C++11 Regular-Expression Library

By Brian Overland

Jun 25, 2013

This chapter is from the book

C++ for the Impatient

Learn More Buy

20.7. String Tokenizing

Although the functionality in the preceding sections can perform nearly any form of pattern matching, C++11 also provides string-tokenizing functionality that is a superior alternative to the C-library **strtok** function. Tokenization is the process of breaking a string into a series of individual words, or *tokens*.

To take advantage of this feature, use the following syntax, in which *str* represents a **string** object containing the target string:

```
sregex_token_iterator  iter_name(str.begin(), str.end(), regex_obj,
sregex_token_iterator  end_iter_name;
```

As with **sregex_iterator**, **sregex_token_iterator** is an adapter built on top of the **string** class; you can use the underlying template, **regex_token_iterator**, with other kinds of strings.

sregex_token_iterator performs a range of operations, most of which are similar to what the standard iterator does, as described in Section 20.5, ""Find All," or Iterative Searches." Specifying -1 as the fourth argument makes the function skip over any patterns matching the *regex_obj*, causing the iterator to iterate through the tokens—which consist of text between each occurrence of the pattern.

For example, the following statements find each word, in which words are delimited by any series of spaces and/or commas.

```
#include <regex>
#include <string>
using std::regex;
using std::string;
using std::sregex_token_iterator;
. . .
// Delimiters are spaces (\s) and/or commas
regex re("[\\s,]+");
string s = "The White Rabbit,  is very,late.";
sregex_token_iterator it(s.begin(), s.end(), re, -1);
sregex_token_iterator reg_end;
for (; it != reg_end; ++it) {
     std::cout << it->str() << std::endl;
}
```

These statements, when executed, print the following, ignoring spaces and commas (except as to recognize them as delimiters):

```
The
White
Rabbit
is
very
late.
```