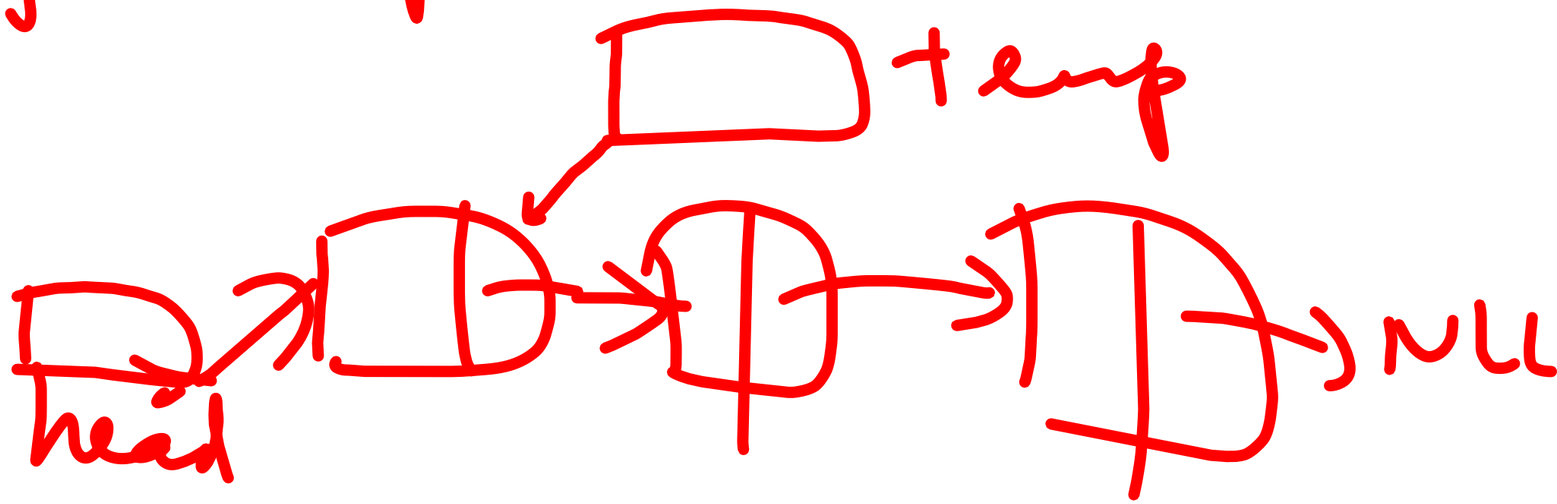
② Create new.

③ put value x

④ Keep new Link



```
show_list( Node * head ) {
    temp = head
    while (temp -> Next != NULL) {
        printf("%d", temp->val);
        temp = temp -> Next
    }
}
```



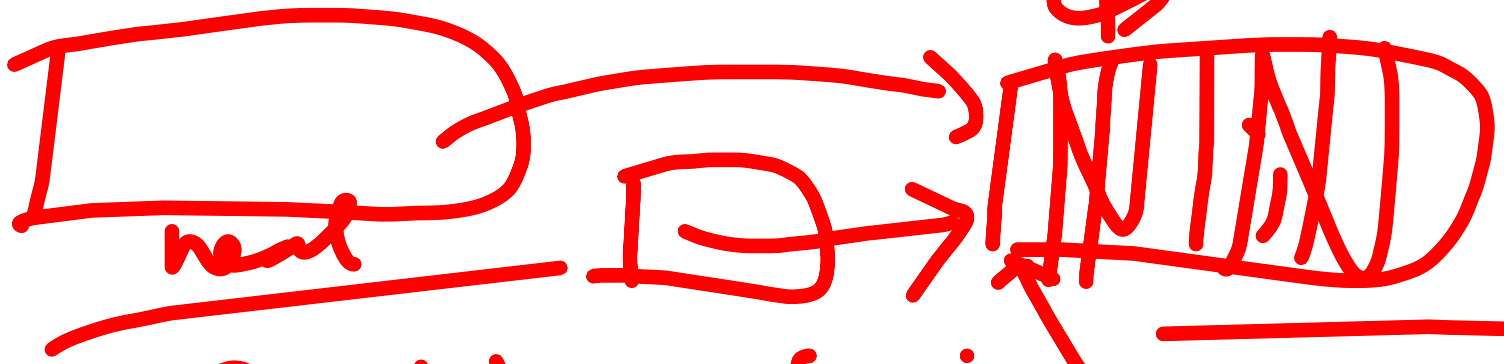
Node * int → 32-bit

int* 3 bit

Node * head = (Node *) malloc(

Node * head = (~~int *~~) malloc(2 * sizeof(int));

2 * 4



char * s = head

s[0]

32-bit

int * a = head s[7]

a[0]

Node * → 32-bit
int index

Node *head = (Node*) malloc(1

Node * head = (int*) malloc(2 + sizeof(int));
2 * 1

next
32-bit

[N|N]

char *c = head
s[0]

int *a = head
a[0]
s[7]

Node* head = (Node*) malloc(sizeof(Node));

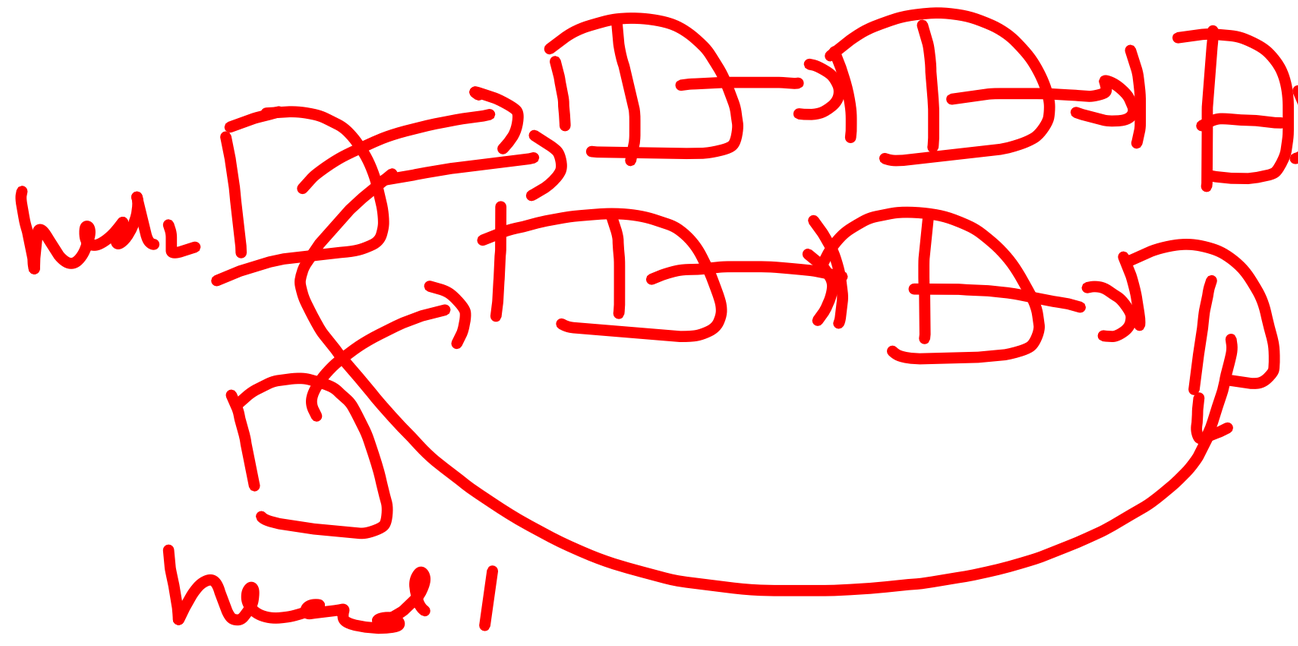
struct Node {
    int val;
    Node* next;
};

32-bit

64-bit
8 bytes

=

Concatenate_List (head1
head2)



pop()

temp = head

head = temp->next

Return temp