

# Problem 1544: Make The String Great

## Problem Information

Difficulty: [Easy](#)

Acceptance Rate: 0.00%

Paid Only: No

## Problem Description

Given a string

$s$

of lower and upper case English letters.

A good string is a string which doesn't have

two adjacent characters

$s[i]$

and

$s[i + 1]$

where:

$0 \leq i \leq s.length - 2$

$s[i]$

is a lower-case letter and

$s[i + 1]$

is the same letter but in upper-case or

vice-versa

.

To make the string good, you can choose

two adjacent

characters that make the string bad and remove them. You can keep doing this until the string becomes good.

Return

the string

after making it good. The answer is guaranteed to be unique under the given constraints.

Notice

that an empty string is also good.

Example 1:

Input:

`s = "leEetcode"`

Output:

`"leetcode"`

Explanation:

In the first step, either you choose  $i = 1$  or  $i = 2$ , both will result `"leEetcode"` to be reduced to `"leetcode"`.

Example 2:

Input:

s = "abBAcC"

Output:

""

Explanation:

We have many possible scenarios, and all lead to the same answer. For example: "abBAcC" --> "aAcC" --> "cC" --> "" "abBAcC" --> "abBA" --> "aA" --> ""

Example 3:

Input:

s = "s"

Output:

"s"

Constraints:

1 <= s.length <= 100

s

contains only lower and upper case English letters.

## Code Snippets

**C++:**

```
class Solution {  
public:  
    string makeGood(string s) {
```

```
}  
};
```

### Java:

```
class Solution {  
    public String makeGood(String s) {  
  
    }  
}
```

### Python3:

```
class Solution:  
    def makeGood(self, s: str) -> str:
```

### Python:

```
class Solution(object):  
    def makeGood(self, s):  
        """  
        :type s: str  
        :rtype: str  
        """
```

### JavaScript:

```
/**  
 * @param {string} s  
 * @return {string}  
 */  
var makeGood = function(s) {  
  
};
```

### TypeScript:

```
function makeGood(s: string): string {  
  
};
```

**C#:**

```
public class Solution {  
    public string MakeGood(string s) {  
  
    }  
}
```

**C:**

```
char* makeGood(char* s) {  
  
}
```

**Go:**

```
func makeGood(s string) string {  
  
}
```

**Kotlin:**

```
class Solution {  
    fun makeGood(s: String): String {  
  
    }  
}
```

**Swift:**

```
class Solution {  
    func makeGood(_ s: String) -> String {  
  
    }  
}
```

**Rust:**

```
impl Solution {  
    pub fn make_good(s: String) -> String {  
  
    }  
}
```

## Ruby:

```
# @param {String} s
# @return {String}
def make_good(s)

end
```

## PHP:

```
class Solution {

    /**
     * @param String $s
     * @return String
     */
    function makeGood($s) {

    }

}
```

## Dart:

```
class Solution {
  String makeGood(String s) {

  }
}
```

## Scala:

```
object Solution {
  def makeGood(s: String): String = {

  }
}
```

## Elixir:

```
defmodule Solution do
  @spec make_good(s :: String.t) :: String.t
  def make_good(s) do
```

```
end
end
```

### Erlang:

```
-spec make_good(S :: unicode:unicode_binary()) -> unicode:unicode_binary().
make_good(S) ->
.
```

### Racket:

```
(define/contract (make-good s)
  (-> string? string?)
)
```

## Solutions

### C++ Solution:

```
/*
 * Problem: Make The String Great
 * Difficulty: Easy
 * Tags: string, stack
 *
 * Approach: String manipulation with hash map or two pointers
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(1) to O(n) depending on approach
 */

class Solution {
public:
    string makeGood(string s) {

    }
};
```

### Java Solution:

```
/**
 * Problem: Make The String Great
```

```

* Difficulty: Easy
* Tags: string, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public String makeGood(String s) {

}

}

```

### Python3 Solution:

```

"""
Problem: Make The String Great
Difficulty: Easy
Tags: string, stack

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def makeGood(self, s: str) -> str:
# TODO: Implement optimized solution
pass

```

### Python Solution:

```

class Solution(object):
def makeGood(self, s):
"""
:type s: str
:rtype: str
"""

```

### JavaScript Solution:

```

/**
 * Problem: Make The String Great
 * Difficulty: Easy
 * Tags: string, stack
 *
 * Approach: String manipulation with hash map or two pointers
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(1) to O(n) depending on approach
 */

/**
 * @param {string} s
 * @return {string}
 */
var makeGood = function(s) {

};

```

### TypeScript Solution:

```

/**
 * Problem: Make The String Great
 * Difficulty: Easy
 * Tags: string, stack
 *
 * Approach: String manipulation with hash map or two pointers
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(1) to O(n) depending on approach
 */

function makeGood(s: string): string {

};

```

### C# Solution:

```

/*
 * Problem: Make The String Great
 * Difficulty: Easy
 * Tags: string, stack
 *
 * Approach: String manipulation with hash map or two pointers

```

```

* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public string MakeGood(string s) {

}

}

```

### C Solution:

```

/*
* Problem: Make The String Great
* Difficulty: Easy
* Tags: string, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

char* makeGood(char* s) {

}

```

### Go Solution:

```

// Problem: Make The String Great
// Difficulty: Easy
// Tags: string, stack
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func makeGood(s string) string {

}

```

### Kotlin Solution:

```

class Solution {
    fun makeGood(s: String): String {

    }
}

```

### Swift Solution:

```

class Solution {
    func makeGood(_ s: String) -> String {

    }
}

```

### Rust Solution:

```

// Problem: Make The String Great
// Difficulty: Easy
// Tags: string, stack
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
    pub fn make_good(s: String) -> String {

    }
}

```

### Ruby Solution:

```

# @param {String} s
# @return {String}
def make_good(s)

end

```

### PHP Solution:

```

class Solution {

```

```

/**
 * @param String $s
 * @return String
 */
function makeGood($s) {

}
}

```

### Dart Solution:

```

class Solution {
  String makeGood(String s) {

  }
}

```

### Scala Solution:

```

object Solution {
  def makeGood(s: String): String = {

  }
}

```

### Elixir Solution:

```

defmodule Solution do
  @spec make_good(s :: String.t) :: String.t
  def make_good(s) do

  end
end

```

### Erlang Solution:

```

-spec make_good(S :: unicode:unicode_binary()) -> unicode:unicode_binary().
make_good(S) ->

.

```

### Racket Solution:

```
(define/contract (make-good s)
  (-> string? string?)
)
```