# Demo ticket

## Session

**ID:** demoE37WKP-FWA
**Time limit:** 120 min.

## Status: closed

**Created on:** 2014-03-17 01:33 UTC
**Started on:** 2014-03-17 01:33 UTC
**Finished on:** 2014-03-17 02:59 UTC

## Tasks in test

## Task score

Test score

100%

100 out of 100 points

EASY

## 1. Brackets

Determine whether a given string of parentheses is properly nested.

**score: 100 of 100**

### Task description

A string S consisting of N characters is considered to be *properly nested* if any of the following conditions is true:

- S is empty;
- S has the form "(U)" or "[U]" or "{U}" where U is a properly nested string;
- S has the form "VW" where V and W are properly nested strings.

For example, the string "{[()()]}" is properly nested but "([)()]" is not.
Write a function:

```
int solution(const string &S);
```

that, given a string S consisting of N characters, returns 1 if S is properly nested and 0 otherwise.
For example, given S = "{[()()]}", the function should return 1 and given S = "([)()]", the function should return 0, as explained above.
Assume that:

- N is an integer within the range [0..200,000];
- string S consists only of the following characters: "(", "{", "[", "]", "}" and/or ")".

Complexity:

- expected worst-case time complexity is O(N);
- expected worst-case space complexity is O(N) (not counting the storage required for input arguments).



### Solution

**Programming language used:** C++

**Total time used: 87 minutes**

**Effective time used: 87 minutes**

**Notes:** correct functionality and scalability

**Task timeline**

01:33:20 02:59:25

Code: 02:59:25 UTC, cpp, final, score: **100.00**

```cpp
#include <map>
#include <stack>

int solution(const string &S) {
    map<char, char> matched;
    stack<char> pushed;

    matched[']'] = '[';
    matched[')'] = '(';
    matched['}'] = '{';

    for (int i = 0; i < (int)S.size(); i++) {
        // to avoid const char errors
        char ch = S[i];
        // as long as it's an opening bracket, keep pushing
        if (ch == '[' || ch == '(' || ch == '{') {
            pushed.push(ch);
        } else {
            // more chars in string but stack is empty
```

```
                (missing opening brackets)
20.             if (pushed.empty()) {
21.                 return 0;
22.             // there is a matching opening bracket on
                top of stack
23.             // pop it!
24.             } else if (matched[ch] == pushed.top()) {
25.                 pushed.pop();
26.             // there is not matching opening bracket on
                top of stack
27.             } else {
28.                 return 0;
29.             }
30.         }
31.     }
32.     if (pushed.empty())
33.         return 1;
34.     // there are left overs
35.     else {
36.         return 0;
37.     }
38. }
```

## Analysis

**Detected time complexity:**

# O(N)

| test | time | result |
|---|---|---|
| example1<br>example test 1 | 0.020 s. | **OK** |
| example2<br>example test 2 | 0.020 s. | **OK** |
| negative_match<br>invalid structures | 0.020 s. | **OK** |
| empty<br>empty string | 0.020 s. | **OK** |
| simple_grouped<br>simple grouped positive and negative test, length=22 | 0.020 s. | **OK** |
| large1<br>simple large positive test, 100K ('s followed by 100K )'s + )( | 0.020 s. | **OK** |
| large2<br>simple large negative test, 10K+1 ('s followed by 10K )'s + )( + () | 0.020 s. | **OK** |
| large_full_ternary_tree<br>tree of the form T=(TTT) and depth 11, length=177K+ | 0.020 s. | **OK** |
| multiple_full_binary_trees<br>sequence of full trees of the form T=(TT), depths [1..10..1], with/without some brackets at the end, length=49K+ | 0.020 s. | **OK** |
| broad_tree_with_deep_paths<br>string of the form [TTT...T] of 300 T's, each T being '{{{...}}}' nested 200-fold, length=120K+ | 0.020 s. | **OK** |

Training center