C++11 Regular-Expression Library

By Brian Overland

Jun 25, 2013

This chapter is from the book

C++ for the Impatient

Learn More Buy

20.4. Matching and Searching Functions

This section summarizes the syntax for the matching and searching functions. Don't forget to include the <regex> header. Also, remember that regex symbols are part of the **std** namespace, so either include a **using** statement or identify each name with its **std** prefix.

```
#include <regex>
using std::regex;
```

Next, you need to specify a regular-expression pattern by creating a **regex** object:

```
regex  name(pattern_string [,flags])
```

This regex constructor includes an optional *flags* argument. The most useful flag is **regex_constants::icase**, which turns off case sensitivity.

regex objects are unusual in that they are compiled and built at runtime, not compile time. Consequently, you want to create as few **regex** objects as possible, because creating many s objects impacts runtime performance. You should make your **regex** objects global variables them by reference if you need to.

After constructing a **regex** object, you can perform matching and searching by calling the

This chapter is from the book

C++ for the Impatient

Learn More Buy

regex_match and **regex_search** functions. Each of these returns a Boolean value. The **regex_match** function returns **true** if the *target_string* matches the pattern stored in *regex_obj* exactly: That is, the entire *target_string* must match the pattern completely.

```
regex_match(target_string, regex_obj)
```

The **regex_search** function returns **true** if the target string contains any substring that matches the pattern stored in *regex_obj*.

```
regex_search(target_string, regex_obj)
```

For example, consider the task of finding a repeated word in the sentence:

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

The following statements execute this search:

```
#include <regex>
#include <string>
#include <iostream>
using std::string;
using std::cout;
using std::endl;
. . .
std::regex reg1("([A-Za-z]+) \\1");
string target = "The the cow jumped over the the moon.";
if (std::regex_search(target, reg1)) {
     cout << "A repeated word was found." << endl;
}
```

The call to **regex_search** failed to find the first repeated-word occurrence (“The the”) because the first word was capitalized and the second was not. That did not matter in this instance, because there was another repeated-word sequence (“the the”). However, sometimes you are interested in getting the position of the first match, so it can matter.

To find the position of the first substring found, it’s necessary to include another argument:

```
regex_search(target_string, match_info, regex_obj)
```

The *match_info* argument is an object of type **cmatch**, **smatch**, or **wmatch**, corresponding to the following formats: C-string, **string** object, and wide-character string. The format must match the type of *target_string*; in this case, the **string** class is used for the target string, so *match_info* must have type **smatch**.

For example:

```
std::smatch sm;
std::regex reg1("([A-Za-z]+) \\1");
std::string target="The the cow jumped over the the moon.";
bool b = std::regex_search(target, sm, reg1);
```

The *match_info* object (sm in this case) can be used to obtain information about the search, as follows:

```
match_obj.str()              // Return the matched string
match_obj.position()     // Return the position of the matched string
```

For example, the following call finds the position of the first repeated word:

```
std::smatch sm;
std::regex reg1("([A-Za-z]+) \\1");
std::string target="The the cow jumped over the the moon.";
bool b = std::regex_search(target, sm, reg1);
cout << "Match found at pos. " << sm.position() << endl;
cout << "Pattern found was: " << sm.str() << endl;
```

These statements print:

```
Match was found at pos. 24
Pattern found was: the the
```

Turning off case sensitivity causes a match to be found at position 0 instead.

```
std::regex reg1("([A-Za-z]+) \\1", regex_constants::icase);
```

You can also obtain information about groups—patterns enclosed in parentheses—within the matched string. This information only applies to groups within the *first* substring found; **regex_search** finds only one substring and then it stops searching. (The next section describes how to find multiple substrings.)

```
match_obj.str(n)         // Return nth group within the matched subst
match_obj.position(n)    // Return position of nth group
```

For example, you can use *match_object* (sm in this case) to get information on the group found *within* the first matched substring.

```
cout << "Text of sub-group: " << sm.str(1);
```

Assuming case sensitivity is turned off, this prints:

```
Text of sub-group: The
```

Share This Save To Your Account

< Back **Page 4** of 10 Next >