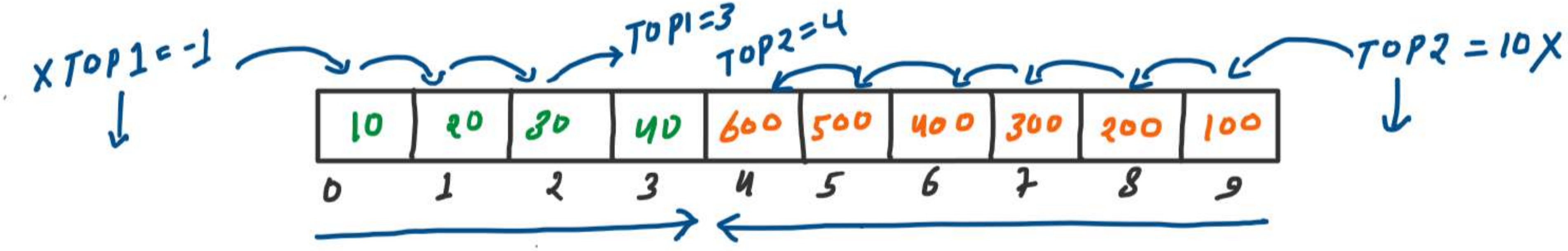
10/11/2023

STACK CLASS - 2

1. Implementation of two stack using a dynamic array



UNDERFLOW

```
if( TOP1 == -1)
    stack1 is Empty
```

```
if( TOP2 == size)
    stack2 is empty
```

Overflow

```
if( TOP2 - TOP1 == 1)
    No space Available
      in array
```

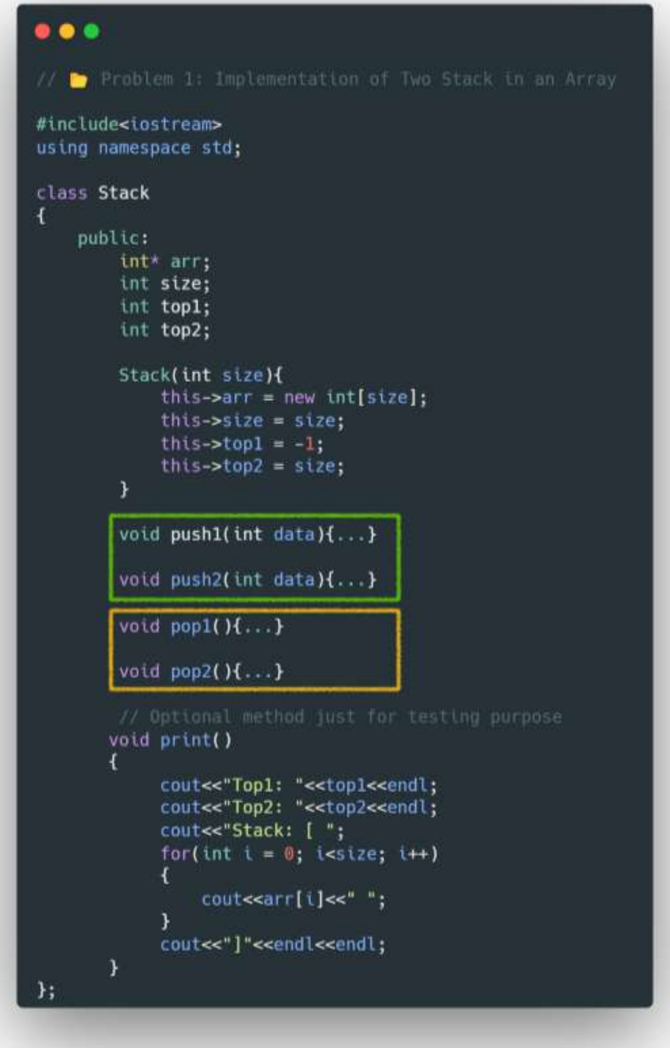
```
class stack
{
    public:
        int* arr;
        int size;
        int TOP1;
        int TOP2;
        ≡
}
```

```
push1() {
    TOP1 ++;
    arr[TOP1] = data;
}
```

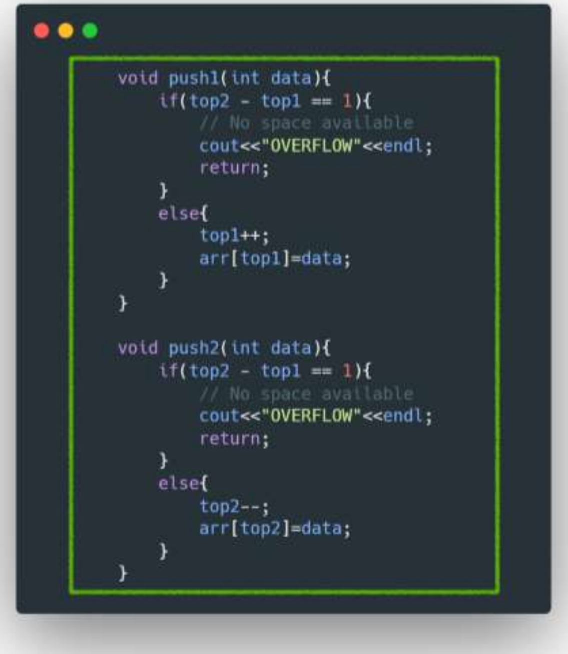
```
push2 () {
    TOP2--;
    arr[TOP2] = data;
}
```

```
pop1() {
    arr[TOP1] = 0;
    TOP--;
}
```

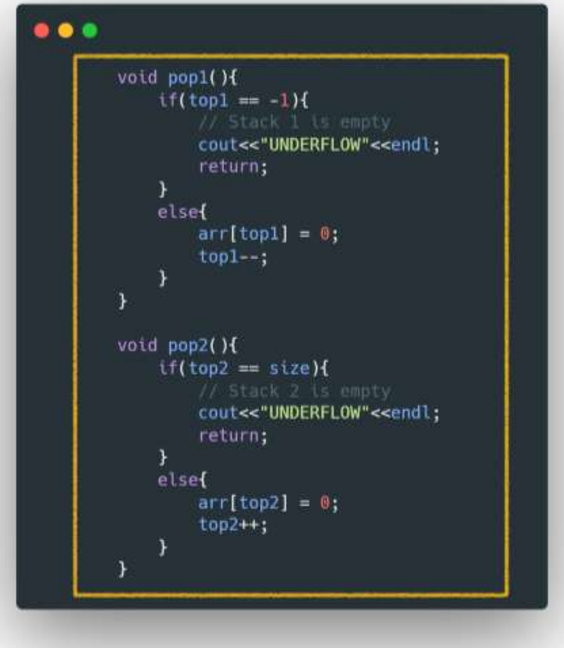
```
pop2() {
    arr[TOP2]=0;
    TOP ++;
}
```



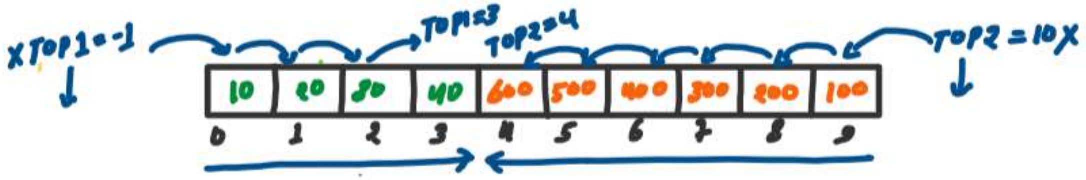
```cpp
// 📁 Problem 1: Implementation of Two Stack in an Array

#include<iostream>
using namespace std;

class Stack
{
    public:
        int* arr;
        int size;
        int top1;
        int top2;

        Stack(int size){
            this->arr = new int[size];
            this->size = size;
            this->top1 = -1;
            this->top2 = size;
        }

        void push1(int data){...}

        void push2(int data){...}

        void pop1(){...}

        void pop2(){...}

       // Optional method just for testing purpose
       void print()
       {
            cout<<"Top1: "<<top1<<endl;
            cout<<"Top2: "<<top2<<endl;
            cout<<"Stack: [ ";
            for(int i = 0; i<size; i++)
            {
                cout<<arr[i]<<" ";
            }
            cout<<"]"<<endl<<endl;
       }
};
```



```cpp
void push1(int data){
    if(top2 - top1 == 1){
        // No space available
        cout<<"OVERFLOW"<<endl;
        return;
    }
    else{
        top1++;
        arr[top1]=data;
    }
}

void push2(int data){
    if(top2 - top1 == 1){
        // No space available
        cout<<"OVERFLOW"<<endl;
        return;
    }
    else{
        top2--;
        arr[top2]=data;
    }
}
```



```cpp
void pop1(){
    if(top1 == -1){
        // Stack 1 is empty
        cout<<"UNDERFLOW"<<endl;
        return;
    }
    else{
        arr[top1] = 0;
        top1--;
    }
}

void pop2(){
    if(top2 == size){
        // Stack 2 is empty
        cout<<"UNDERFLOW"<<endl;
        return;
    }
    else{
        arr[top2] = 0;
        top2++;
    }
}
```



10	20	30	40	600	500	400	300	200	100
0	1	2	3	4	5	6	7	8	9

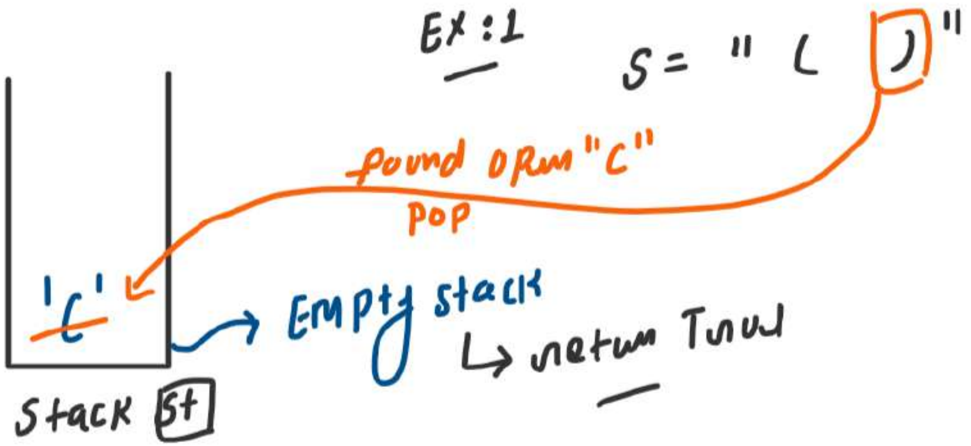
2. Valid Parentheses (Leetcode-20)

V.V.V.Imp.

```
Example 1:
Input: s = "()"
Output: true

Example 2:
Input: s = "(]"
Output: false

Example 3:
Input: s = "()[]{}"
Output: true

Example 4:
Input: s = "( () ( () ) )"
Output: true
```

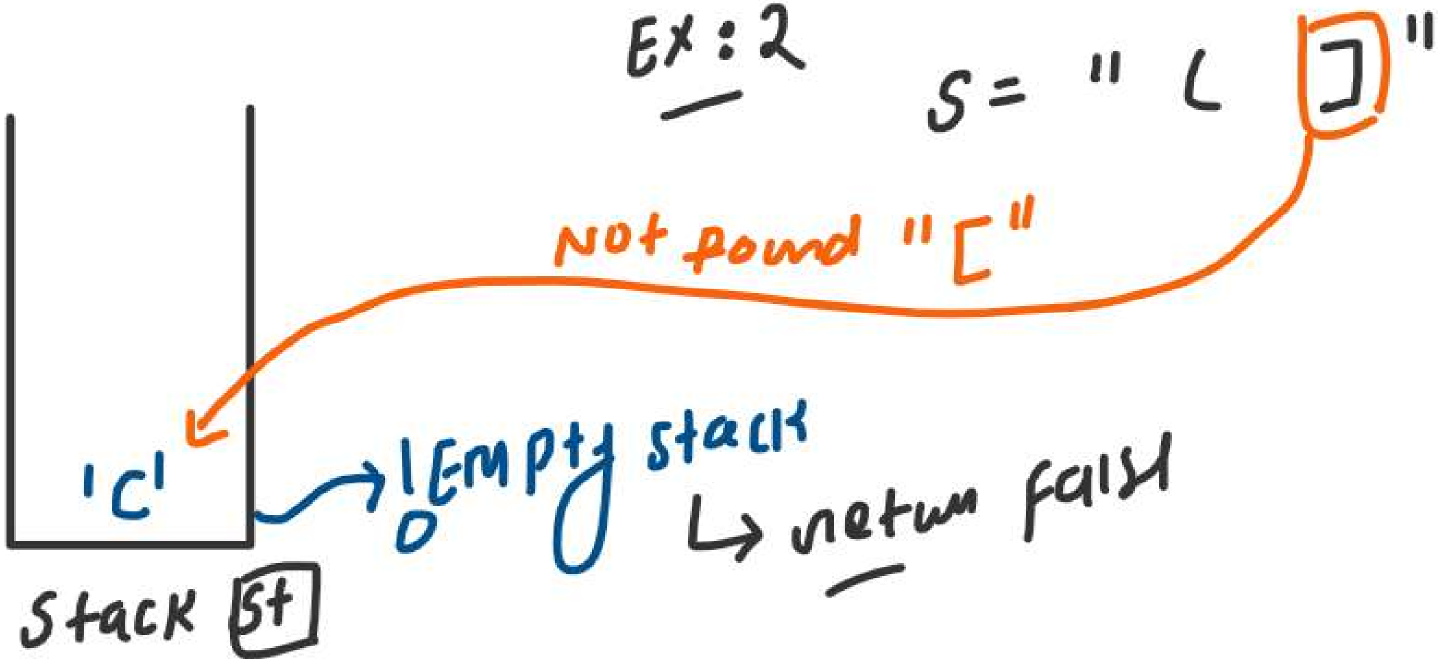
Open Brackets	Close Brackets
([{	)] }

DRY RUN



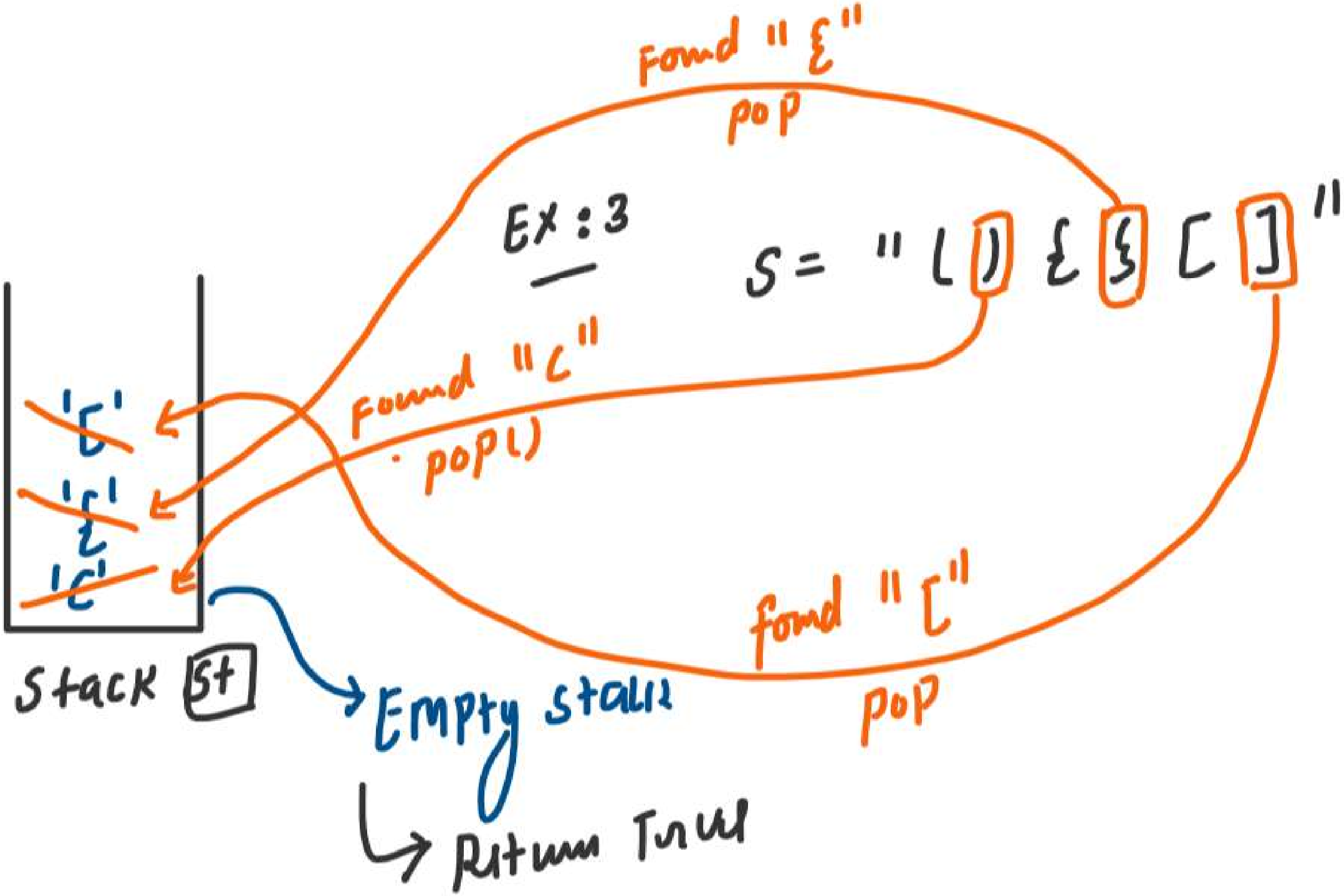
```
if( Open Bracket )
  ↳ St.push("(");
if( close && open)
  ↳ Find open Bracket
    St.pop();
```

Ex : 2

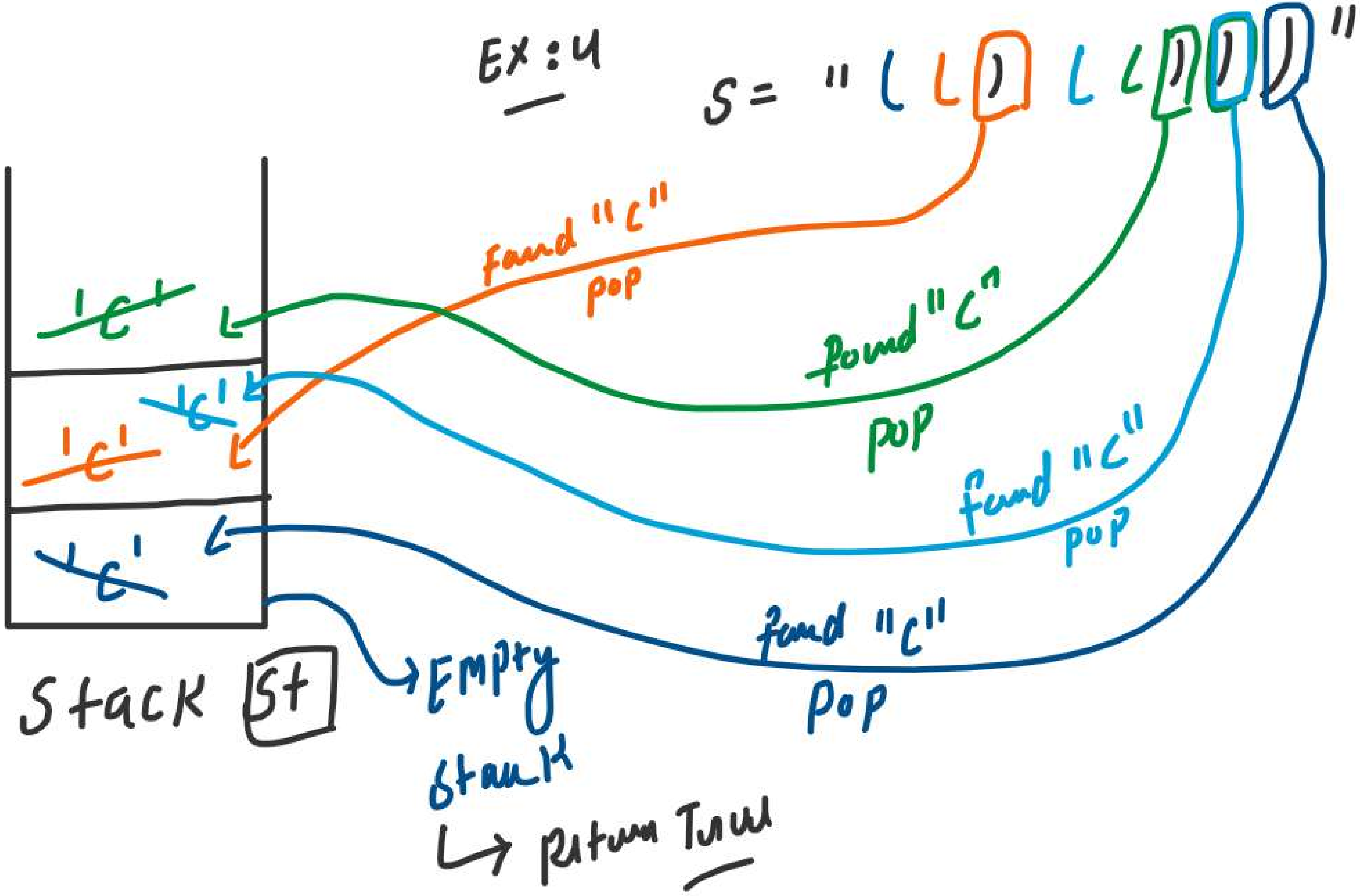


Ex : 3

S = "( ) { } [ ]"

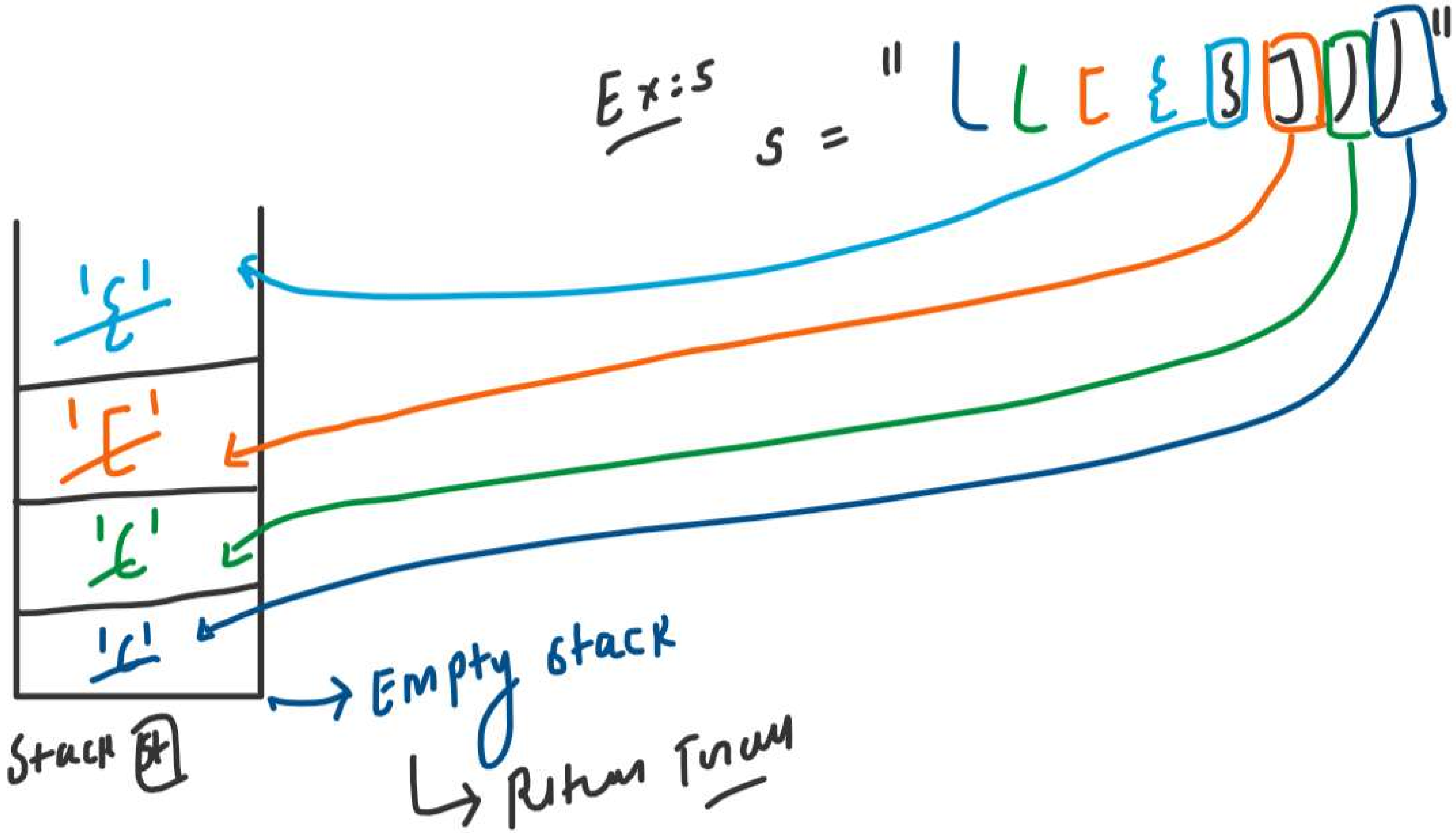

Found "{"
pop
Found "("
pop()
found "["
pop
~~'['~~
~~'{'~~
~~'('~~
Stack St
Empty stack
Return True

Ex : 4


S = "(()(()))"
~~'('~~
~~'('~~
~~'('~~
~~'('~~
found "("
pop
found "("
pop
found "("
pop
found "("
pop
Stack St
Empty stack
↳ Return True

Ex:5

s = "(([{}]))"



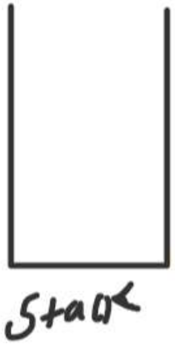
→ Empty stack

↳ Return True

***
HIDDEN
Testcase

Ex:6

S = "] "

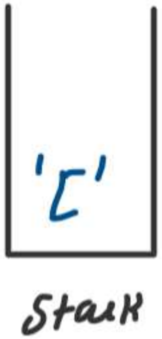


return False

Empty stack

Ex:7

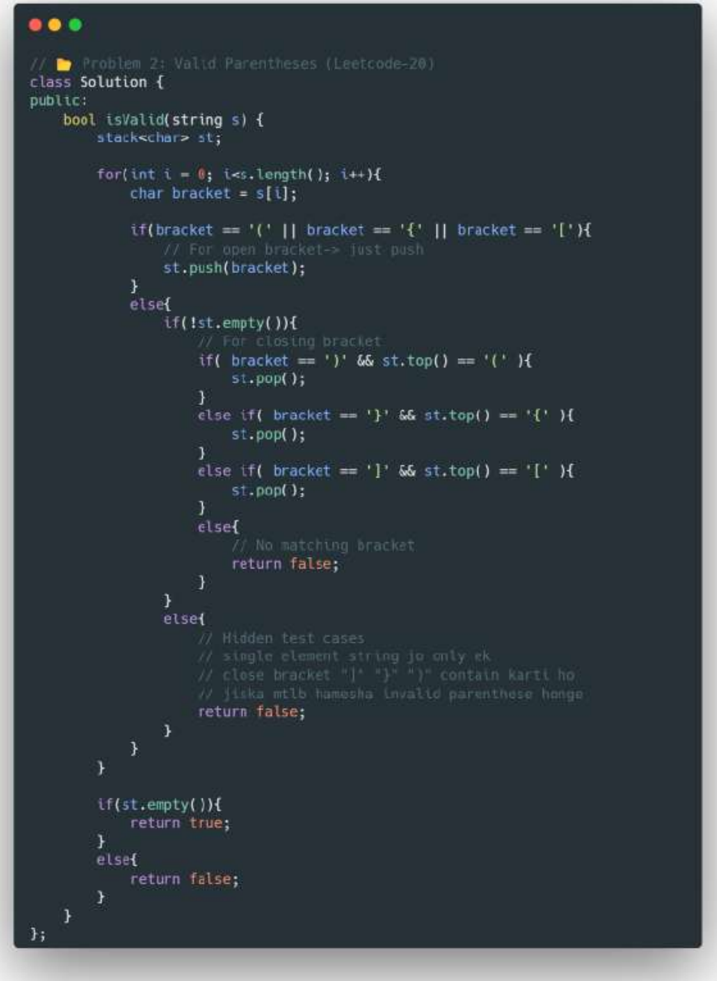
S = " ["



return False

Non-Empty stack

No matching Bracket



```cpp
// 📂 Problem 2: Valid Parentheses (Leetcode-20)
class Solution {
public:
    bool isValid(string s) {
        stack<char> st;

        for(int i = 0; i<s.length(); i++){
            char bracket = s[i];

            if(bracket == '(' || bracket == '{' || bracket == '['){
                // For open bracket-> just push
                st.push(bracket);
            }
            else{
                if(!st.empty()){
                    // For closing bracket
                    if( bracket == ')' && st.top() == '(' ){
                        st.pop();
                    }
                    else if( bracket == '}' && st.top() == '{' ){
                        st.pop();
                    }
                    else if( bracket == ']' && st.top() == '[' ){
                        st.pop();
                    }
                    else{
                        // No matching bracket
                        return false;
                    }
                }
                else{
                    // Hidden test cases
                    // single element string jo only ek
                    // close bracket "]" "}" ")" contain karti ho
                    // jiska mtlb hamesha invalid parenthese honge
                    return false;
                }
            }
        }

        if(st.empty()){
            return true;
        }
        else{
            return false;
        }
    }
};
```

T.C. O(N)

where N is size of string.

S.C. O(N)

where N is Number of oPen Brackets in stack.

3. Remove Redundant Brackets

Ex	output
1) (a+b)	False
2) (a+(b))	True
3) ((a+b))	True
4) ((a)+(b))	True
5) (a+b*c)	False
6) (a+(b*c))	False
7) ((a)+(b*c))	TRUE

True → REDUNDANT BRACKET PRESENT HAI

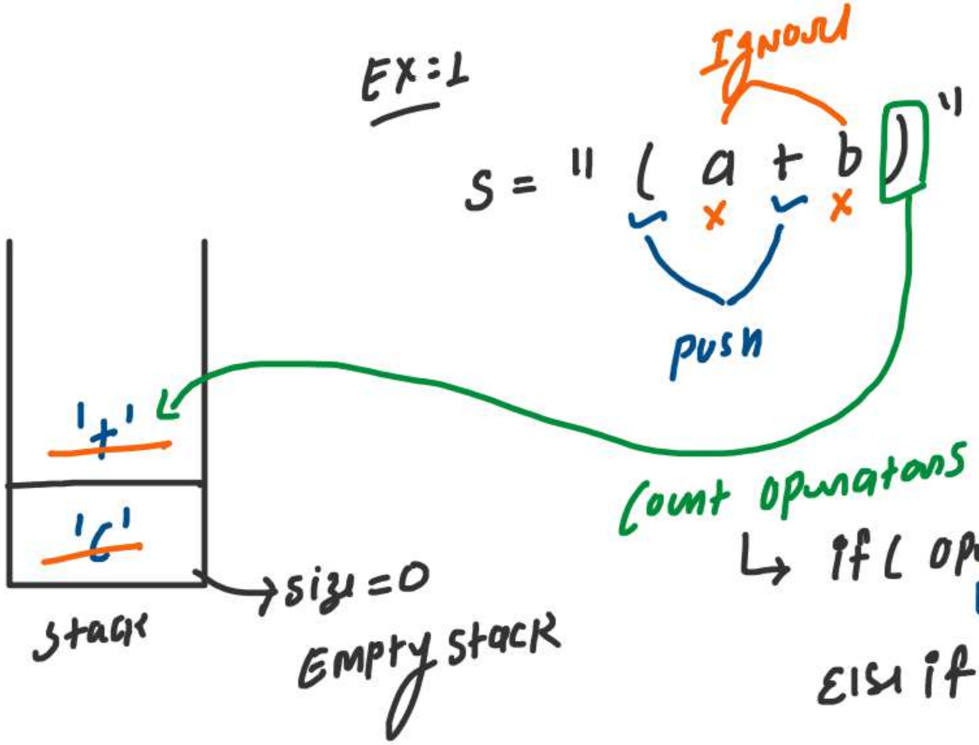
False → REDUNDANT BRACKET PRESENT NAHI HAI

(+ / * — %)

→ we will count operators only jab tak open Bracket nahi mil jata

DRY RUN

EX:1



Operator = 1

O/P ⇒ False

Count operators and pop

```
↳ if(operator==0)
    ↳ return True;
Else if(operator>0) ✓
    ↳ return False;
```

EX:2

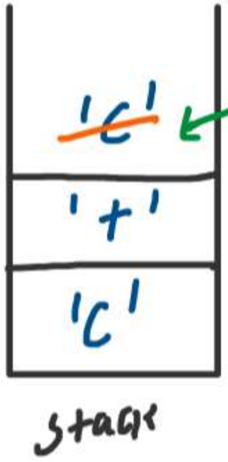
REDUNDAT BRACKET $\boxed{(b)}$

S = " (a + (b)) "

✓ ✗ ✓ ✓ ✗

operator = 0

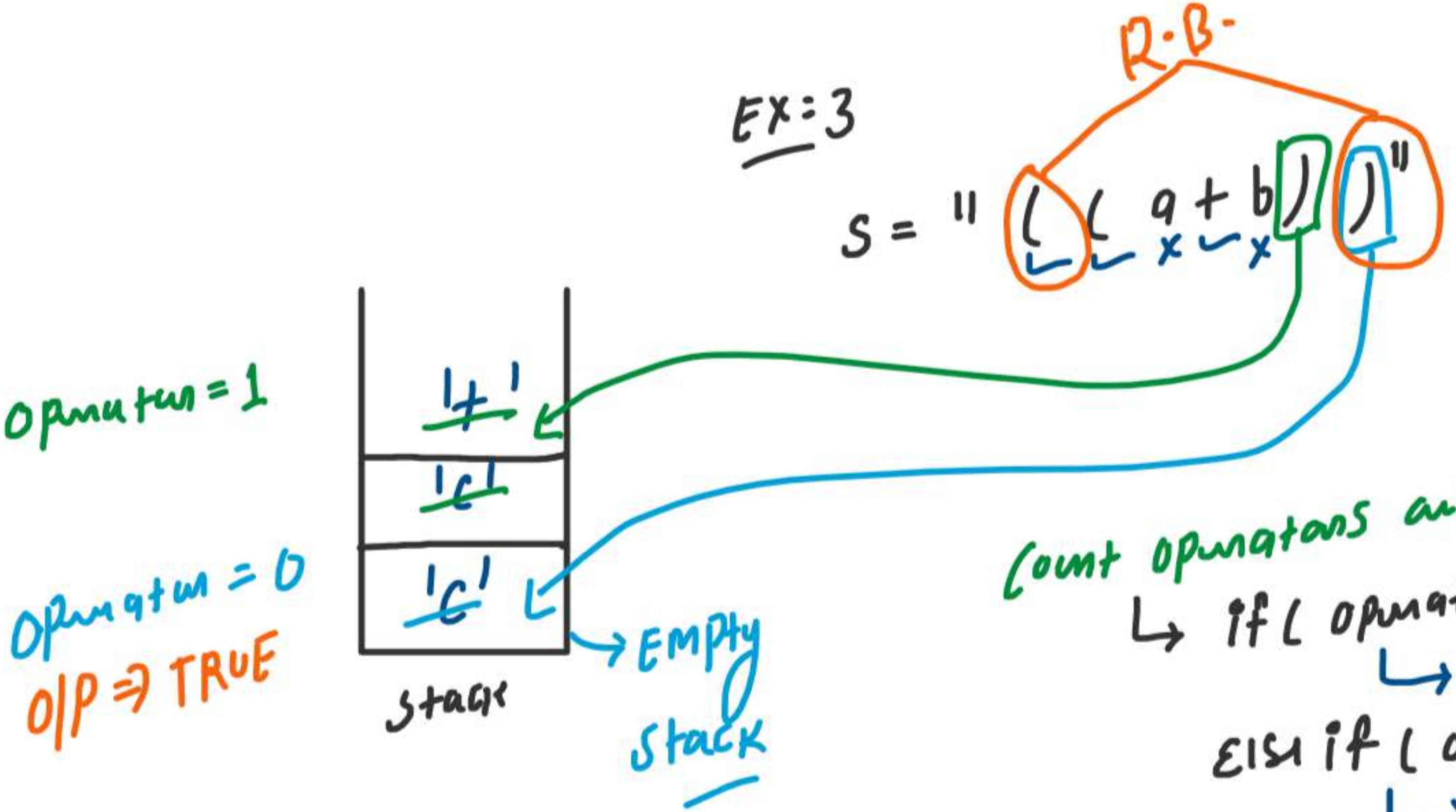
O/P ⇒ TRUE



Count operators and pop

```
↳ if ( operator == 0 ) ✓
      ↳ return True;
Else if ( operator > 0 )
      ↳ return False;
```

EX:3



```
Count operators and pop
↳ if ( operators == 0 )
      ↳ return True;
  Else if ( operators > 0 )
      ↳ return False;
```

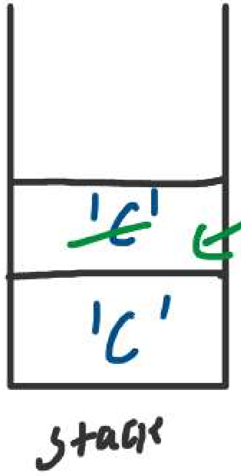
Ex:4

s = " ((a) + (b)) "

✓ ✓ ✗

operator = 0

O/P ⇒ TRUE



Non Empty stack

Count operators and pop

```
↳ if ( operator == 0 ) ✓
      ↳ return True;
  Else if ( operator > 0 )
      ↳ return False;
```

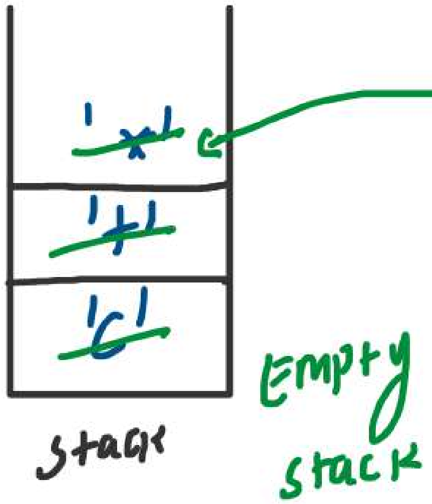
EX:5

S = " (a + b * c) "

✓ ✗ ✓ ✗ ✓ ✗

operator = 2

O/P => FASE



Count operators and pop

```
↳ if ( operator == 0 )
        ↳ return True;
  Else if ( operator > 0 ) ✓
        ↳ return False;
```

Ex: 6

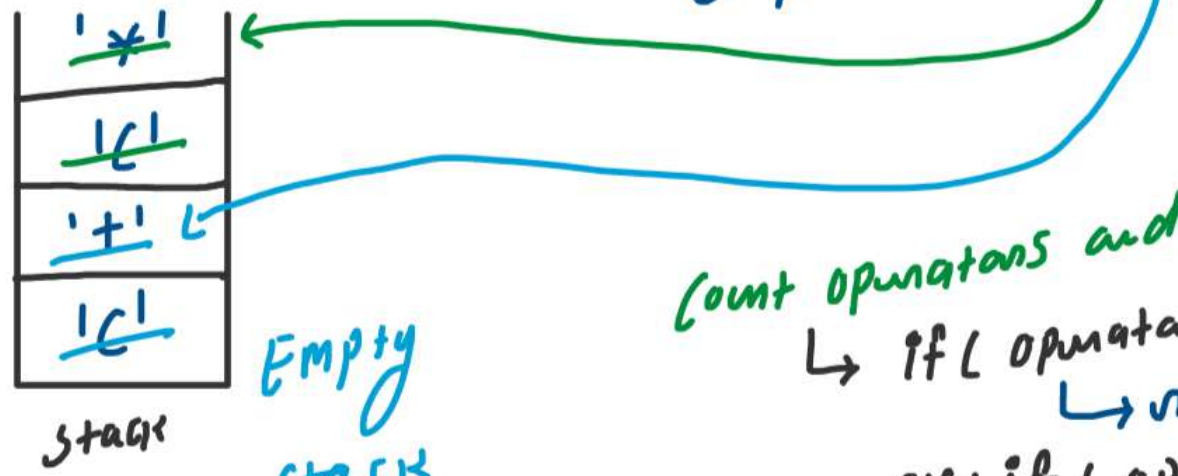
s = "(a+(b*c))"

✓ ✗ ✓ ✓ ✓ ✗ ✓ ✓ ✗

operator = 1

operator = 1

O/P → False



Empty stack

Count operators and pop

```
↳ if( operator == 0)
    ↳ return True;
  else if( operator > 0) ✓
    ↳ return False;
```

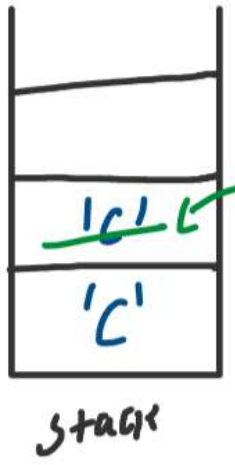
Ex:7

s = "((a) + (b * c))"

✓ ✓ ×

open = 0

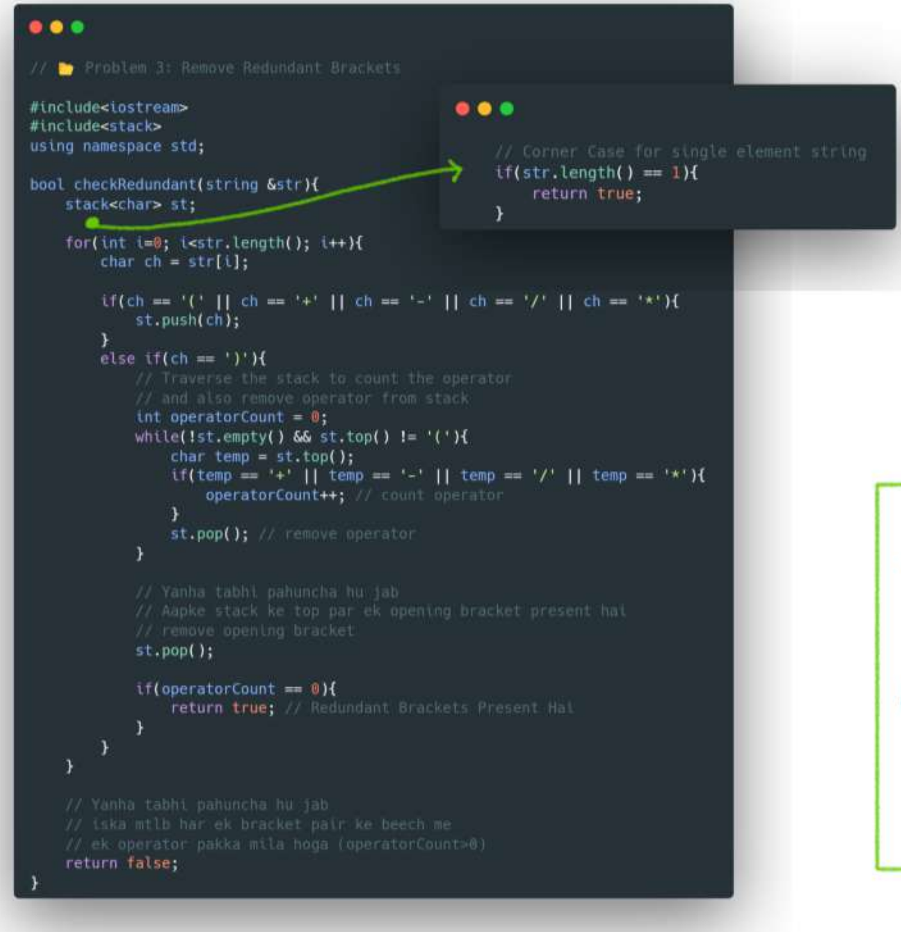
O/P => TRUE



Non Empty stack

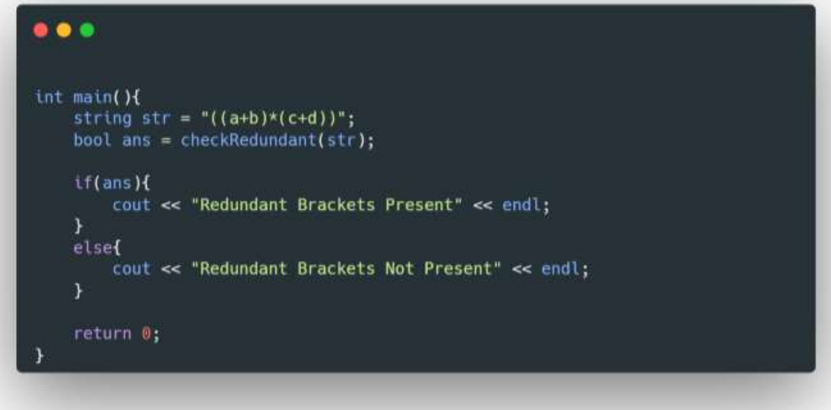
Count operators and pop

```
↳ if(operator == 0) ✓
    ↳ return True;
else if(operator > 0)
    ↳ return False;
```



```cpp
// 📁 Problem 3: Remove Redundant Brackets

#include<iostream>
#include<stack>
using namespace std;

bool checkRedundant(string &str){
    stack<char> st;

    for(int i=0; i<str.length(); i++){
        char ch = str[i];

        if(ch == '(' || ch == '+' || ch == '-' || ch == '/' || ch == '*'){
            st.push(ch);
        }
        else if(ch == ')'){
            // Traverse the stack to count the operator
            // and also remove operator from stack
            int operatorCount = 0;
            while(!st.empty() && st.top() != '('){
                char temp = st.top();
                if(temp == '+' || temp == '-' || temp == '/' || temp == '*'){
                    operatorCount++; // count operator
                }
                st.pop(); // remove operator
            }

            // Yanha tabhi pahuncha hu jab
            // Aapke stack ke top par ek opening bracket present hai
            // remove opening bracket
            st.pop();

            if(operatorCount == 0){
                return true; // Redundant Brackets Present Hai
            }
        }
    }

    // Yanha tabhi pahuncha hu jab
    // iska mtlb har ek bracket pair ke beech me
    // ek operator pakka mila hoga (operatorCount>0)
    return false;
}
```

```cpp
    // Corner Case for single element string
    if(str.length() == 1){
        return true;
    }
```



```cpp
int main(){
    string str = "((a+b)*(c+d))";
    bool ans = checkRedundant(str);

    if(ans){
        cout << "Redundant Brackets Present" << endl;
    }
    else{
        cout << "Redundant Brackets Not Present" << endl;
    }

    return 0;
}
```

T.C. ⇒ O(N), where N is length of string.

S.C. ⇒ O(N), where N is Number of opening Brackets and operators in stack.

Hidden Test Cases

```
Input: s = "("
Output: true

Input: s = ")"
Output: true

Input: s = "+"
Output: true
```

```
if ( string length == 1)
    return true;
```