Codility_

CodeCheck Report: trainingJQVCEJ-7GR

Test Name:

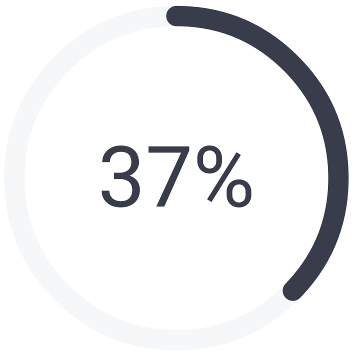
Check out Codility training tasks

Summary Timeline AI Assistant Transcript

Tasks summary

Task	Time spent	Score
Brackets C++ ⚠	9 min	37%

Total score



Tasks Details

Easy

1. Brackets

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

Task Score	Correctness	Performance
37%	33%	40%

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.

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:

Solution

Programming language used:	C++
Total time used:	9 minutes
Effective time used:	9 minutes
Notes:	*not defined yet*

Task timeline

05:04:10 05:12:51

Code: 05:12:51 UTC, cpp, final, score: **37** show code in pop-up

```
// you can use includes, for example:
// #include <algorithm>
```

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

```
#include <vector>

// you can write to stdout for debugging purpo
// cout << "this is a debug message" << endl;

int solution(string &S) {
    //stack
    vector<char> stack;
    int flag = 0;

    for (const char& e : S) {
        if (e == '(' || e == '{' || e == '[')
            stack.push_back(e);
        if (e == ')' || e == '}' || e == ']')
            if (stack.back() == '(' && e == ')
                flag = 1;
            if (stack.back() == '{' && e == ']
                flag = 1;
            if (stack.back() == '[' && e == ']
                flag = 1;

            if (flag == 1) {
                stack.pop_back();
                flag = 0;
            }
        }
    }

    if (stack.size() == 0)
        return 1;
    return 0;
}
```

Analysis summary

The following issues have been detected: wrong answers, runtime errors.

For example, for the input ')(' the solution terminated unexpectedly.

Analysis

expand all Example tests

▶ example1 ✔ OK
example test 1

▶ example2 ✔ OK
example test 2

expand all Correctness tests

▼ negative_match ✘ RUNTIME ERROR
invalid structures tested program terminated with exit code 1

1. 0.001 s **RUNTIME ERROR**, tested program terminated with exit code 1

stderr:
```
Segmentation Fault
```

2. 0.001 s **OK**
3. 0.001 s **OK**
4. 0.001 s **OK**
5. 0.001 s **OK**

▶

empty ✔ OK
empty string

▼ simple_grouped ✘ RUNTIME ERROR
simple grouped positive and negative test, length=22
tested program terminated with exit code 1

1. 0.001 s **OK**
2. 0.001 s **OK**
3. 0.001 s **RUNTIME ERROR**, tested program terminated with exit code 1

   stderr:
   ```
   Segmentation Fault
   ```
4. 0.001 s **RUNTIME ERROR**, tested program terminated with exit code 1

   stderr:
   ```
   Segmentation Fault
   ```
5. 0.001 s **OK**

expand all Performance tests

▼ large1 ✘ RUNTIME ERROR
simple large positive test, 100K ('s followed by 100K)'s +)(
tested program terminated with exit code 1

1. 0.004 s **OK**
2. 0.001 s **RUNTIME ERROR**, tested program terminated with exit code 1

   stderr:
   ```
   Segmentation Fault
   ```
3. 0.001 s **OK**

▼ large2 ✘ RUNTIME ERROR
simple large negative test, 10K+1 ('s followed by 10K)'s +)(+ ()
tested program terminated with exit code 1

1. 0.001 s **OK**
2. 0.001 s **RUNTIME ERROR**, tested program terminated with exit code 1

   stderr:
   ```
   Segmentation Fault
   ```
3. 0.001 s **OK**

▶ large_full_ternary_tree ✔ OK
tree of the form T=(TTT) and depth 11, length=177K+

▼ multiple_full_binary_trees ✘ WRONG ANSWER
sequence of full trees of the form T=(TT), depths [1..10..1], with/without some brackets at the end, length=49K+
got 1 expected 0

1. 0.001 s **OK**
2. 0.001 s **WRONG ANSWER**, got 1 expected 0
3. 0.001 s **OK**
4. 0.001 s **OK**

5. 0.001 s **OK**

▶ broad_tree_with_deep_paths ✔ **OK**
string of the form [TTT...T] of 300 T's, each T being '{{{...}}}' nested 200-fold, length=120K+