# 1 Problem: Check brackets in the code

## Problem Introduction

In this problem you will implement a feature for a text editor to find errors in the usage of brackets in the code.

## Problem Description

**Task.** Your friend is making a text editor for programmers. He is currently working on a feature that will find errors in the usage of different types of brackets. Code can contain any brackets from the set `[]{}()`, where the opening brackets are `[`,`{`, and `(` and the closing brackets corresponding to them are `]`,`}`, and `)`.

For convenience, the text editor should not only inform the user that there is an error in the usage of brackets, but also point to the exact place in the code with the problematic bracket. First priority is to find the first unmatched closing bracket which either doesn't have an opening bracket before it, like `]` in `](`), or closes the wrong opening bracket, like `}` in `()[}`. If there are no such mistakes, then it should find the first unmatched opening bracket without the corresponding closing bracket after it, like `(` in `{}([]`. If there are no mistakes, text editor should inform the user that the usage of brackets is correct.

Apart from the brackets, code can contain big and small latin letters, digits and punctuation marks.

More formally, all brackets in the code should be divided into pairs of matching brackets, such that in each pair the opening bracket goes before the closing bracket, and for any two pairs of brackets either one of them is nested inside another one as in `(foo[bar])` or they are separate as in `f(a,b)-g[c]`. The bracket `[` corresponds to the bracket `]`, `{` corresponds to `}`, and `(` corresponds to `)`.

**Input Format.** Input contains one string $S$ which consists of big and small latin letters, digits, punctuation marks and brackets from the set `[]{}()`.

**Constraints.** The length of $S$ is at least 1 and at most $10^5$.

**Output Format.** If the code in $S$ uses brackets correctly, output "Success" (without the quotes). Otherwise, output the 1-based index of the first unmatched closing bracket, and if there are no unmatched closing brackets, output the 1-based index of the first unmatched opening bracket.

**Time Limits.** `C`: 1 sec, `C++`: 1 sec, `Java`: 1 sec, `Python`: 1 sec. `C#`: 1.5 sec, `Haskell`: 2 sec, `JavaScript`: 3 sec, `Ruby`: 3 sec, `Scala`: 3 sec.

**Memory Limit.** 512MB.

**Sample 1.**

Input:

```
[]
```

Output:

```
Success
```

Explanation:

The brackets are used correctly: there is just one pair of brackets `[` and `]`, they correspond to each other, the left bracket `[` goes before the right bracket `]`, and no two pairs of brackets intersect, because there is just one pair of brackets.

**Sample 2.**

Input:

```
{}[]
```

Output:

```
Success
```

Explanation:
The brackets are used correctly: there are two pairs of brackets — first pair of `{` and `}`, and second pair of `[` and `]` — and these pairs do not intersect.

**Sample 3.**

Input:

```
[()]
```

Output:

```
Success
```

Explanation:
The brackets are used correctly: there are two pairs of brackets — first pair of `[` and `]`, and second pair of `(` and `)` — and the second pair is nested inside the first pair.

**Sample 4.**

Input:

```
(())
```

Output:

```
Success
```

Explanation:
Pairs with the same types of brackets can also be nested.

**Sample 5.**

Input:

```
{[]}()
```

Output:

```
Success
```

Explanation:
Here there are 3 pairs of brackets, one of them is nested into another one, and the third one is separate from the first two.

**Sample 6.**

Input:

```
{
```

Output:

```
1
```

Explanation:
The code `{` doesn't use brackets correctly, because brackets cannot be divided into pairs (there is just one bracket). There are no closing brackets, and the first unmatched opening bracket is `{`, and its position is 1, so we output 1.

**Sample 7.**

Input:

```
{[}
```

Output:

```
3
```

Explanation:
The bracket `}` is unmatched, because the last unmatched opening bracket before it is `[` and not `{`. It is the first unmatched closing bracket, and our first priority is to output the first unmatched closing bracket, and its position is 3, so we output 3.

**Sample 8.**

Input:

```
foo(bar);
```

Output:

```
Success
```

Explanation:
All the brackets are matching, and all the other symbols can be ignored.

**Sample 9.**

Input:

```
foo(bar[i);
```

Output:

```
10
```

Explanation:
`)` doesn't match `[`, so `)` is the first unmatched closing bracket, so we output its position, which is 10.

## Starter Files

There are starter solutions only for C++, Java and Python3, and if you use other languages, you need to implement solution from scratch. Starter solutions read the code from the input and go through the code character-by-character and provide convenience methods. You need to implement the processing of the brackets to find the answer to the problem and to output the answer.

## What to Do

To solve this problem, you can slightly modify the `IsBalanced` algorithm from the lectures to account not only for the brackets, but also for other characters in the code, and return not just whether the code uses brackets correctly, but also what is the first position where the code becomes broken.

## Need Help?

Ask a question or see the questions asked by other learners at this forum thread.