Data Structure →

1. Arrays → Normal Array ✓ / Array List ✓
2. LinkedList ✓
3. Stack (Idea) → Recursion (Application) / → Linked List as a Stack

peek
push
pop
size

Stack →
- ✓ (i) Usage. → STL (Inbuilt Stack functions)
- ✓ (ii) Application → Questions
- ✓ (iii) Implementation → Create Stack → with Array / linkedList
- (iv) Adapters. → Queue / Linkedlist

STL

st.push

st.pop

st.peek

Add / Remove

function → Adapter

→ functionality of Any kind of Data Structure.

Add → push

Remove → pop

Get → peek.

# next greater on Right

① Brute Force →

arr →

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 10 | 6 | 12 | 5 | 3 | 2 | 4 | 8 | 1 |

Step I → create an array. have initial value is arr.length / OR -1 depend on question,

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 12 | 12 | -1 | 8 | 4 | 4 | 8 | -1 | -1 |

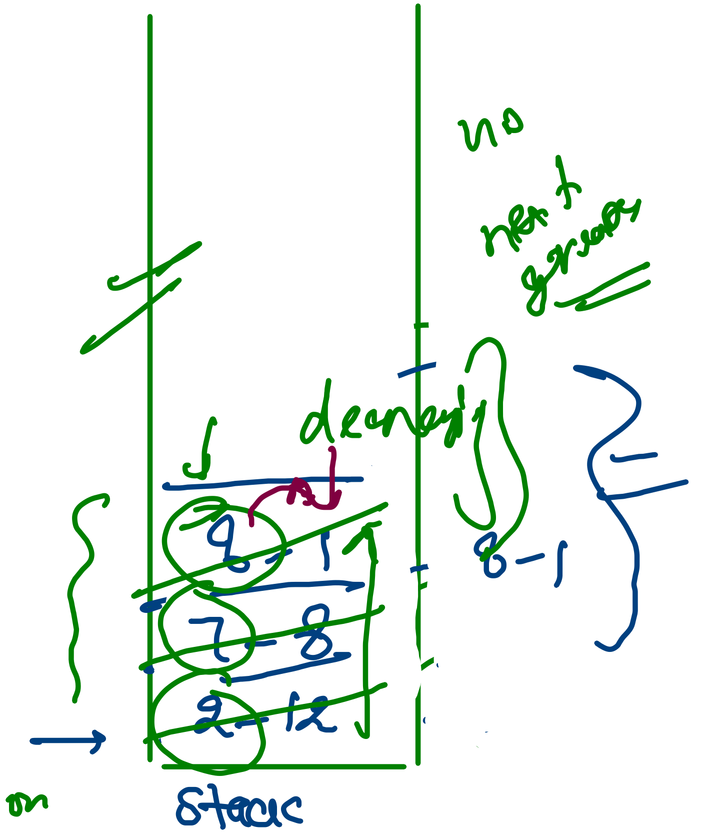
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 12 | 12 | -1 | 8 | 4 | 4 | 8 | -1 | -1 |

nge →

Step-II → Create Stack, Enter the Element in stack until top is greater than current entering element

Step-III → If top is smaller than current entering,

Step IV → Process remaining stack according to question requirement;



~~next~~ next greater on right → submit on portal.

~~next~~ next greater on left

next smaller on right

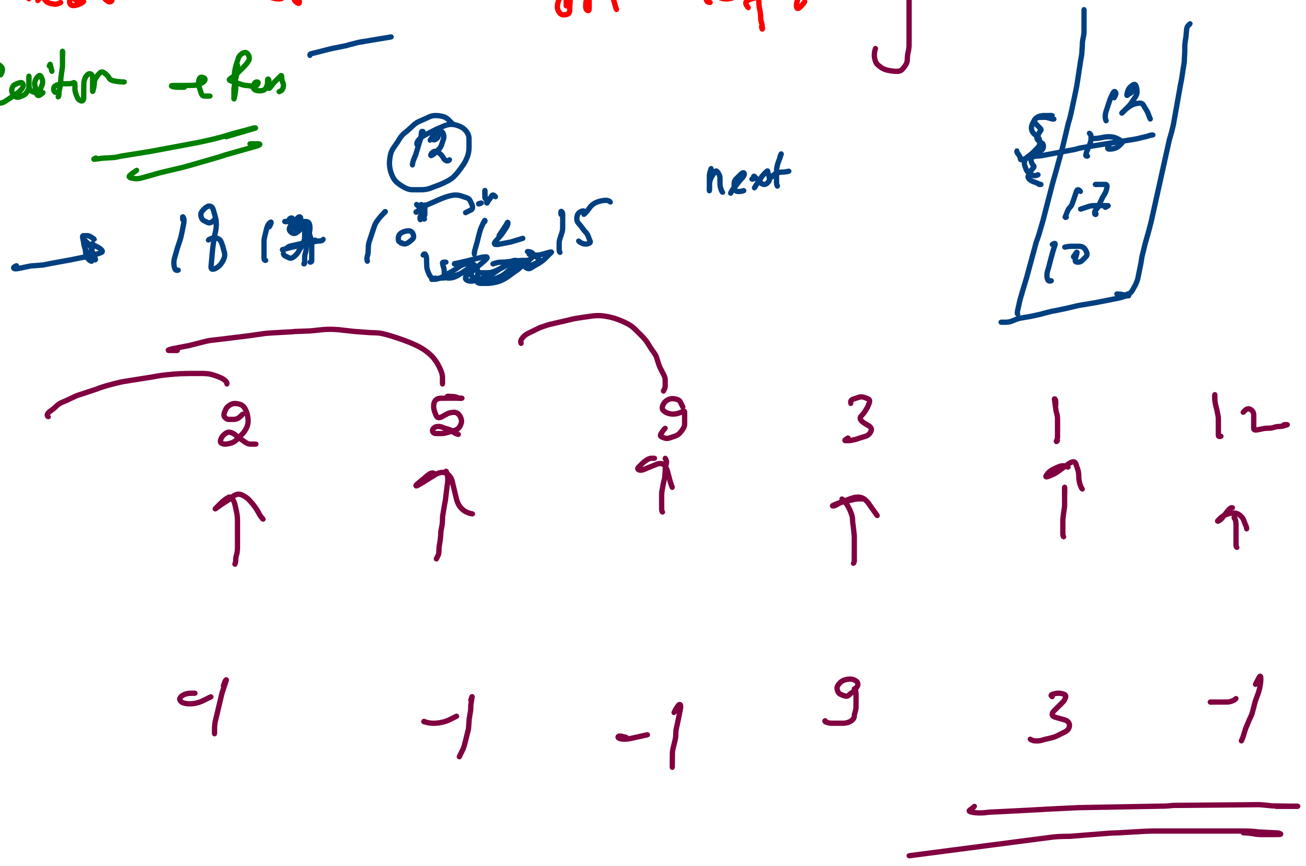
next smaller on left

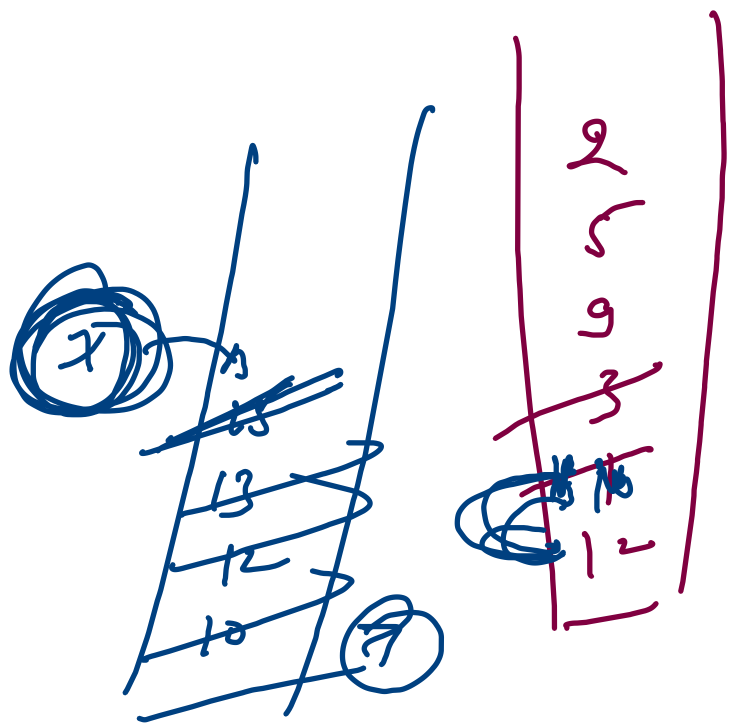
Loop && condition

Edition → fees

greater → Decreasing order maintain in Stack

Smaller → Increasing order





# Duplicate Brackets

$\{(c+d)]$

Case-I

$((a+b) + (c+d))$

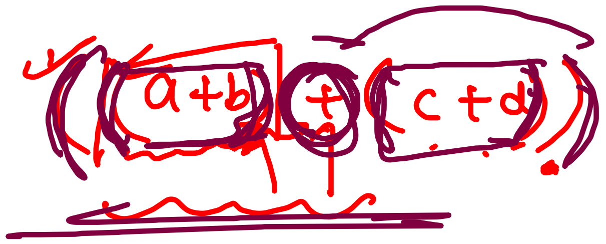
Relevant Bracket } → False

Case-II

$(a+b) + ((c+d))$

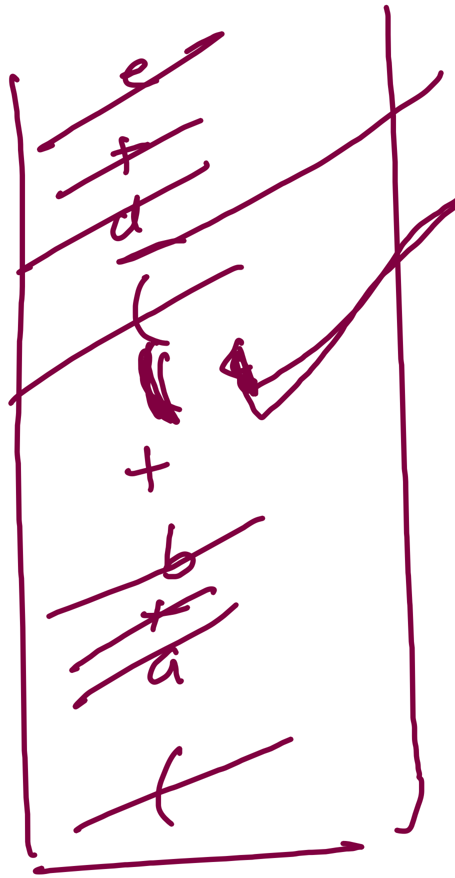
Irrelevant Bracket — Duplicate.

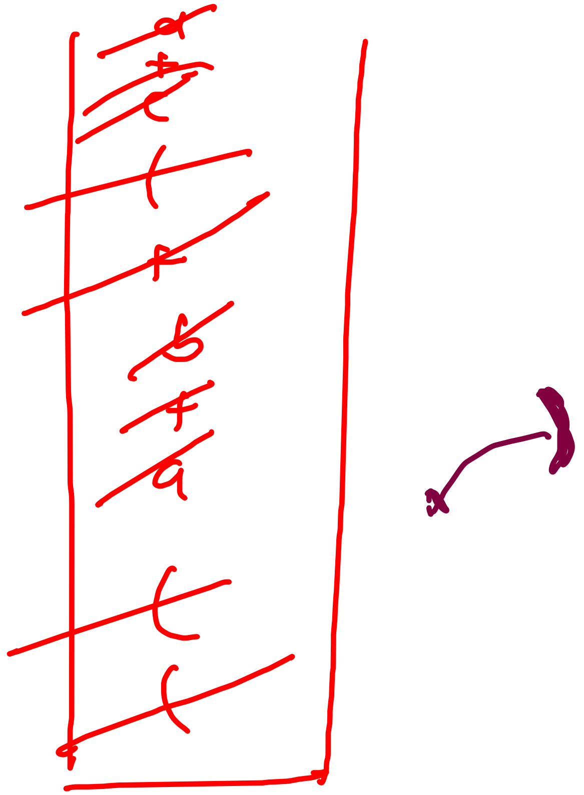
→ True } it have duplicate Brackets.

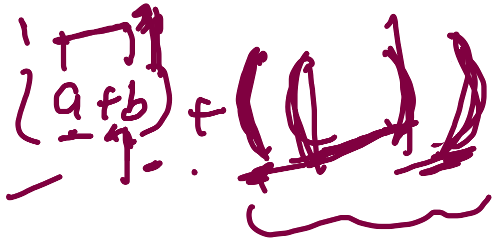


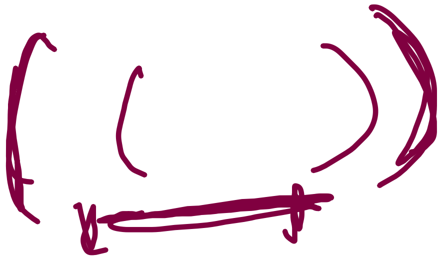
Duplicacy → False .

$a+b + (c+d)$





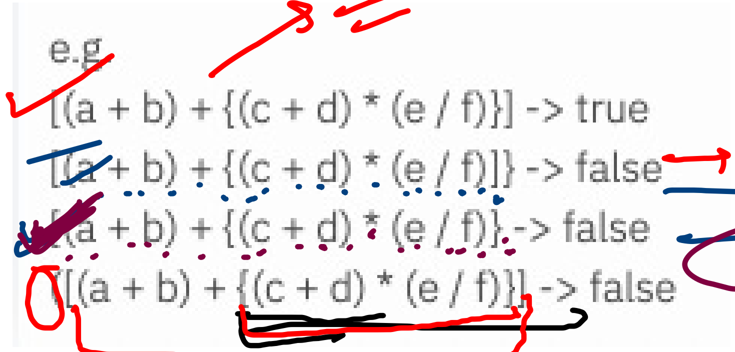




return False.

# Balanced Brackets →

True



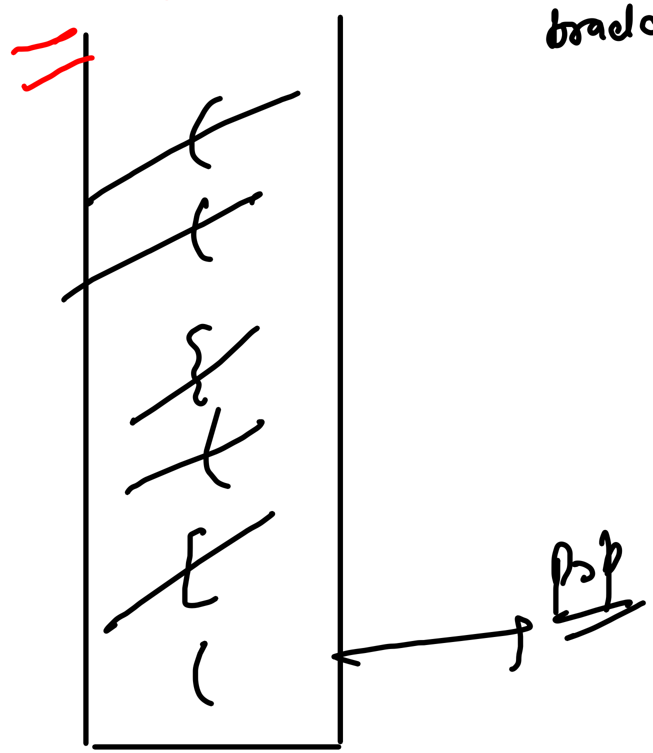
e.g

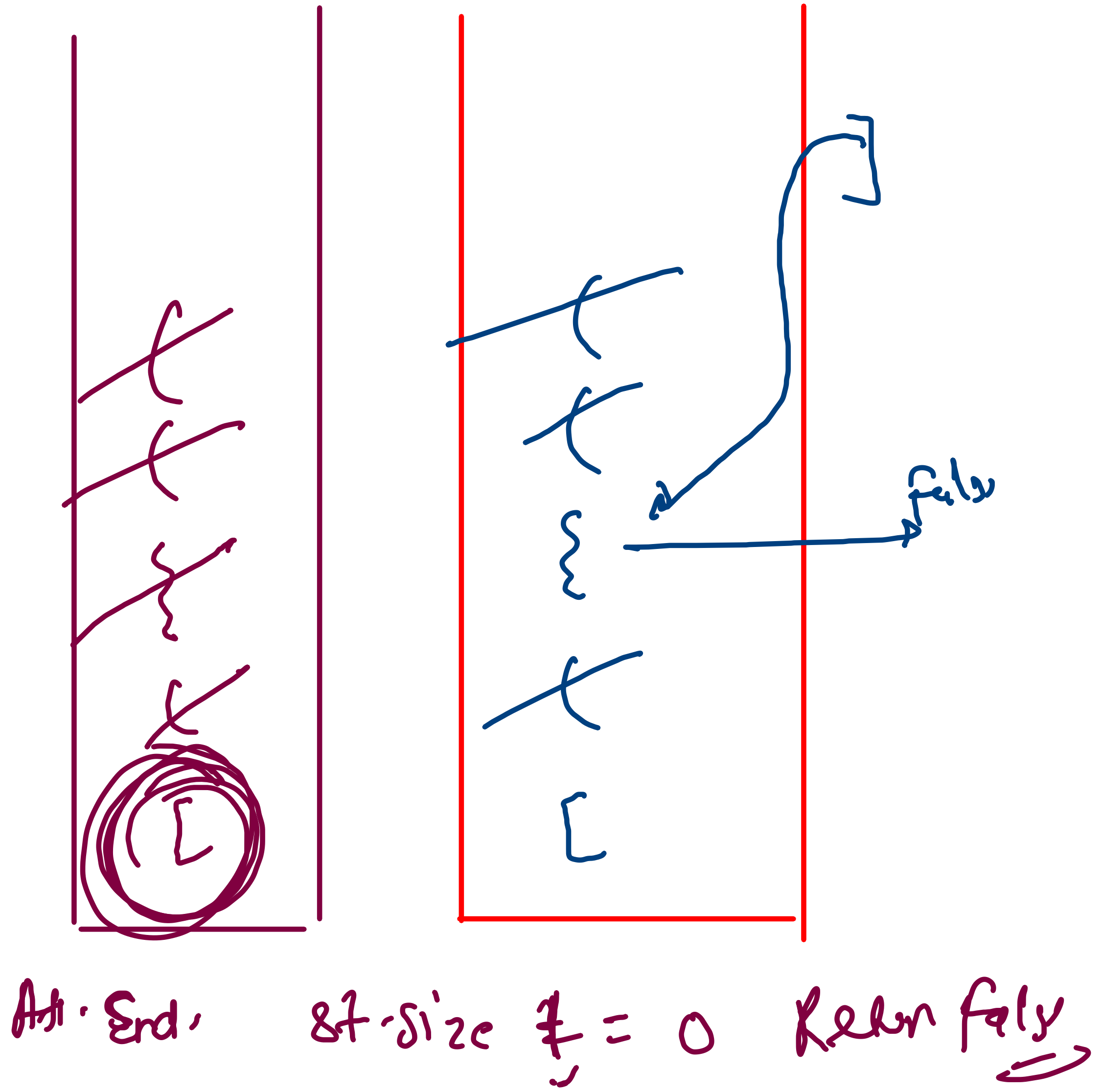
[(a + b) + {(c + d) * (e / f)}] -> true

[(a + b) + {(c + d) * (e / f)]} -> false → Bracket Mismatch,

(a + b) + {(c + d) * (e / f)} -> false → closing less Bracket

([(a + b) + {(c + d) * (e / f)}] -> false → less opening brackets





charcter.      ch      [(a+b)                                        (a+b)]

Ch ==      opening      bracket → push

Ch ==      closing      bracket → check the top if similar opening present → pop.
                                                          else return false;
                                → ~~if~~ st.size()==0 → return false;

ch ==      character. → Nothing to do

After all check      if (st.size == 0) return true;

return st.size() == 0;