Codility_

CodeCheck Report: training9YR9VJ-N4G

Test Name:

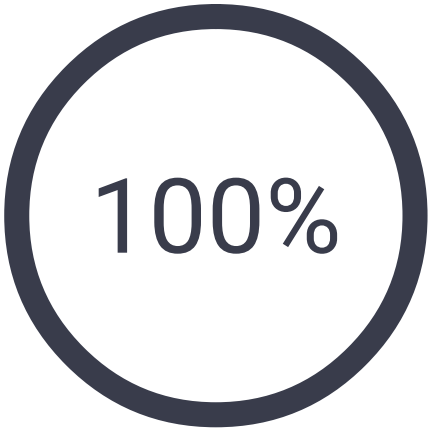
Check out Codility training tasks

Summary Timeline

Tasks summary

Task	Time spent	Score
Brackets C++ ⚠	2 min	100%

Total score



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

Solution

Programming language used: C++

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(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 ")".

Total time used:	2 minutes
Effective time used:	2 minutes
Notes:	*not defined yet*

Task timeline

Code: 07:19:02 UTC, cpp, final, score: **100**

show code in pop-up

```cpp
#include <bits/stdc++.h>
int solution(string &S){

        stack<char> pila;
        for(char n:S){
                if(n=='['|| n=='(' || n=='{')
                        pila.push(n);
                else if(!pila.empty()){
                        if((n==']' && pila.top() == '[') ||
                                (n=='}' && pila.top() == '{') ||
                                (n==')' && pila.top() == '(') )
                                        pila.pop();
                }
                else{
                        pila.push(n);
                }
        }

        if(pila.empty())
                return 1;
        else
                return 0;

}
```

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