
14.3. Regex Iterators

To iterate over all matches of a regular search, we can also use regex iterators. These iterators are of type `regex_iterator<>` and have the usual instantiations for strings and character sequences with prefixes `s`, `c`, `ws`, or `wc`. Consider the following example:

Click here to view code image

```
// regex/regexiter1.cpp

#include <string>
#include <regex>
#include <iostream>
#include <algorithm>
using namespace std;

int main()
{
    string data = "<person>\n"
                  " <first>Nico</first>\n"
                  " <last>Josuttis</last>\n"
                  "</person>\n";

    regex reg("<(.*)>(.*)</(\\1)>");

    // iterate over all matches (using a regex_iterator):
    sregex_iterator pos(data.cbegin(),data.cend(),reg);
    sregex_iterator end;
    for ( ; pos!=end ; ++pos ) {
        cout << "match:  " << pos->str() << endl;
        cout << " tag:   " << pos->str(1) << endl;
        cout << " value: " << pos->str(2) << endl;
    }

    // use a regex_iterator to process each matched substring as element in an algorithm:
    sregex_iterator beg(data.cbegin(),data.cend(),reg);
    for_each (beg,end,[](const smatch& m) {
                          cout << "match:  " << m.str() << endl;
                          cout << " tag:   " << m.str(1) << endl;
                          cout << " value: " << m.str(2) << endl;
                      });
}
```

Here, with

```
sregex_iterator pos(data.cbegin(),data.cend(),reg);
```

we initialize a regex iterator, iterating over data to search for matches of `reg`. The default constructor of this type defines a past-the-end iterator:

```
sregex_iterator end;
```

We can now use this iterator as any other bidirectional iterator (see Section 9.2.4, page 437): Operator `*` yields the current match, while operators `++` and `--` move to the next or previous match. Thus, the following prints all the matches, their tags, and their values (as in the previous example):

Click here to view code image

```
for ( ; pos!=end ; ++pos ) {
    cout << "match:  " << pos->str() << endl;
    cout << " tag:   " << pos->str(1) << endl;
    cout << " value: " << pos->str(2) << endl;
}
```

And, of course, you can use such an iterator in an algorithm. Thus, the following calls the lambda passed as third argument for each match (see Section 6.9, page 229, for details about lambdas and algorithms):

Click here to view code image

```
// use a regex_iterator to process each matched substring as element in an algorithm:
sregex_iterator beg(data.cbegin(),data.cend(),reg);
sregex_iterator end;
for_each (beg,end,[](const smatch& m) {
                     cout << "match:  " << m.str() << endl;
                     cout << " tag:   " << m.str(1) << endl;
                     cout << " value: " << m.str(2) << endl;
                  });
```