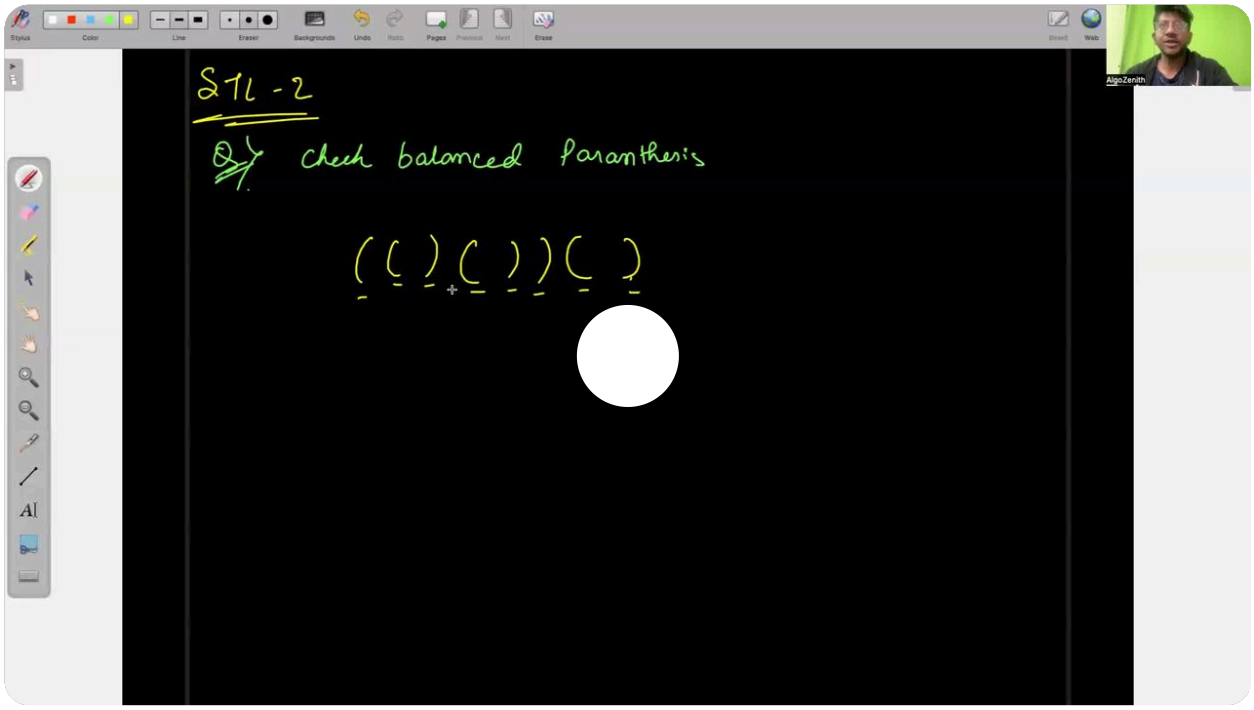
STL Application - 2



STL Applications and Problem Ideas-2

1. Problem: Determine if a given string of parentheses is valid.

Algorithm Explanation:

- The algorithm uses a simple approach to check for valid parentheses.
- It iterates through each character in the input string.
- If it encounters an opening parenthesis '(', it increments the depth counter.
- If it encounters a closing parenthesis ')', it checks if there is a matching opening parenthesis. If the depth is zero, indicating no opening parenthesis to match, it returns false.
- For every closing parenthesis, it decrements the depth counter.
- The final check ensures that all opening parentheses have a corresponding closing parenthesis, resulting in a depth of 0.

Code:

```
#include <iostream>
#include <stack>
```

```
bool isValidParenthesis(const std::string& s) {
    int depth = 0;


    for (char c : s) {
        if (c == '(') {
            depth++;
        } else if (c == ')') {
            if (depth <= 0) {
                return false;  // Unmatched closing parenthesis
            }
            depth--;
        }
    }


    return depth == 0;  // Check if all opening parentheses are matched
}


int main() {
    // Example usage:
    std::string testString = "((()))";

    if (isValidParenthesis(testString)) {
        std::cout << "The given string has valid parentheses." << std::endl;
    } else {
        std::cout << "The given string does not have valid parentheses." << std::endl;
    }


    return 0;
}
```

Time Complexity O(n) as the whole string is being traversed.

But, what would happen if we have several different types of parentheses? – then, it's more appropriate to use a stack-based approach. This allows for handling different types of opening and closing parentheses.

Stack Approach:

- The algorithm essentially uses a simple counter (depth) to keep track of the parentheses balance.
- Alternatively, a stack data structure can be employed for a more general approach.
- Push each opening parenthesis onto the stack and pop from the stack for every closing parenthesis.
- The stack approach allows handling more complex scenarios involving multiple types of parentheses.

Code:

Copy

```
#include <bits/stdc++.h>
using namespace std;
```

```
int main()
{
    map<char, int> mp;
    mp['('] = +1;
    mp['['] = +2;
    mp['{'] = +3;
    mp[')'] = -1;
    mp[']'] = -2;
    mp['}'] = -3;
    mp['<'] = +4;
    mp['>'] = -4;

    string s;
    cin >> s;
    stack<int> st;
    int is_balanced = 1;

    for (auto v : s)
    {
        int val = mp[v];
        if (val > 0)
        {
            // open bracket type
            st.push(val);
        }
        else
        {
            if (!st.empty() && st.top() + val == 0)
            {
                st.pop();
            }
            else
            {
                is_balanced = 0;
                break;
            }
        }
    }

    if (!st.empty())
        is_balanced = 0;

    if (is_balanced)
        cout << "Is Balanced\n";
    else
        cout << "Not balanced\n";
}
```

Time Complexity O(n) as the whole string is being traversed.

Mark as Read

Write a comment