Most efficient way to remove special characters from string

Asked 15 years, 5 months ago Modified 2 days ago Viewed 617k times

320

I want to remove all special characters from a string. Allowed characters are A-Z (uppercase or lowercase), numbers (0-9), underscore (_), or the dot sign (.).

I have the following, it works but I suspect (I know!) it's not very efficient:

```
public static string RemoveSpecialCharacters(string str)
{
    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < str.Length; i++)
    {
        if ((str[i] >= '0' && str[i] <= '9')
            || (str[i] >= 'A' && str[i] <= 'z'
                || (str[i] == '.' || str[i] == '_')))
        {
            sb.Append(str[i]);
        }
    }

    return sb.ToString();
}
```

What is the most efficient way to do this? What would a regular expression look like, and how does it compare with normal string manipulation?

The strings that will be cleaned will be rather short, usually between 10 and 30 characters in length.

c# string special-characters

Share

Improve this question

Follow

edited Dec 2, 2022 at 12:33

Dariusz Woźniak

10.3k ● 7 ● 63 ● 80

asked Jul 13, 2009 at 15:33

ObiWanKenobi

14.9k ● 11 ● 49 ● 51

5 I won't put this in an answer since it won't be any more efficient, but there are a number of static char methods like char.IsLetterOrDigit() that you could use in your if statement to make it more legible at least. – Martin Harris Jul 13, 2009 at 15:40

5 I'm not sure that checking for A to z is safe, in that it brings in 6 characters that aren't alphabetical, only one of which is desired (underbar). – Steven Sudit Jul 13, 2009 at 15:41

4 Focus on making your code more readable. unless you are doing this in a loop like 500 times a second, the efficiency isn't a big deal. Use a regexp and it will be much easier to read.l – Byron Whitlock Jul 13, 2009 at 15:42

5 Byron, you're probably right about needing to emphasize readability. However, I'm skeptical about regexp being readable. :-) – Steven Sudit Jul 13, 2009 at 15:45

2 Regular expressions being readable or not is kind of like German being readable or not; it depends on if you know it or not (although in both cases you will every now and then come across grammatical rules that make no sense ;) – Blixt Jul 13, 2009 at 15:50

31 Answers

Sorted by: Highest score (default)

398

Why do you think that your method is not efficient? It's actually one of the most efficient ways that you can do it.

You should of course read the character into a local variable or use an enumerator to reduce the number of array accesses:

```
public static string RemoveSpecialCharacters(this string str) {
   StringBuilder sb = new StringBuilder();
   foreach (char c in str) {
      if ((c >= '0' && c <= '9') || (c >= 'A' && c <= 'Z') || (c >= 'a' && c <=
'z') || c == '.' || c == '_') {
         sb.Append(c);
      }
   }
   return sb.ToString();
}
```

One thing that makes a method like this efficient is that it scales well. The execution time will be relative to the length of the string. There is no nasty surprises if you would use it on a large string.

Edit:
I made a quick performance test, running each function a million times with a 24 character string. These are the results:

Original function: 54.5 ms.
My suggested change: 47.1 ms.
Mine with setting StringBuilder capacity: 43.3 ms.
Regular expression: 294.4 ms.

Edit 2: I added the distinction between A-Z and a-z in the code above. (I reran the performance test, and there is no noticable difference.)

Edit 3:
I tested the lookup+char[] solution, and it runs in about 13 ms.

The price to pay is, of course, the initialization of the huge lookup table and keeping it in memory. Well, it's not that much data, but it's much for such a trivial function...

```
private static bool[] _lookup;

static Program() {
   _lookup = new bool[65536];
   for (char c = '0'; c <= '9'; c++) _lookup[c] = true;
   for (char c = 'A'; c <= 'Z'; c++) _lookup[c] = true;
   for (char c = 'a'; c <= 'z'; c++) _lookup[c] = true;
   _lookup['.'] = true;
   _lookup['_'] = true;
}

public static string RemoveSpecialCharacters(string str) {
   char[] buffer = new char[str.Length];
   int index = 0;
   foreach (char c in str) {
      if (_lookup[c]) {
         buffer[index] = c;
         index++;
      }
   }
   return new string(buffer, 0, index);
}
```

Share

Improve this answer

Follow

edited Jun 25, 2017 at 8:46

nologo

6,248 ● 4 ● 38 ● 50

answered Jul 13, 2009 at 15:45

Guffa

700k ● 110 ● 750 ● 1k

6 I agree. The only other change I would make is to add the initial capacity argument to the StringBuilder constructor, "= new StringBuilder(str.Length)". – David Jul 13, 2009 at 16:05

2 My answer, using a `char[]` buffer rather than `StringBuilder`, has a slight edge on this one according to my testing. (Mine's less readable though, so the small performance benefit probably isn't worth it.) – LukeH Jul 13, 2009 at 16:46

1 @Steven: That may well be the case, but the benchmarks speak for themselves! In my tests, using a `char[]` buffer performs (slightly) better than `StringBuilder`, even when scaling up to strings that are tens of thousands of characters in length. – LukeH Jul 13, 2009 at 17:02

12 @downvoter: Why the downvote? If you don't explain what you think is wrong, it can't improve the answer. – Guffa Aug 6, 2011 at 18:56

2 @SILENT: No, it doesn't, but you should only do that once. If you allocate an array that large each time you call the method (and if you call the method frequently) then the method becomes the slowest by far, and causes a lot of work for the garbage collector. – Guffa May 5, 2015 at 7:36

241

Well, unless you really need to squeeze the performance out of your function, just go with what is easiest to maintain and understand. A regular expression would look like this:

For additional performance, you can either pre-compile it or just tell it to compile on first call (subsequent calls will be faster.)

```
public static string RemoveSpecialCharacters(string str)
{
    return Regex.Replace(str, "[^a-zA-Z0-9_.]+", "", RegexOptions.Compiled);
}
```

Share

Improve this answer

Follow

edited Jul 13, 2009 at 16:37

answered Jul 13, 2009 at 15:40

Blixt

50.1k ● 13 ● 124 ● 154

1 I'd guess that this is probably a complex enough query that it would be faster than the OP's approach, especially if pre-compiled. I have no evidence to back that up, however. It should be tested. Unless it's drastically slower, I'd choose this approach regardless, since it's way easier to read and maintain. +1 – rmeador Jul 13, 2009 at 15:48

6 Its a very simple regex (no backtracking or any complex stuff in there) so it should be pretty damn fast. – user1228 Jul 13, 2009 at 16:00

10 @rmeador: without it being compiled it is about 5x slower, compiled it is 3x slower than his method. Still 10x simpler though :-D – user7116 Jul 13, 2009 at 16:15

7 Regular expressions are no magical hammers and never faster than hand optimized code. – Christian Klauser Jul 13, 2009 at 16:58

3 For those who remember Knuth's famous quote about optimization, this is where to start. Then, if you find that you need the extra thousandth of a millisecond performance, go with one of the other techniques. – John Feb 25, 2014 at 19:02

19

A regular expression will look like:

```
public string RemoveSpecialChars(string input)
{
    return Regex.Replace(input, @"[^0-9a-zA-Z\._]", string.Empty);
}
```

But if performance is highly important, I recommend you to do some benchmarks before selecting the "regex path"...

Share Improve this answer Follow

answered Jul 13, 2009 at 15:42

Christian C. Salvadó

826k ● 184 ● 927 ● 843

16

I suggest creating a simple lookup table, which you can initialize in the static constructor to set any combination of characters to valid. This lets you do a quick, single check.

edit

Also, for speed, you'll want to initialize the capacity of your StringBuilder to the length of your input string. This will avoid reallocations. These two methods together will give you both speed and flexibility.

another edit

I think the compiler might optimize it out, but as a matter of style as well as efficiency, I recommend foreach instead of for.

Share

Improve this answer

Follow

edited Jul 13, 2009 at 15:44

answered Jul 13, 2009 at 15:39

Steven Sudit

19.6k ● 1 ● 52 ● 54

For arrays, `for` and `foreach` produce similar code. I don't know about strings though. I doubt that the JIT knows about the array-like nature of String. – Christian Klauser Jul 13, 2009 at 15:52

1 I bet the JIT knows more about the array-like nature of string than your [joke removed]. Anders etal did a lot of work optimizing everything about strings in .net – user1228 Jul 13, 2009 at 16:02

I've done this using HashSet<char> and it is about 2x slower than his method. Using bool[] is barely faster (0.0469ms/iter v. 0.0559ms/iter) than the version he has in OP...with the problem of being less readable. – user7116 Jul 13, 2009 at 16:36

1 I couldn't see any performance difference between using a bool array and an int array. I would use a bool array, as it brings down the lookup table from 256 kb to 64 kb, but it's still a lot of data for such a trivial function... And it's only about 30% faster. – Guffa Jul 13, 2009 at 17:07

1 @Guffa 2) Given that we're only keeping alphanumerics and a few Basic Latin characters, we only need a table for the low byte, so size isn't really an issue. If we wanted to be general-purpose, then the standard Unicode technique is double-indirection. In other words, a table of 256 table references, many of which point to the same empty table. – Steven Sudit Jul 13, 2009 at 18:17

16

```
public static string RemoveSpecialCharacters(string str)
{
    char[] buffer = new char[str.Length];
    int idx = 0;
```

```
    foreach (char c in str)
    {
        if ((c >= '0' && c <= '9') || (c >= 'A' && c <= 'Z')
            || (c >= 'a' && c <= 'z') || (c == '.') || (c == '_'))
        {
            buffer[idx] = c;
            idx++;
        }
    }

    return new string(buffer, 0, idx);
}
```

Share Improve this answer Follow

answered Jul 13, 2009 at 16:06

LukeH

269k ● 59 ● 370 ● 411

1 +1, tested and it is about 40% faster than StringBuilder. 0.0294ms/string v. 0.0399ms/string – user7116 Jul 13, 2009 at 17:14

Just to be sure, do you mean StringBuilder with or without pre-allocation? – Steven Sudit Jul 13, 2009 at 19:28

With pre-allocation, it is still 40% slower than the char[] allocation and new string. – user7116 Jul 14, 2009 at 2:28

2 I like this. I tweaked this method `foreach (char c in input.Where(c => char.IsLetterOrDigit(c) || allowedSpecialCharacters.Any(x => x == c))) buffer[idx++] = c;` – Chris Marisic Oct 17, 2012 at 15:47

14

If you're using a dynamic list of characters, LINQ may offer a much faster and graceful solution:

```
public static string RemoveSpecialCharacters(string value, char[]
specialCharacters)
{
    return new String(value.Except(specialCharacters).ToArray());
}
```

I compared this approach against two of the previous "fast" approaches (release compilation):

- Char array solution by LukeH - 427 ms
- StringBuilder solution - 429 ms
- LINQ (this answer) - 98 ms

Note that the algorithm is slightly modified - the characters are passed in as an array rather than hard-coded, which could be impacting things slightly (ie/ the other

solutions would have an inner foor loop to check the character array).

If I switch to a hard-coded solution using a LINQ where clause, the results are:

- Char array solution - 7ms
- StringBuilder solution - 22ms
- LINQ - 60 ms

Might be worth looking at LINQ or a modified approach if you're planning on writing a more generic solution, rather than hard-coding the list of characters. LINQ definitely gives you concise, highly readable code - even more so than Regex.

Share Improve this answer Follow

answered Jul 4, 2012 at 21:31

ShadowChaser

5,600 ● 2 ● 33 ● 33

5 This approach looks nice, but it doesn't work - Except() is a set operation, so you will end up with only the first appearance of each unique character in the string. – McKenzieG1 Mar 2, 2017 at 17:50

5

I'm not convinced your algorithm is anything but efficient. It's O(n) and only looks at each character once. *You're not gonna get any better than that unless you magically know values before checking them.*

I would however initialize the capacity of your `StringBuilder` to the initial size of the string. I'm guessing your perceived performance problem comes from memory reallocation.

Side note: Checking `A`-`z` is not safe. You're including `[`, `\`, `]`, `^`, `_`, and `` ` ``...

Side note 2: For that extra bit of efficiency, put the comparisons in an order to minimize the number of comparisons. (At worst, you're talking 8 comparisons tho, so don't think too hard.) This changes with your expected input, but one example could be:

```
if (str[i] >= '0' && str[i] <= 'z' &&
    (str[i] >= 'a' || str[i] <= '9' ||  (str[i] >= 'A' && str[i] <= 'Z') ||
    str[i] == '_') || str[i] == '.')
```

Side note 3: If for whatever reason you REALLY need this to be fast, a switch statement may be faster. The compiler should create a jump table for you, resulting in only a single comparison:

```
switch (str[i])
{
    case '0':
    case '1':
    .
    .
    .
    case '.':
        sb.Append(str[i]);
        break;
}
```

Share

Improve this answer

Follow

edited Jul 13, 2009 at 15:59

answered Jul 13, 2009 at 15:43

lc.

116k • 20 • 161 • 188

1 I agree that you can't beat O(n) on this one. However, there is a cost per comparison which can be lowered. A table lookup has a low, fixed cost, while a series of comparisons is going to increase in cost as you add more exceptions. – Steven Sudit Jul 13, 2009 at 15:47

About side note 3, do you really think the jump table would be faster than table lookup? – Steven Sudit Jul 13, 2009 at 16:12

I ran the quick performance test on the switch solution, and it performs the same as the comparison. – Guffa Jul 13, 2009 at 16:54

@Steven Sudit - I'd venture they're actually about the same. Care to run a test? – lc. Jul 13, 2009 at 17:12

7 O(n) notation sometimes pisses me off. People will make stupid assumptions based on the fact the algorithm is already O(n). If we changed this routine to replace the str[i] calls with a function that retrieved the comparison value by constructing a one-time SSL connection with a server on the opposite side of the world... you damn sure would see a massive performance difference and the algorithm is STILL O(n). The cost of O(1) for each algorithm is significant and NOT equivalent! – darron Jul 13, 2009 at 17:47

5

You can use regular expresion as follows:

```
return Regex.Replace(strIn, @"[^\w\.@-]", "", RegexOptions.None,
TimeSpan.FromSeconds(1.0));
```

Share Improve this answer Follow

answered Dec 10, 2017 at 15:48

Giovanny Farto M.

1,588 • 18 • 21

It seems good to me. The only improvement I would make is to initialize the `StringBuilder` with the length of the string.

4

```
StringBuilder sb = new StringBuilder(str.Length);
```

Share Improve this answer Follow

answered Jul 13, 2009 at 15:42

bruno conde

48.2k ● 15 ● 102 ● 118

4

```
StringBuilder sb = new StringBuilder();

for (int i = 0; i < fName.Length; i++)
{
   if (char.IsLetterOrDigit(fName[i]))
    {
       sb.Append(fName[i]);
    }
}
```

Share

Improve this answer

Follow

edited Apr 28, 2010 at 3:16

sth

229k ● 56 ● 286 ● 368

answered Mar 27, 2010 at 19:32

Chamika Sandamal

24.3k ● 5 ● 66 ● 87

4

I agree with this code sample. The only different it I make it into Extension Method of string type. So that you can use it in a very simple line or code:

```
string test = "abc@#$123";
test.RemoveSpecialCharacters();
```

Thank to Guffa for your experiment.

```
public static class MethodExtensionHelper
    {
    public static string RemoveSpecialCharacters(this string str)
        {
            StringBuilder sb = new StringBuilder();
            foreach (char c in str)
            {
                if ((c >= '0' && c <= '9') || (c >= 'A' && c <= 'Z') || (c >=
'a' && c <= 'z') || c == '_')
                {
                    sb.Append(c);
                }
            }
            return sb.ToString();
        }
}
```

Share Improve this answer Follow

answered Dec 13, 2011 at 18:59

Tola Ch.
198 ● 1 ● 1 ● 7

4

Another way that attempts to improve performance by reducing allocations, especially if this function is called many times.

It works because you can guarantee the result won't be longer than the input, so the input and output can be passed without creating extra copies in memory. For this reason you can't use `stackalloc` to create the buffer array as this would require a copy out of the buffer.

```
public static string RemoveSpecialCharacters(this string str)
{
    return RemoveSpecialCharacters(str.AsSpan()).ToString();
}

public static ReadOnlySpan<char> RemoveSpecialCharacters(this
ReadOnlySpan<char> str)
{
    Span<char> buffer = new char[str.Length];
    int idx = 0;

    foreach (char c in str)
    {
        if (char.IsLetterOrDigit(c))
        {
            buffer[idx] = c;
            idx++;
        }
    }

    return buffer.Slice(0, idx);
}
```

Share

Improve this answer

Follow

edited Jan 2, 2022 at 12:24

answered Jan 2, 2022 at 12:10

James Westgate
11.4k ● 8 ● 65 ● 69

3

There are lots of proposed solutions here, some more efficient than others, but perhaps not very readable. Here's one that may not be the most efficient, but certainly usable for most situations, and is quite concise and readable, leveraging Linq:

```
string stringToclean = "This is a test.  Do not try this at home; you might get
hurt. Don't believe it?";

var validPunctuation = new HashSet<char>(". -");

var cleanedVersion = new String(stringToclean.Where(x => (x >= 'A' && x <= 'Z')
|| (x >= 'a' && x <= 'z') || validPunctuation.Contains(x)).ToArray());
```

```
var cleanedLowercaseVersion = new String(stringToclean.ToLower().Where(x => (x
>= 'a' && x <= 'z') || validPunctuation.Contains(x)).ToArray());
```

Share Improve this answer Follow

answered Jan 3, 2018 at 18:34

Steve Faiwiszewski
313 ● 2 ● 6

2

I would use a String Replace with a Regular Expression searching for "special characters", replacing all characters found with an empty string.

Share Improve this answer Follow

answered Jul 13, 2009 at 15:38

Stephen Wrighton
37.8k ● 6 ● 70 ● 87

+1 certainly less code and arguably more readable ignoring write-once Regex. – kenny Jul 13, 2009 at 16:38

1 @kenny - I agree. The original question even states that the strings are short - 10-30 chars. But apparently a lot of people still think we're selling CPU time by the second... – Tom Bushell Nov 12, 2011 at 0:10

Reguler expressin works so lazy.So it shouldn't be used always. – RockOnGom Jul 10, 2013 at 19:45

2

I had to do something similar for work, but in my case I had to filter all that is not a letter, number or whitespace (but you could easily modify it to your needs). The filtering is done client-side in JavaScript, but for security reasons I am also doing the filtering server-side. Since I can expect most of the strings to be clean, I would like to avoid copying the string unless I really need to. This let my to the implementation below, which should perform better for both clean and dirty strings.

```
public static string EnsureOnlyLetterDigitOrWhiteSpace(string input)
{
    StringBuilder cleanedInput = null;
    for (var i = 0; i < input.Length; ++i)
    {
        var currentChar = input[i];
        var charIsValid = char.IsLetterOrDigit(currentChar) ||
char.IsWhiteSpace(currentChar);

        if (charIsValid)
        {
            if(cleanedInput != null)
                cleanedInput.Append(currentChar);
        }
```

```
        else
        {
            if (cleanedInput != null) continue;
            cleanedInput = new StringBuilder();
            if (i > 0)
                cleanedInput.Append(input.Substring(0, i));
        }
    }

    return cleanedInput == null ? input : cleanedInput.ToString();
}
```

Share

Improve this answer

Follow

edited Nov 26, 2013 at 15:06

answered Aug 29, 2013 at 17:46

Daniel Blankensteiner

76 • 4

1

For S&G's, Linq-ified way:

```
var original = "(*^%foo)(@)&^@#><>?:\":';=-+_";
var valid = new char[] {
    'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o',
    'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', 'A', 'B', 'C', 'D',
    'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S',
    'T', 'U', 'V', 'W', 'X', 'Y', 'Z', '1', '2', '3', '4', '5', '6', '7', '8',
    '9', '0', '.', '_' };
var result = string.Join("",
    (from x in original.ToCharArray()
     where valid.Contains(x) select x.ToString())
        .ToArray());
```

I don't think this is going to be the most efficient way, however.

Share Improve this answer Follow

answered Jul 13, 2009 at 16:16

user1228

2 It's not, because it's a linear search. – Steven Sudit Jul 13, 2009 at 16:41

1

```
public string RemoveSpecial(string evalstr)
{
StringBuilder finalstr = new StringBuilder();
            foreach(char c in evalstr){
            int charassci = Convert.ToInt16(c);
            if (!(charassci >= 33 && charassci <= 47))// special char ???
             finalstr.append(c);
            }
return finalstr.ToString();
}
```

Share

Improve this answer

Follow

edited Apr 28, 2010 at 3:16

sth

229k ● 56 ● 286 ● 368

answered Apr 27, 2010 at 17:19

Shiko

11 ● 1

1

Use:

```
s.erase(std::remove_if(s.begin(), s.end(), my_predicate), s.end());

bool my_predicate(char c)
{
 return !(isalpha(c) || c=='_' || c==' '); // depending on you definition of
special characters
}
```

And you'll get a clean string `s`.

`erase()` will strip it of all the special characters and is highly customisable with the `my_predicate()` function.

Share

Improve this answer

Follow

edited Sep 24, 2012 at 0:18

Austin Henley

4,633 ● 13 ● 48 ● 80

answered Sep 23, 2012 at 8:02

Bhavya Agarwal

107 ● 1 ● 11

1

HashSet is O(1)

Not sure if it is faster than the existing comparison

```
private static HashSet<char> ValidChars = new HashSet<char>() { 'a', 'b', 'c',
'A', 'B', 'C', '1', '2', '3', '_' };
public static string RemoveSpecialCharacters(string str)
{
    StringBuilder sb = new StringBuilder(str.Length / 2);
    foreach (char c in str)
    {
        if (ValidChars.Contains(c)) sb.Append(c);
    }
    return sb.ToString();
}
```

I tested and this in not faster than the accepted answer.

I will leave it up as if you needed a configurable set of characters this would be a good solution.

Share

Improve this answer

Follow

edited Sep 24, 2013 at 20:24

answered Sep 24, 2013 at 19:29

paparazzo

45.1k ● 24 ● 109 ● 179

Why do you think that the comparison is not O(1)? – Guffa Sep 24, 2013 at 19:47

@Guffa I am not sure it is not and I removed my comment. And +1. I should have done more testing before making the comment. – paparazzo Sep 24, 2013 at 20:16

1

I wonder if a Regex-based replacement (possibly compiled) is faster. ~~Would have to test that~~ Someone has found this to be ~5 times slower.

Other than that, you should initialize the StringBuilder with an expected length, so that the intermediate string doesn't have to be copied around while it grows.

A good number is the length of the original string, or something slightly lower (depending on the nature of the functions inputs).

Finally, you can use a lookup table (in the range 0..127) to find out whether a character is to be accepted.

Share

Improve this answer

Follow

edited Sep 25, 2013 at 17:38

answered Jul 13, 2009 at 15:50

Christian Klauser

4,466 ● 3 ● 32 ● 43

A regular expression has been tested already, and it's about five times slower. With a lookup table in the range 0..127 you still have to range check the character code before using the lookup table, as characters are 16 bit values, not 7 bit values. – Guffa Sep 24, 2013 at 21:56

@Guffa Err... yes? ;) – Christian Klauser Sep 25, 2013 at 17:39

1

The following code has the following output (conclusion is that we can also save some memory resources allocating array smaller size):

```
lookup = new bool[123];

for (var c = '0'; c <= '9'; c++)
{
    lookup[c] = true; System.Diagnostics.Debug.WriteLine((int)c + ": " +
(char)c);
}

for (var c = 'A'; c <= 'Z'; c++)
{
    lookup[c] = true; System.Diagnostics.Debug.WriteLine((int)c + ": " +
(char)c);
}

for (var c = 'a'; c <= 'z'; c++)
{
    lookup[c] = true; System.Diagnostics.Debug.WriteLine((int)c + ": " +
(char)c);
```

```
}

48: 0
49: 1
50: 2
51: 3
52: 4
53: 5
54: 6
55: 7
56: 8
57: 9
65: A
66: B
67: C
68: D
69: E
70: F
71: G
72: H
73: I
74: J
75: K
76: L
77: M
78: N
79: O
80: P
81: Q
82: R
83: S
84: T
85: U
86: V
87: W
88: X
89: Y
90: Z
97: a
98: b
99: c
100: d
101: e
102: f
103: g
104: h
105: i
106: j
107: k
108: l
109: m
110: n
111: o
112: p
113: q
114: r
115: s
116: t
117: u
118: v
119: w
120: x
```

```
121: y
122: z
```

You can also add the following code lines to support Russian locale (array size will be 1104):

```
for (var c = 'A'; c <= 'Я'; c++)
{
    lookup[c] = true; System.Diagnostics.Debug.WriteLine((int)c + ": " +
(char)c);
}

for (var c = 'a'; c <= 'я'; c++)
{
    lookup[c] = true; System.Diagnostics.Debug.WriteLine((int)c + ": " +
(char)c);
}
```

Share

Improve this answer

Follow

edited Nov 26, 2013 at 15:19

rhughes

9,563 ● 11 ● 63 ● 91

answered Apr 19, 2012 at 14:16

Pavel Shkleinik

6,337 ● 2 ● 27 ● 36

0

I'm not sure it is the most efficient way, but It works for me

```
 Public Function RemoverTildes(stIn As String) As String
    Dim stFormD As String = stIn.Normalize(NormalizationForm.FormD)
    Dim sb As New StringBuilder()

    For ich As Integer = 0 To stFormD.Length - 1
        Dim uc As UnicodeCategory =
CharUnicodeInfo.GetUnicodeCategory(stFormD(ich))
        If uc <> UnicodeCategory.NonSpacingMark Then
            sb.Append(stFormD(ich))
        End If
    Next
    Return (sb.ToString().Normalize(NormalizationForm.FormC))
End Function
```

Share Improve this answer Follow

answered Aug 25, 2015 at 0:16

RonaldPaguay

335 ● 3 ● 12

The answer *does* work, but the question was for **C#.** (P.S: I know this was practically five years ago, but still..) I used the Telerik VB to C# Converter, (And vice-versa) and the code worked just fine - not sure about anyone else, though. (Another thing, converter.telerik.com) – Momoro Apr 18, 2020 at 3:29

Shortest way just a 3 line...

```
public static string RemoveSpecialCharacters(string str)
{
    var sb = new StringBuilder();
    foreach (var c in str.Where(c => c >= '0' && c <= '9' || c >= 'A' && c <=
'Z' || c >= 'a' && c <= 'z' || c == '.' || c == '_')) sb.Append(c);
    return sb.ToString();
}
```

Share Improve this answer Follow

answered May 19, 2021 at 16:00

Erçin Dedeoğlu

5,344 ● 4 ● 51 ● 71

If you need to clean up the input string in case of injections or typos (rare events), the fastest way is to use the `switch()` to check all characters (the compiler does a good job of optimizing the execution time of `switch()`) plus the additional code to remove the unwanted characters if there were found. Here is the solution:

```
    public static string RemoveExtraCharacters(string input)
    {
        if (string.IsNullOrEmpty(input))
            return "";

        input = input.Trim();

        StringBuilder sb = null;

    reStart:
        if (!string.IsNullOrEmpty(input))
        {
            var len = input.Length; ;

            for (int i = 0; i < len; i++)
            {
                switch (input[i])
                {
                    case '0':
                    case '1':
                    case '2':
                    case '3':
                    case '4':
                    case '5':
                    case '6':
                    case '7':
                    case '8':
                    case '9':
                    case 'A':
                    case 'B':
                    case 'C':
                    case 'D':
                    case 'E':
                    case 'F':
                    case 'G':
                    case 'H':
                    case 'I':
```

```
                    case 'J':
                    case 'K':
                    case 'L':
                    case 'M':
                    case 'N':
                    case 'O':
                    case 'Q':
                    case 'P':
                    case 'R':
                    case 'S':
                    case 'T':
                    case 'U':
                    case 'V':
                    case 'W':
                    case 'X':
                    case 'Y':
                    case 'Z':
                    case 'a':
                    case 'b':
                    case 'c':
                    case 'd':
                    case 'e':
                    case 'f':
                    case 'g':
                    case 'h':
                    case 'i':
                    case 'j':
                    case 'k':
                    case 'l':
                    case 'm':
                    case 'n':
                    case 'o':
                    case 'q':
                    case 'p':
                    case 'r':
                    case 's':
                    case 't':
                    case 'u':
                    case 'v':
                    case 'w':
                    case 'x':
                    case 'y':
                    case 'z':
                    case '/':
                    case '_':
                    case '-':
                    case '+':
                    case '.':
                    case ',':
                    case '*':
                    case ':':
                    case '=':
                    case ' ':
                    case '^':
                    case '$':
                        break;

                    default:
                        if (sb == null)
                            sb = new StringBuilder();

                        sb.Append(input.Substring(0, i));
```

```
                        if (i + 1 < len)
                        {
                            input = input.Substring(i + 1);
                            goto reStart;
                        }
                        else
                            input = null;
                        break;
                    }
                }
            }

            if (sb != null)
            {
                if (input != null)
                    sb.Append(input);
                return sb.ToString();
            }

            return input;
        }
```

Share

Improve this answer

Follow

edited Jun 27, 2022 at 22:55

answered Jun 27, 2022 at 21:48

Mykola Uspalenko

106 ● 1 ● 7

0

The following only does the minimum allocations implied by the function signature:

```
static bool IsOkChar(char c) => (c >= '0' && c <= '9') ||
    (c >= 'A' && c <= 'Z') || (c >= 'a' && c <= 'z') ||
    c == '.' || c == '_';

public static string RemoveChars(string s) {
    int newLen = 0;
    foreach (char c in s) {
        if (IsOkChar(c)) {
            newLen++;
        }
    }
    if (newLen == s.Length) {
        return s;
    }
    return string.Create(newLen, s, static (dst, s) => {
        int i = 0;
        foreach (char c in s) {
            if (IsOkChar(c)) {
                dst[i++] = c;
            }
        }
    });
}
```

This `string.Create()` overload did not exist when this question was first asked. It uses a callback taking a span to grant temporary access to the new string's memory.

In a larger application, the intermediate strings might not be optimal compared to mutating buffers more directly, but the maintainability advantage of strings is probably worth it. On the other hand, for a frequently used function, the optimization (or non-pessimization) using `string.Create()` seems useful.

Share Improve this answer Follow

answered Mar 20 at 22:49

jilles

11.1k ● 2 ● 27 ● 39

0

Using Regular Expression.

```
string pattern = @"(\s+|\`|~|!|@|#|\$|%|\^|\&|\*|\(|\)|_|\+|\-|\=|\[|\]|\\|\||\
{|\}|;|'|\<|\>|\?|\,|\.|\/)";
string s = "Go $@ 123 west @ Life is$ peaceful ~!@# there  سيبشبسيبسيب سبسيب
~`!@#$%^&*()_+=- {}| []\\ ;' <>? ,./";
s = Regex.Replace(s, pattern, " ");
s = Regex.Replace(s, @"\s{2,}", " ");
Console.WriteLine(s.Trim());
```

Share Improve this answer Follow

answered Sep 18 at 9:30

Sender

6,848 ● 12 ● 49 ● 69

0

As the string will be short you can use Linq to do this with negligible perform differences

```
public string RemoveSpecialChars(string str) =>
string.IsNullOrWhiteSpace(str)
    ? str
    : string.Concat(str.Where(c => char.IsLetterOrDigit(c) || c == '.' || c ==
'_'));
```

Share Improve this answer Follow

answered 2 days ago

Maze90

138 ● 1 ● 17

-1

```
public static string RemoveAllSpecialCharacters(this string text) {
  if (string.IsNullOrEmpty(text))
    return text;

  string result = Regex.Replace(text, "[:!@#$%^&*(){|\":?><\\[\\]\\;'/.,~]", "
");
  return result;
}
```

Share

Improve this answer

Follow

edited Jun 13, 2020 at 11:59

kyun

10.2k ● 9 ● 35 ● 79

answered Jun 13, 2020 at 11:26

Hasan_H

87 ● 4

Answer is wrong. If you are gonna use regex, it should be inclusive, not exclusive one, because you miss some characters now. Actually, there is already answer with regex. And to be full - regex is SLOWER then direct compare chars function. – TPAKTOPA Jun 13, 2020 at 13:26

-3

If you're worried about speed, use pointers to edit the existing string. You could pin the string and get a pointer to it, then run a for loop over each character, overwriting each invalid character with a replacement character. It would be extremely efficient and would not require allocating any new string memory. You would also need to compile your module with the unsafe option, and add the "unsafe" modifier to your method header in order to use pointers.

```
static void Main(string[] args)
{
    string str = "string!$%with^&*invalid!!characters";
    Console.WriteLine( str ); //print original string
    FixMyString( str, ' ' );
    Console.WriteLine( str ); //print string again to verify that it has been
modified
    Console.ReadLine(); //pause to leave command prompt open
}


public static unsafe void FixMyString( string str, char replacement_char )
{
    fixed (char* p_str = str)
    {
        char* c = p_str; //temp pointer, since p_str is read-only
        for (int i = 0; i < str.Length; i++, c++) //loop through each character
in string, advancing the character pointer as well
            if (!IsValidChar(*c)) //check whether the current character is
invalid
                (*c) = replacement_char; //overwrite character in existing
string with replacement character
    }
}

public static bool IsValidChar( char c )
{
    return (c >= '0' && c <= '9') || (c >= 'A' && c <= 'Z') || (c >= 'a' && c
<= 'z') || (c == '.' || c == '_');
    //return char.IsLetterOrDigit( c ) || c == '.' || c == '_'; //this may work
as well
}
```

Share

Improve this answer

edited Jul 13, 2009 at 16:26

answered Jul 13, 2009 at 15:57

Follow

Triynko
19.2k ● 21 ● 112 ● 181

15 Noooooooooo! Changing a string in .NET is BAAAAAAAAAAAD! Everything in the framework relies on the rule that strings are immutable, and if you break that you can get very surprising side effects... – Guffa Jul 13, 2009 at 16:52

-3

```
public static string RemoveSpecialCharacters(string str){
    return str.replaceAll("[^A-Za-z0-9_\\\\.]", "");
}
```

Share

Improve this answer

Follow

edited Dec 18, 2012 at 14:33

Rory McCrossan
337k ● 41 ● 319 ● 350

answered Dec 18, 2012 at 14:14

Jawaid
11 ● 1

1 I'm afraid `replaceAll` is not C# String function but either Java or JavaScript – Csaba Toth Sep 27, 2013 at 18:38

1 2 Next

Highly active question. Earn 10 reputation (not counting the association bonus) in order to answer this question. The reputation requirement helps protect this question from spam and non-answer activity.