

the trusted technology learning source

[Home](#) > [Articles](#) > [Programming](#) > [C/C++](#)

C++11 Regular-Expression Library



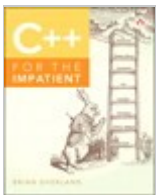
By [Brian Overland](#)

Jun 25, 2013

[Contents](#) [Print](#) [Share This](#)

[< Back](#) [Page 6 of 10](#) [Next >](#)

This chapter is from the book



[C++ for the Impatient](#)

[Learn More](#)

[Buy](#)

20.6. Replacing Text

One of the most powerful regular-expression capabilities is to selectively search-and-replace patterns within a string of text. Here's one possible use (out of zillions): to transform a target string by replacing each repeated pair of words with just one word.

For example, given this text:

```
The cow cow jumped over the the moon.
```

it would be useful to produce a string consisting of:

```
The cow jumped over the moon.
```

The `regex_replace` function performs this task by returning the transformed string. It has the following syntax:

```
regex_replace(target_string, regex_obj, replacement_pattern_str);
```

Related Resources

[Store](#)

[Articles](#)

[Blogs](#)



**[Game Programming in C++:
Creating 3D Games](#)**

By [Sanjay Madhav](#)

Book \$39.99



**[Revel for Introduction to C++
Programming -- Access Card,
4th Edition](#)**

By [Y. Daniel Liang](#)

Book \$73.67



**[C++ Templates: The
Complete Guide, 2nd Edition](#)**

By [David Vandevor](#), [Nicolai
M. Josuttis](#), [Douglas Gregor](#)

Book \$63.99

[See All Related Store Items](#)

The `replacement_pattern_str` is a string that can contain the following special sequences (in addition to ordinary characters).

► `$&`

Refers to the entire matched string.

► `$n`

Refers to the *n*th group within the matched string. For example, “\$1” refers to the first group of characters tagged by the regex object; “\$2” refers to the second group of tagged characters (if there is one), and so on. The example that follows should clarify.

► `$$`

A literal dollar sign (\$).

The following declarations set up a search-and-replace designed to fix the repeated-word pattern, replacing it, where found, with one copy of the word.

```
using std::regex;
using std::regex_replace;
using std::string;

regex reg1("[A-Za-z]+) \\1"); // Find double word.
string replacement = "$1";    // Replace with one word.
```

With these objects defined, the following statements execute search-and-replace on the string shown earlier.

```
string target = "The cow cow jumped over the the moon.";
string result = regex_replace(target, reg1, replacement);
std::cout << result << std::endl;
```

The output is:

```
The cow jumped over the moon.
```

which is what we wanted.

Let’s review how this works. When the text “cow cow” was matched by the regular-expression object, the first occurrence of “cow” was tagged because it matched the expression inside the parentheses: “[A-Za-z]+”. The rest of the expression, “\\1”, indicated that the regex object then needed to match a space, followed by a recurrence of the tagged characters, to match the overall expression. Therefore, “cow” gets tagged and “cow cow” matches the entire regular expression.

The replacement pattern, “\$1”, causes the matched text—“cow cow”—to be replaced by “cow”, the tagged group. Suppose the replacement pattern were “XX\$1YY\$1ZZ\$1”. Then the replacement text would have been “XXcowYYcowZZcow” and *that* would have replaced “cow cow”.

Characters not matched by the regex object, `reg1`, are just copied into the result as they are. So, for example, the words “jumped over” are copied without being transformed.

Here’s an example of another regex object and replacement-pattern string: When used with the call to `regex_replace` shown earlier, these result in the switching of two words separated by an ampersand (&). For example, “boy&girl” would be replaced by “girl&boy” and vice versa.

```
regex reg1("([A-Za-z]+)&([A-Za-z]+)"); // Find word&word
string replacement = "$2&$1";          // Switch order.
```

The **`regex_replace`** function is particularly convenient. It isn't necessary to iterate through the target string. Instead, **`regex_replace`** carries out replacements on all the substrings matching the pattern in the regex object, while leaving the rest of the text alone.

[!\[\]\(4729e517bc6a7cd81c8025b9646574fb_img.jpg\) Share This](#) [!\[\]\(90a2fb2f2c617b26262139ae4159c0a0_img.jpg\) Save To Your Account](#)

[< Back](#) **Page 6 of 10** [Next >](#)