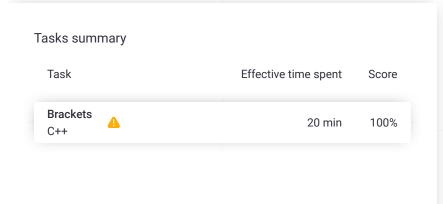
# Codility\_

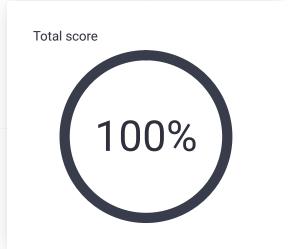
## Screen Report: Anonymous

Test Name:

Summary Timeline ★ Al Assistant Chat

Check out Codility training tasks





## Tasks Details

#### 1. Brackets

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

Task Score

Correctness

Solution

100%

Performance

100%

0

### 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(string &S);

that, given a string S consisting of N characters, returns 1 if S is properly nested and 0 otherwise.

Programming language used: C+

Time spent on task: 20 minutes

100%

Notes: not defined yet

Task timeline

11:00:10 11:19:12

Code: 11:19:12 UTC, cpp, show code in pop-up

1 of 3

final, score: 100

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 is made only of the following characters: '(', '{', '[', ']', '}' and/or ')'.

Copyright 2009–2024 by Codility Limited. All Rights Reserved. Unauthorized copying, publication or disclosure prohibited.

```
1
     #include <stack>
2
     #include <string>
3
4
     using namespace std;
 5
 6
     int solution(string &S) {
7
 8
         if(S.empty())
9
10
              return 1;
11
         }
12
         stack<char> nestingTracker;
13
14
15
         for(unsigned int i = 0; i < S.length(); i++)</pre>
16
17
              char letter{S[i]};
18
              if(letter == '{' || letter == '(' || lette
19
20
21
                  nestingTracker.push(letter);
22
             }
             else if(nestingTracker.empty())
23
24
             {
25
                  return 0;
26
             }
             else if(letter == '}')
27
28
29
                  if(nestingTracker.top() == '{')
30
                  {
31
                      nestingTracker.pop();
32
                  }
33
                  else
34
                  {
35
                      return 0;
36
37
38
              else if(letter == ')')
39
40
                  if(nestingTracker.top() == '(')
41
                  {
                      nestingTracker.pop();
42
43
                  }
44
                  else
45
                  {
46
                      return 0;
47
                  }
48
              else if(letter == ']')
49
50
                  if(nestingTracker.top() == '[')
51
52
                  {
53
                      nestingTracker.pop();
54
                  }
55
                  else
56
                  {
57
                      return 0;
58
                  }
59
60
         }
61
         if(nestingTracker.empty())
62
63
         {
64
              return 1;
65
         }
66
67
         return 0;
68
     }
```

2 of 3 10/8/2024, 1:20 PM

## Analysis summary

The solution obtained perfect score.

# Analysis

Detected time complexity: O(N)

expa	nd all Example test	ts
<b>&gt;</b>	example1 example test 1	√ OK
<b>&gt;</b>	example2 example test 2	√ OK
ехра	nd all Correctness te	ests
•	negative_match invalid structures	√ OK
•	empty empty string	√ OK
•	simple_grouped simple grouped positive and negative test, length=22	√ OK
ехра	nd all Performance to	ests
<b>&gt;</b>	large1 simple large positive test, 100K ('s followed by 100K)'s + )(	√ OK
<b>&gt;</b>	large2 simple large negative test, 10K+1 ('s followed by 10K)'s + )( + ()	√ OK
<b>&gt;</b>	large_full_ternary_tree tree of the form T=(TTT) and depth 11, length=177K+	√ OK
<b>&gt;</b>	multiple_full_binary_trees sequence of full trees of the form T=(TT), depths [1101], with/without some brackets at the end, length=49K+	√ OK
<b>&gt;</b>	broad_tree_with_deep_paths string of the form [TTTT] of 300 T's, each T being '{{{}}}' nested 200-fold, length=120K+	√ OK

3 of 3