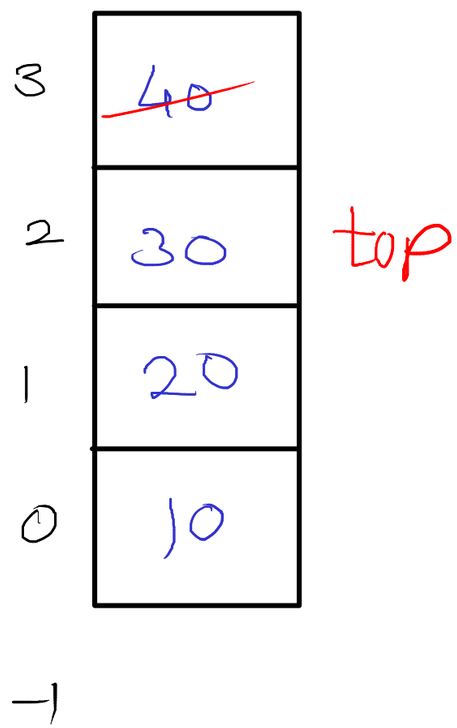
# Stack

- is a linear structure of similar data/elements
- insert and remove is allowed  from only one end (top)
- works on principle of "Last In First Out"/ (LIFO)

- top will always point to last inserted data




**Operations:**

1. **Add/Insert/Push:**
   a. reposition the top (inc)
   b. add data at topn index
2. **Delete/Remove/Pop:**
   a. reposition the top (dec)
3. **Peek/collect:**
   a. read/return data of top index

**Condition:**

1. **Empty**
   top == -1
2. **Full**
   top == SIZE-1

# Stack and Queue Time Complexity Analysis
# (Array Implementation)

| | Stack | Linear Queue | Circular Queue |
|---|---|---|---|
| Push | O(1) | O(1) | O(1) |
| Pop | O(1) | O(1) | O(1) |
| Peek | O(1) | O(1) | O(1) |

# Stack Application

## Expression Evaluation and Conversion

1. Postfix Evaluation
2. Prefix Evaluation
3. Infix to Postfix Conversion
4. Infix to Prefix Conversion

**Expression:**

- set/combination of operands and operators

  operands - values/variables

  operators - mathematical symbols (+, -, /, *, %)

e.g. a + b, 4 * 2 - 3

**Types:**

| | | |
|---|---|---|
| 1. Infix | a + b | human |
| 2. Prefix | + a b | computer |
| 3. Postfix | a b + | computer |

**Operators:**

()

power

* / %

+ -

# Postfix Evaluation

**Postfix : 4 5 6 * 3 / + 9 + 7 -**

**left** ⟶ **right**

**Result = 16**

5 $23 - 7 = 16$

4 $14 + 9 = 23$

3 $4 + 10 = 14$

2 $30 / 3 = 10$

1 $5 * 6 = 30$

**Stack**

| |
|---|
| |
| |
| |
| |
| |
| 16 |
| ~~7~~ |
| ~~23~~ |
| ~~9~~ |
| ~~14~~ |
| ~~10~~ |
| ~~3~~ |
| ~~30~~ |
| ~~6~~ |
| ~~5~~ |
| ~~4~~ |

# Prefix Evaluation

**Prefix : - + + 4 / * 5 6 3 9 7**

**left ← right**

**Result = 16**

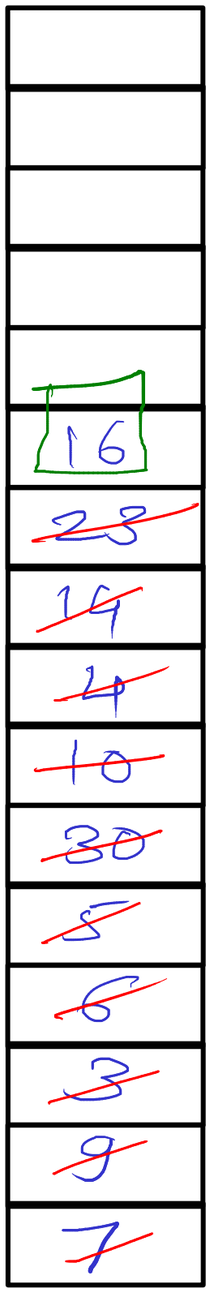
23 − 7
= 16

14 + 9
= 23

4 + 10
= 14

80/3
= 10

5 * 6
= 80

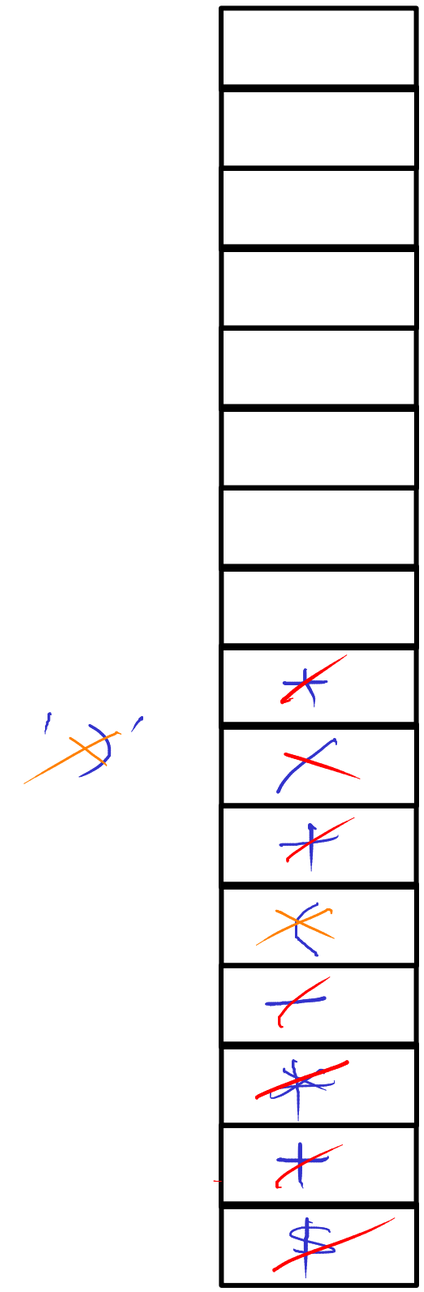


# Infix to Postfix conversion

**Infix : 1 $ 9 + 3 * 4 - (6 + 8 / 2) + 7**

**left** ⟶ **right**

**Postfix:** 19$34*+682/+-7+



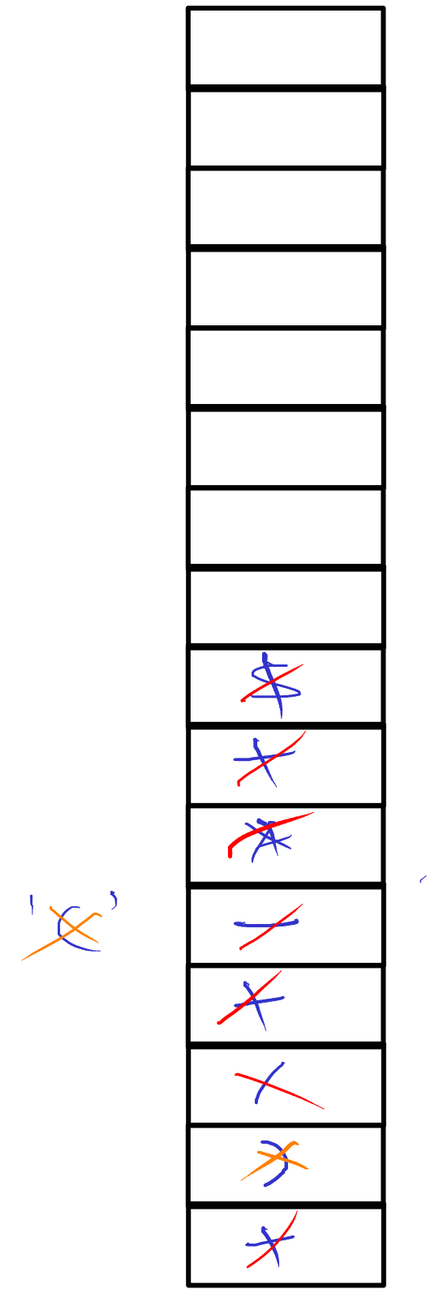
# Infix to Prefix conversion

**Infix : 1 $ 9 + 3 * 4 - (6 + 8 / 2) + 7**

**left** ← **right**

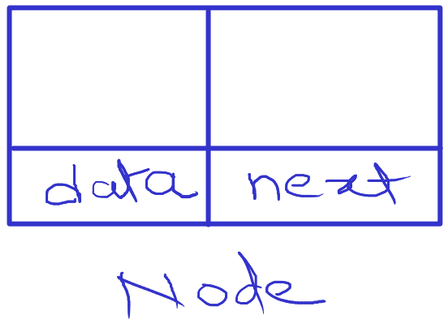
**Expression:** 7 2 8 / 6 + 4 3 * 9 1 $ + - +

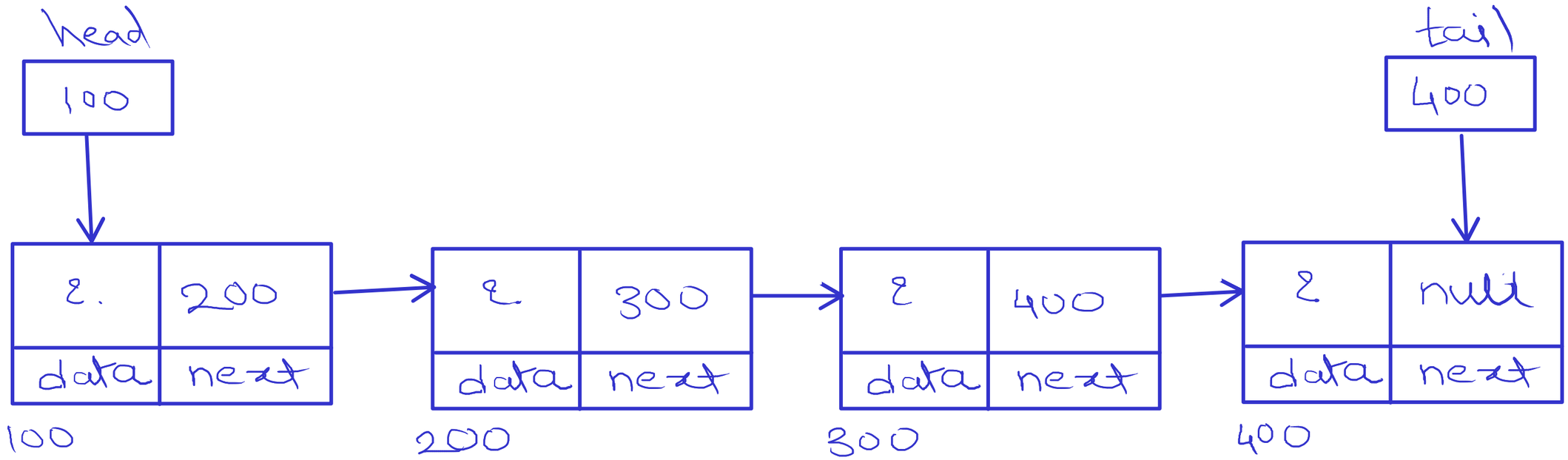
**Prefix:** + - + $ 1 9 * 3 4 + 6 / 8 2 7



# Linked List

- Linear data structure of similar element/ data
- address / link of next data is kept with current data
- single element of linked list is known as "node"
- Node consists of two parts:
    1. data
    2. link/address of next data
- address of first node is kept into head (pointer/referance)
- address of last node is kept into tail (pointer/referance) (optional)



## Linked list Operations

1. Add at first
2. Add at last
3. Add at in between (position)

4. Delete from first
5. Delete from last
6. Delete from in between (position)

7. Traverse (Display)

8. search
9. sort
10. reverse
11. mid

## Linked List Types:

1. Singly linear linked list
2. Singly circular linked list
3. Doubly linear linked list
4. Doubly circular linked list

```
class List{
        class Node{
                private int data;
                private Node next;
                public Node(){}
        }

        private Node head;
        private Node tail;
        public List(){}
        public isEmpty(){}
        public void add(data){}
        public void delete(){}
        public void display(){}
}
```