

Demo ticket

Session

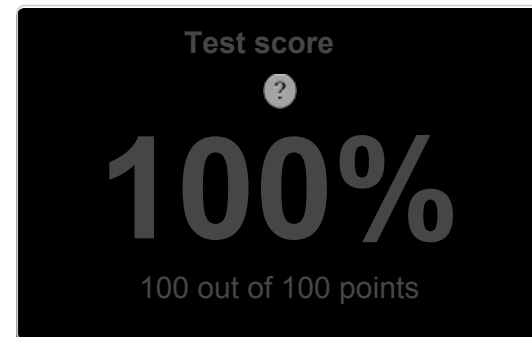
ID: demoE37WKP-FWA
Time limit: 120 min.

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



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).

Copyright 2009–2014 by Codility Limited. All Rights Reserved. Unauthorized copying, publication or disclosure prohibited.

Solution

Programming language used: C++

Total time used: 87 minutes

Effective time used: 87 minutes

Notes: correct functionality and scalability

Task timeline



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

```
01. #include <map>
02. #include <stack>
03.
04. int solution(const string &S) {
05.     map<char, char> matched;
06.     stack<char> pushed;
07.
08.     matched['['] = ']';
09.     matched['('] = ')';
10.     matched['{'] = '}';
11.
12.     for (int i = 0; i < (int)S.size(); i++) {
13.         // to avoid const char errors
14.         char ch = S[i];
15.         // as long as it's an opening bracket, keep
16.         // pushing
17.         if (ch == '[' || ch == '(' || ch == '{') {
18.             pushed.push(ch);
19.         } else {
20.             // more chars in string but stack is empty
```

```

20.         (missing opening brackets)
21.         if (pushed.empty()) {
22.             return 0;
23.             // there is a matching opening bracket on
24.             // top of stack
25.             // pop it!
26.         } else if (matched[ch] == pushed.top()) {
27.             pushed.pop();
28.             // there is not matching opening bracket on
29.             // top of stack
30.         } else {
31.             return 0;
32.         }
33.     }
34.     if (pushed.empty())
35.         return 1;
36.     // there are left overs
37.     else {
38.         return 0;
39.     }

```

Analysis

Detected time complexity:

$O(N)$

test	time	result
example1 example test 1	0.020 s.	OK
example2 example test 2	0.020 s.	OK
negative_match invalid structures	0.020 s.	OK
empty empty string	0.020 s.	OK
simple_grouped simple grouped positive and negative test, length=22	0.020 s.	OK
large1 simple large positive test, 100K ('s followed by 100K)'s +)(0.020 s.	OK
large2 simple large negative test, 10K+1 ('s followed by 10K)'s +)(+ (0.020 s.	OK
large_full_ternary_tree tree of the form T=(TTT) and depth 11, length=177K+	0.020 s.	OK
multiple_full_binary_trees 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 string of the form [TTT...T] of 300 T's, each T being '{{{...}}}' nested 200-fold, length=120K+	0.020 s.	OK

Training center