Repeated Search

We will now introduce the concept of iterators in regular expressions.

WE'LL COVER THE FOLLOWING ^

- std::regex_iterator
- std::regex_token_iterator

It's quite convenient to iterate with std::regex_token_iterator over the matched texts. std::regex_iterator supports the matches and their capture groups. std::regex_token_iterator supports more. You can address the components of each capture and by using a negative index, your can access the text between the matches.

std::regex_iterator

C++ defines the following four type synonyms for std::regex_iterator.

```
typedef cregex_iterator
typedef wcregex_iterator
typedef sregex_iterator
typedef wsregex_iterator
typedef wsregex_iterator
typedef wsregex_iterator
typedef wsregex_iterator
regex_iterator<const char*>
regex_iterator<const wchar_t*>
regex_iterator<std::string::const_iterator>
typedef wsregex_iterator
```

You can use std::regex_iterator to count the occurrences of the words in a
text:

```
#include <regex>

#include <iostream>
#include <string>
#include <unordered_map>

int main(){

std::cout << std::endl;

// Bjarne Stroustrup about C++0x on http://www2.research.att.com/~bs/C++0xFAQ.html
std::string text{"That's a (to me) amazingly frequent question. It may be the most frequent</pre>
```

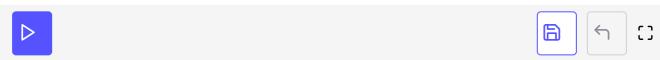
```
// regular expression for a word
std::regex wordReg(R"(\w+)");

// get all words from text
std::sregex_iterator wordItBegin(text.begin(), text.end(), wordReg);
const std::sregex_iterator wordItEnd;

// use unordered_map to count the wourds
std::unordered_map<std::string, std::size_t> allWords;

// count the words
for (; wordItBegin != wordItEnd;++wordItBegin){
    ++allWords[wordItBegin->str()];
}

for ( auto wordIt: allWords) std::cout << wordIt.first << ": " << wordIt.second << "\n";
std::cout << "\n\n";
}</pre>
```



std::regex_iterator

A word consists of a least one character (\w+). This regular expression is used to define the begin iterator wordItBegin and the end iterator wordItEnd. The iteration through the matches happens in the for loop. Each word increments the counter: ++allWords[wordItBegin]->str()]. A word with counter equals to 1 is created if it is not already in allWords.

std::regex_token_iterator

C++ defines the following four type synonyms for std::regex_token_iterator.

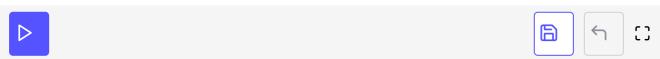
```
typedef cregex_iterator
typedef wcregex_iterator
typedef sregex_iterator
typedef sregex_iterator
typedef wsregex_iterator
typedef wsregex_iterator

regex_iterator<const char*>
regex_iterator<const wchar_t*>
regex_iterator<std::string::const_iterator>
regex_iterator<std::wstring::const_iterator>
```

std::regex_token_iterator enables you by using indexes to specify which capture groups you are interested in explicitly. If you don't specify the index, you will get all capture groups, but you can also request specific capture groups using its respective index. The -1 index is special: You can use -1 to address the text between the matches:

...

```
#include <regex>
#include <iostream>
#include <string>
#include <vector>
int main(){
  std::cout << std::endl;</pre>
  // a few books
  std::string text{"Pete Becker, The C++ Standard Library Extensions, 2006:Nicolai Josuttis,
  // regular expression for a book
  std::regex regBook(R''((\w+)\s(\w+), ([\w\s+]*), (\d{4}))");
  // get all books from text
  std::sregex_token_iterator bookItBegin(text.begin(), text.end(), regBook);
  const std::sregex_token_iterator bookItEnd;
  std::cout << "##### std::match_results ######" << "\n\n";
  while ( bookItBegin != bookItEnd){
    std::cout << *bookItBegin++ << std::endl;</pre>
  std::cout << "\n\n" << "#### last name, date of publication ######" << "\n\n";
  // get all last name and date of publication for the entries
  std::sregex_token_iterator bookItNameIssueBegin(text.begin(), text.end(), regBook, {{2, 4}}
  const std::sregex_token_iterator bookItNameIssueEnd;
  while ( bookItNameIssueBegin != bookItNameIssueEnd){
      std::cout << *bookItNameIssueBegin++ << ", ";</pre>
      std::cout << *bookItNameIssueBegin++ << std::endl;</pre>
  }
  // regular expression for a book, using negativ search
  std::regex regBookNeg(":");
  std::cout << "\n\n" << "#### get each entry, using negativ search ######" << "\n\n";
  // get all entries, only using ":" as regular expression
  std::sregex_token_iterator bookItNegBegin(text.begin(), text.end(), regBookNeg, -1);
  const std::sregex_token_iterator bookItNegEnd;
   while ( bookItNegBegin != bookItNegEnd){
       std::cout << *bookItNegBegin++ << std::endl;</pre>
   }
  std::cout << std::endl;</pre>
}
```



bookItBegin using no indices and bookItNegbegin using the negative index returns both the total capture group, but bookNameIssueBegin only the second and fourth capture group {{2,4}}.