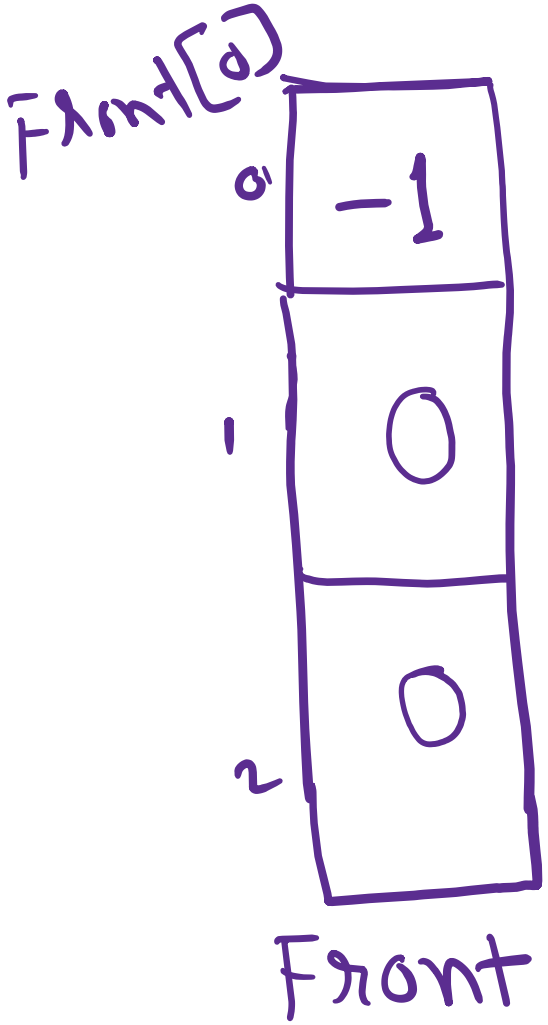
Multilevel Queue:

-1 + 1 = 0



Q1 Level 1 Queue

0	1	2	3
~~10~~	~~20~~	~~30~~	~~40~~

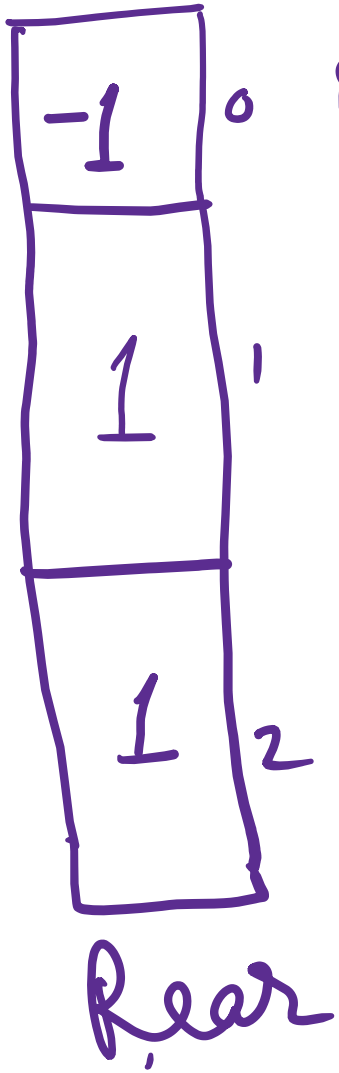
Q2 Level 2 Queue

0	1	2	3
40	50		

Q3

0	1	2	3
60	70		

Level 3 Queue



Insert 10 level 1 maxsize = 4

if (Rear[0] == maxsize − 1)

Rear[0] = Rear[0] + 1

Q1[Rear[0]] = 10

Q1[0] = 10

Front[0] = Front[0] + 1

Insert 20 level 1

Rear[0] = Rear[0] + 1

Q1[Rear[0]] = 20

Q1[1] = 20

Insert 30 level 1

Insert 40 level 1

Insert 40 level 2

Insert 50 level 2

Insert 60 level 3

Insert 70 level 3

Delete level 1

```
Write('Deleted element:', Q1[Front[0]])
Front[0] = Front[0] + 1
```

Delete level 1

Delete level 1

Delete level 1

This scheme can be used to implement multiple queues.

This scheme can also be used to implement a priority queue

Priority Queue:

→ Each element has a priority

→ priorities are expressed using integers

→ ① Sometimes lower integers represent higher priority

② Sometimes lower integers represent lower priority

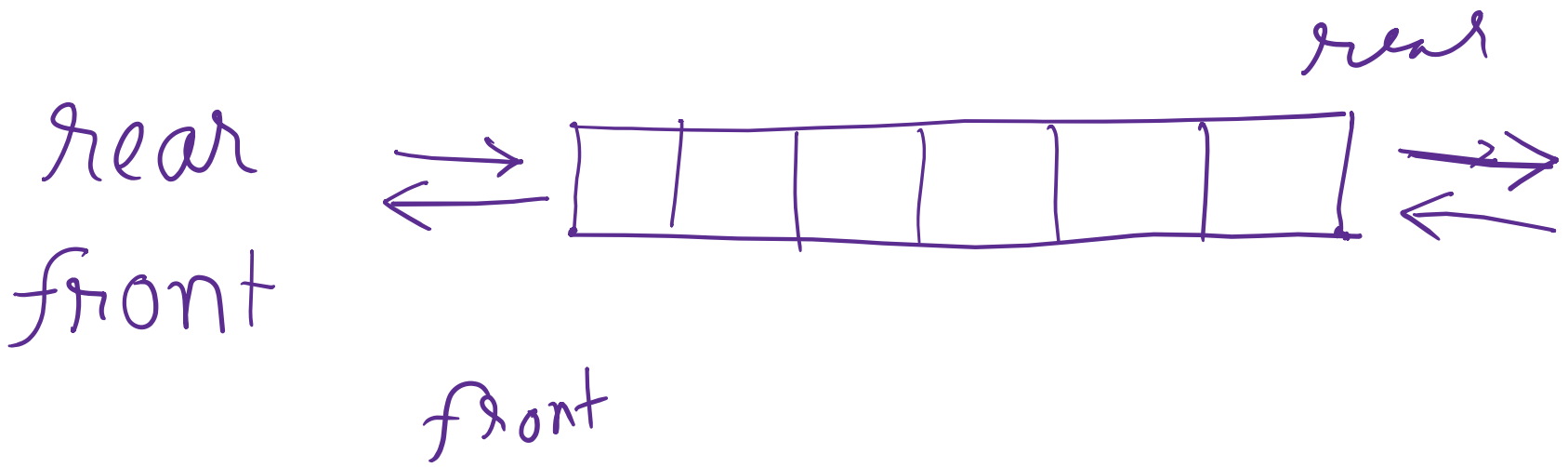
Element: A priority 0

B priority 1

In a priority queue, the element with the highest priority is deleted first.

Implementation of priority queue using multilevel queue

Double ended queue: insertions and deletions can be done at both ends of the queue



Linked Linear List (Linked list):

Static memory allocation ?

↳ memory allocated at compile time

```
int x;

x=10;
```

Dynamic memory allocation:

↳ memory allocated at run (execution) time

e.g. malloc function in C

```
int *p;
int q;
p = &q;
*p = 20;
```

q

*p

refer to the content of q by *p

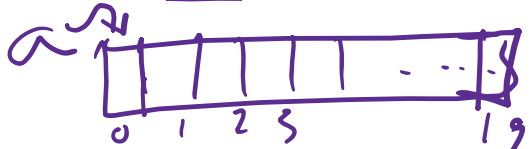
```
int *p;
int n;
printf("How many?");
scanf("%d",&n);
p=(int *) malloc(sizeof(int)*n);
p[0]=50;
```

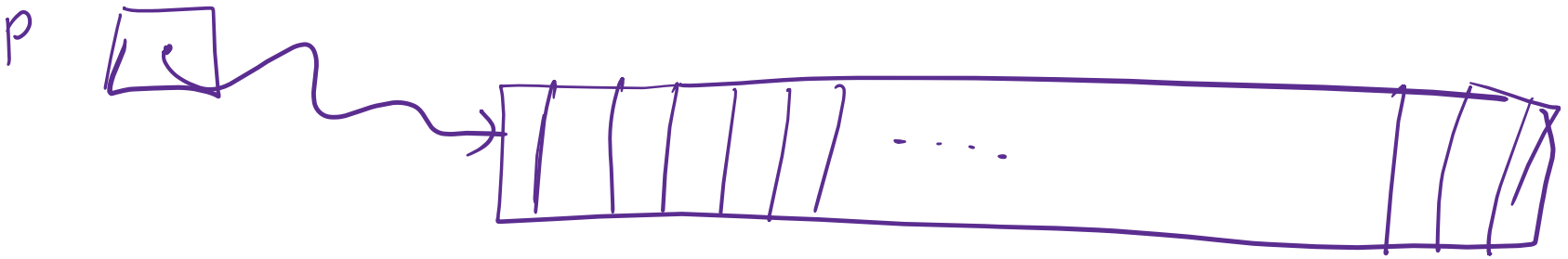
*a=a+1

*a=20
a[0]=20

int 4
n 10

int a[10];





Linked List: It is a linked representation of a linear list

e.g (10 20 30)

We have 'node's that are dynamically allocated

A node is a block of memory. Each such block has an address

Each node has two parts: → information part
                          → link part

field

Link field is a pointer field

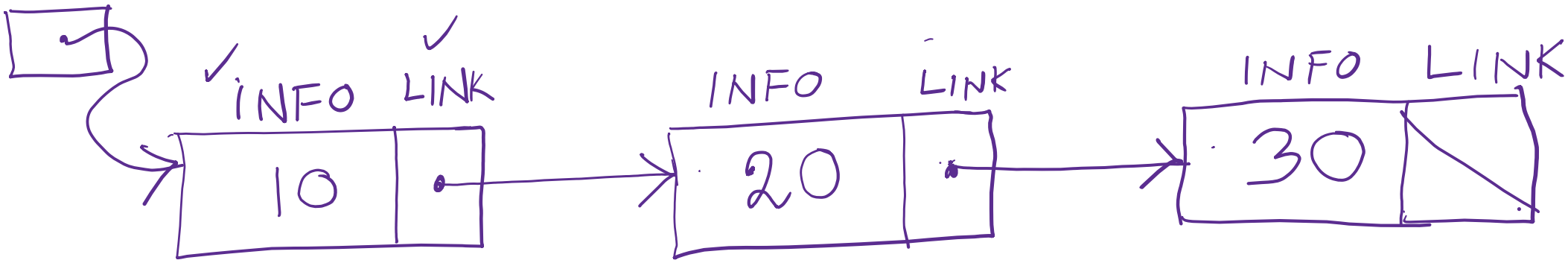
int x;
float y;

int *
float *

NULL

#define NULL -

START

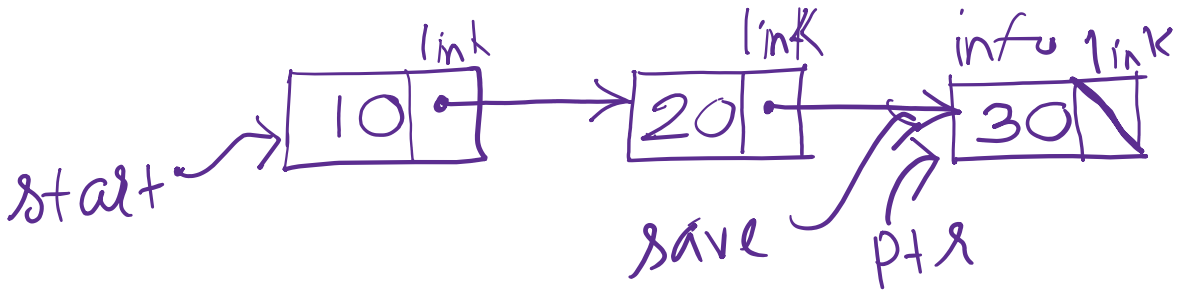


In order to access the content (fields) of a node, we must have its address

- In C language, each node can be declared as a structure

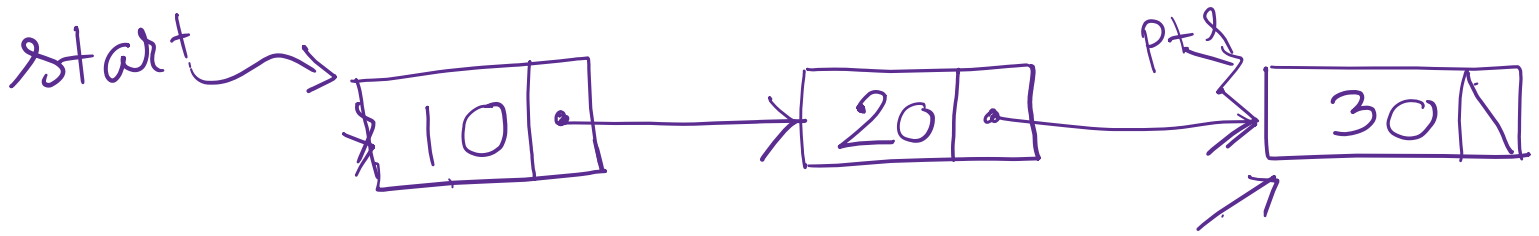
- Self-referential

```
struct node {
        int info;
        struct node *link;
        };
struct node *ptr, *start, *save;
ptr = (struct node *) malloc(sizeof(struct node));
start = ptr;
save = ptr;
```



```
ptr -> info = 10;
ptr -> link = null;
ptr = (struct node *) malloc (sizeof (struct node));
ptr -> info = 20;
ptr -> link = null;
save -> link = ptr;
save = ptr;
```

```
ptr = (struct node *) malloc (sizeof (struct node));
ptr -> info = 30;
ptr -> link = null;
save -> link = ptr;
save = ptr;
```



trav = null

```
int x, choice=1;
struct node *start, *ptr, *save, *trav;
printf("Enter information: ");
scanf("%d", &x);
ptr=(struct node *) malloc(sizeof(struct node));
ptr->info=x;
ptr->link=null;
start= ptr;
save= ptr;
```

10
20

```
printf("Continue?");
scanf("%d", &choice);
while (choice) {
    printf("Information: ");
    scanf("%d", &x);
    ptr = (struct node *) malloc(sizeof(struct node));
    ptr -> info = x;
    ptr -> link = null;
    save -> link = ptr;
```

```
save = ptr;
printf("Continue? ");
scanf("%d", &choice);
}
```

-999

---

* Create a singly linked linear list where each node stores an integer. The entry stops when -999 is entered.

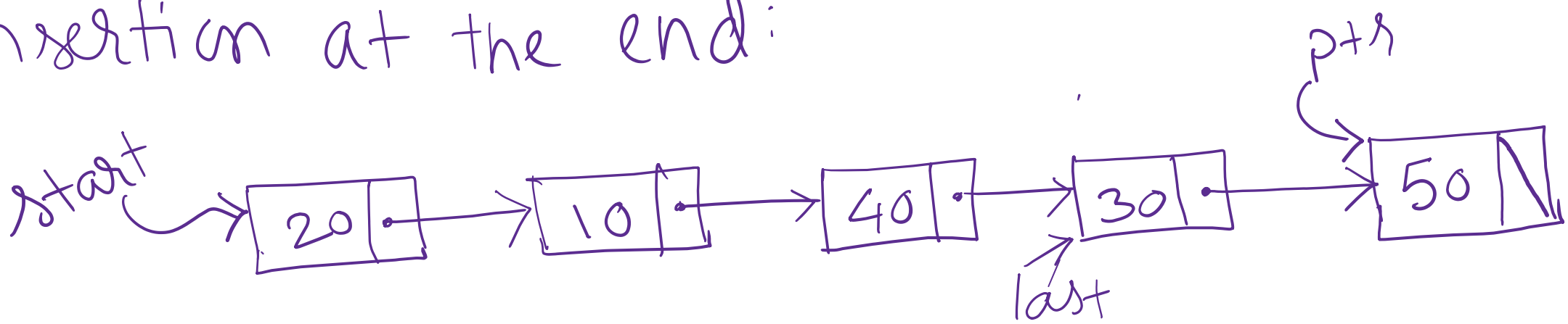
Traversing the list:

```
trav = start;
while (trav != null ) {
    printf("%d\n", trav -> info );
    trav = trav -> link;
}
```

10
20
30

Insertion of nodes in a linked list:

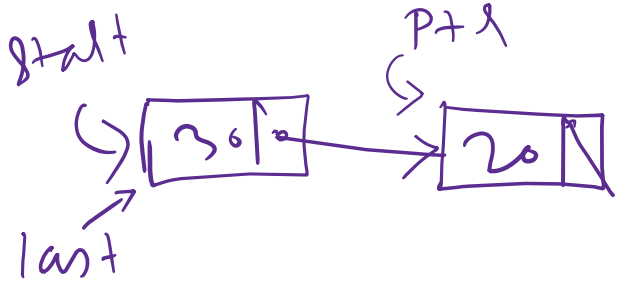
1) Insertion at the end:



How many addresses to be found?

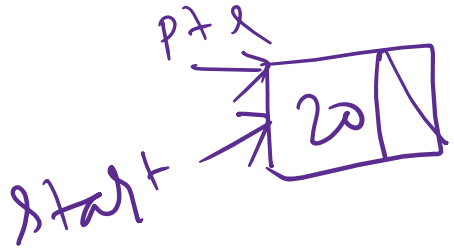
Which pointers are to be modified?

```
struct node *last, *ptr;
last = start;
while (last -> link != null)
    last = last -> link;
```




```
ptr = (struct node *) malloc(sizeof(struct node));
printf("Enter new element: ");
scanf("%d", &ptr -> info);
ptr -> link = null;
```

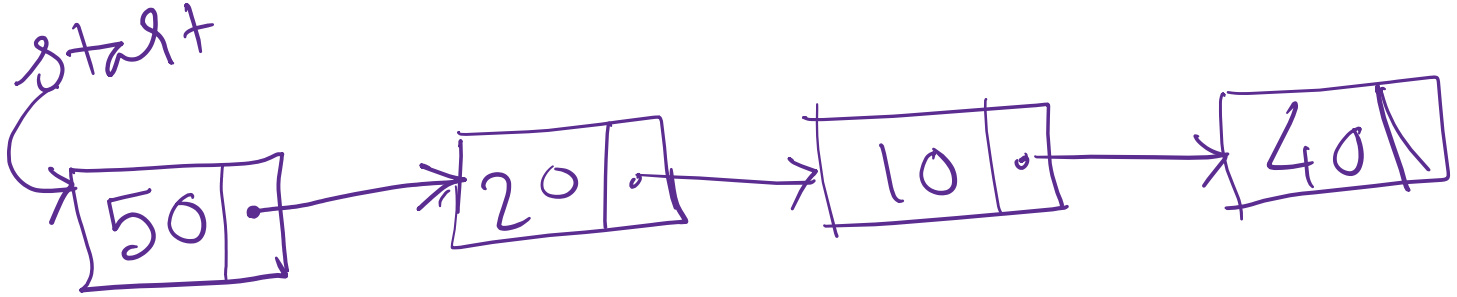
```
last -> link = ptr;
ptr = last = null;
```



Handling of empty list:

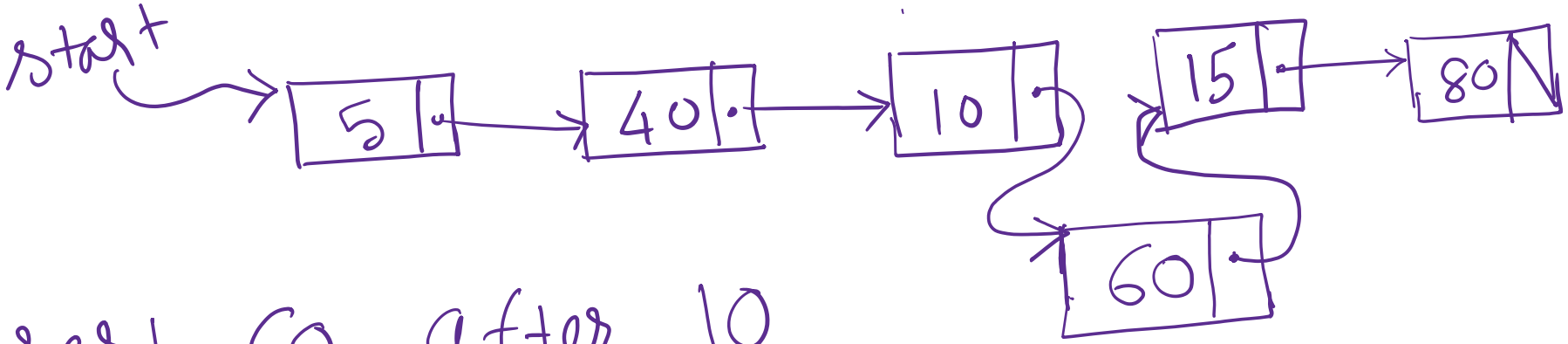
```
if (start == null) {
    ptr = ..... .
    ptr -> info = .....
    ptr -> link = null;
    start = ptr; }
else {                }
```

2) Insertion at the beginning:




```
struct node *ptr;
ptr=(struct node *) malloc(sizeof(struct node));
scanf("%d", &ptr->info);
ptr->link = start;
start = ptr;  ptr = null;
```

3) Insertion after a given node:



Insert 60 after 10

```
ptr = (struct node *) malloc(sizeof
                             (struct
ptr->info = 80;
next = trav->link;
```

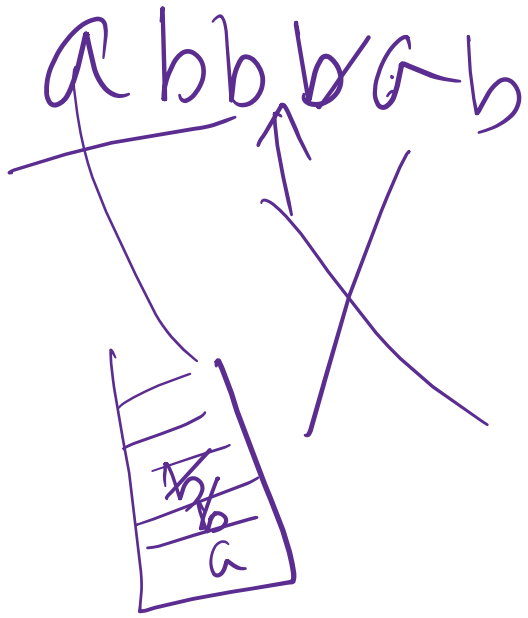
```
struct node *trav, *next;
trav = start;
while ( trav->info != 15)
    trav = trav->link;
```

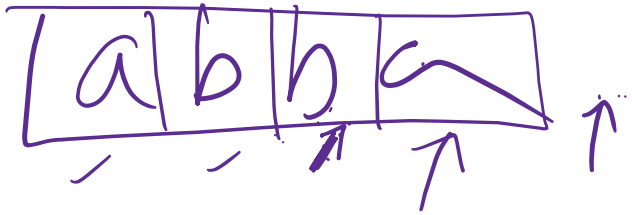
```
trav->link = ptr;
ptr->link = next;
```

```
trav = ptr = next = null;
```

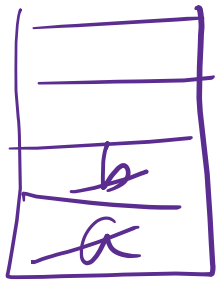
Applications of Stack:

palindromes





malayalam



Balanced Parentheses

✓ ((( )))

✗ ((( )

✓ Asymptotic Complexity

✓ Stack

✓ Queue

Insertion: `if (start == null) { }`
`else { }`

Element not present in the list:

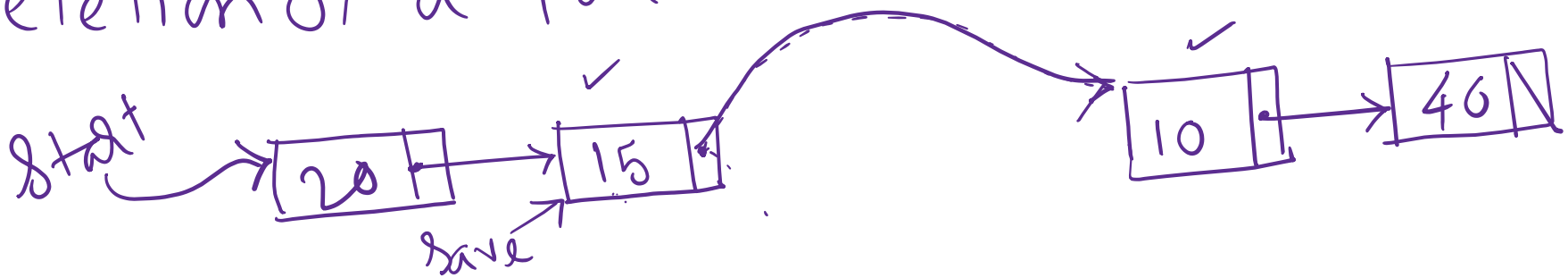
```
:
while (trav -> info != x)
  { trav = trav -> link;
    if (trav == null) break;
  }
if (trav != null) { ..... }
else { printf("Element not found!"); }
```

```
:
while (trav -> info != x && trav != null)
    {trav=trav->link;}
if (trav != null) { ..... }
else { ..... }
```

short circuiting

Deletion of element:

1) Deletion of a particular element.



Delete 30 from the list

```
int x;
struct node *trav, *save=null;
printf("which element to delete?");
scanf("%d", &x);
```

free(...)

free (trav);

```
trav = start;
while ( trav -> info != x)  {
        save = trav;
        trav = trav -> link;
      }
save -> link = trav -> link;
trav -> link = null;
free(trav);
```

x = 30