codility

Check out Codility training tasks

Candidate Report: Anonymous

Test Name:

Summary Timeline

Test Score

100 out of 100 points

100%

Tasks in Test

	Time Spent	Task Score
Brackets Submitted in: C++	2 min	100%

TASKS DETAILS

EASY

1. Brackets

Determine whether a given string of parentheses (multiple types) is properly nested.

Task Score	Correctness	Performance
100%	100%	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.

Solution

Programming language used:	C++
Total time used:	2 minutes
Effective time used:	2 minutes
Notes:	*not defined yet*

Write a function:

```
int solution(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.

Write an **efficient** algorithm for the following assumptions:

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

Task timeline

05:41:55 05:42:58

Code: 05:42:58 UTC, cpp, final, score: **100** show code in pop-up

```cpp
#include <string>
#include <stack>

int solution(std::string& S);

//S is empty;
//S has the form "(U)" or "[U]" or "{U}" where U
//S has the form "VW" where V and W are properly
//For example, the string "{[()()]}" is properly

int solution(std::string& S)
{
   int result = 1;
   std::stack<char> chars;
   if (!S.empty()) {
      for (auto c : S) {
         switch (c) {
         case '(':
         case '{':
         case '[':
            chars.push(c);
            break;
         case ')':
         {
            char startb = chars.top();
            chars.pop();
            if (startb != '(') {
               result = 0;
               break;
            }
         }
         break;
         case ']':
         {
            char startb = chars.top();
            chars.pop();
            if (startb != '[') {
               result = 0;
               break;
            }
         }
         break;

         case '}':
         {
            char startb = chars.top();
            chars.pop();
            if (startb != '{') {
               result = 0;
```

```
50                  break;
51              }
52          }
53          break;
54          }
55      }
56   }
57   if (!chars.empty()) {
58      result = 0;
59   }
60   return result;
61}
```

Analysis summary

The solution obtained perfect score.

Analysis

Detected time complexity: **O(N)**

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