Paranthesis matching problem:stack

Given a string containing parentheses — typically ( and ), or extended to include {}, [], etc. — determine whether the parentheses are **balanced** and **properly nested**.

**🔹 Example**

**Valid strings:**

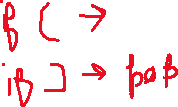
* ()
* (())
* (()())
* {[()()]}

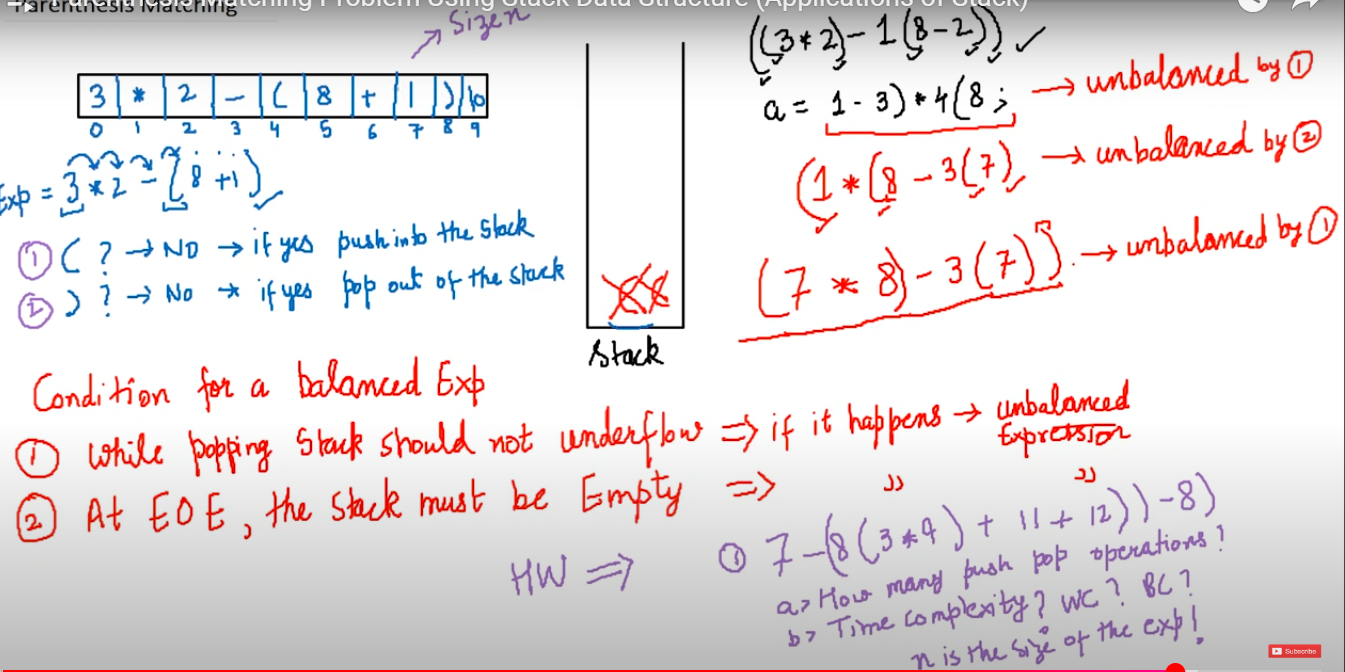
**Invalid strings:**

* (
* (()
* ())(
* {[()]}}

Let first we have an expression 3\*2+(5-2) now we have to see if all pranthesis are open and closed

sta





**🔹 What is "3\*2+(5-4)"?**

It’s a **string literal** in C, which is **stored in memory** like this:

sql

CopyEdit

+---+---+---+---+---+---+---+---+---+---+

| 3 | \* | 2 | + | ( | 5 | - | 4 | ) | \0|

+---+---+---+---+---+---+---+---+---+---+

↑

|

Address of the first character (pointer)

That string is **actually an array of characters** ending with a special null character \0.

**🔹 Why char \*exp?**

Because:

* A **string** in C is represented as a **pointer to its first character**.
* So when you write:

char \*exp = "3\*2+(5-4)";

You're saying:

"exp is a pointer to a character, and it's pointing to the first character ('3') of this string."

**🔹 What Happens Behind the Scenes?**

char \*exp = "3\*2+(5-4)";

* "3\*2+(5-4)" is stored in **read-only memory**.
* exp holds the **memory address** of the first character.

**🔹 Alternative (modifiable string):**

char exp[] = "3\*2+(5-4)";

Here:

* You're creating an **array** in local stack memory.
* You can **modify** the characters.

**🔸 Comparison Table:**

| **Declaration** | **Type** | **Can Modify?** | **Memory Location** |
| --- | --- | --- | --- |
| char \*exp = "abc"; | Pointer | ❌ No | Read-only |
| char exp[] = "abc"; | Array | ✅ Yes | Stack (local array) |

**✅ 1. In C, Arrays Are Always Passed as Pointers**

In C:

c

CopyEdit

void parenthesisMatch(char exp[]) {

// This is exactly the same as:

// void parenthesisMatch(char \*exp)

}

* ✅ char exp[] and char \*exp **mean the same thing** in function parameters.
* ❌ You can’t do just exp alone — C **needs the type** for every parameter.
* exp is a variable name, not a type.

**✅ 2. Why You Must Write char exp[] or char \*exp**

In C, function parameters **must** include the type:

c

CopyEdit

// ❌ Invalid in C:

void func(exp) { ... }

// ✅ Valid:

void func(char \*exp) { ... }

// or

void func(char exp[]) { ... }

So this:

c

CopyEdit

perenthesisMatch(exp);

passes the **address of the first character** (pointer) to the function, and inside the function:

c

CopyEdit

void perenthesisMatch(char exp[])

you receive that pointer.

**✅ 3. So Why Not Just Write exp?**

Because:

* exp is a variable, **not a type**.
* In C, you must declare the **type of every parameter** explicitly.

C++ lets you use **references** (int &x) which is syntactic sugar over pointers, but C doesn't support that — you have to use pointers directly (int \*x or char \*exp).

->why not all things inside fun in c you cannot do fun inside fun so you cannotvoid perenthesisMatch(char exp[])

{

    // Step 1: Define Stack structure inside function

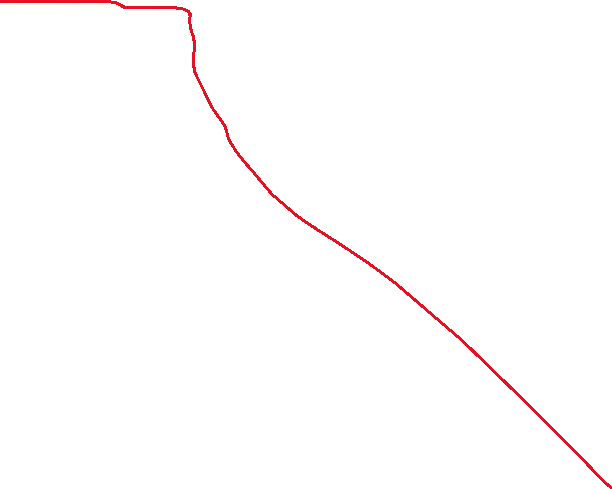
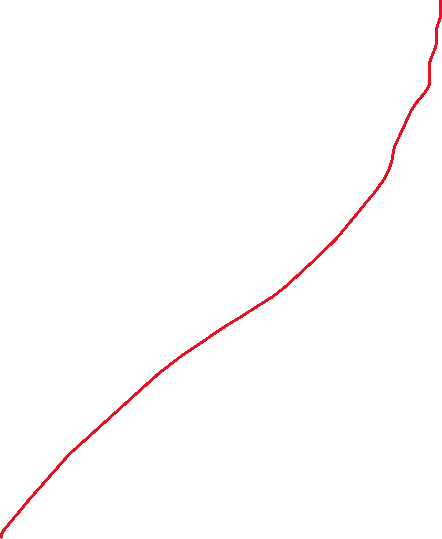
    struct Stack

    {

        int top;

        char items[100]; // You can change size

    };



    // Step 2: Declare and initialize stack

    struct Stack s;

    s.top = -1;

    void push(char c)

    {

        s.top++;

        s.items[s.top] = c;

    }

}