The Search Result

Whenever we verify if a piece of text satisfies our regular expression, we have to store the results somewhere. std::match_results allows us to do just that.

WE'LL COVER THE FOLLOWING

- std::sub_match

The object of type `std::match_results` is the result of an `std::regex_match` or `std::regex_search`. `std::match_results` is a sequential container having at least one capture group of an `std::sub_match` object. The `std::sub_match` objects are sequences of characters.

> **ℹ What is a capture group?**
> Capture groups allow us to further analyse the search results of a regular expression. They are defined by a pair of parentheses `()`. The regular expression `((a+)(b+)(c+))` has four capture groups: `((a+)(b+)(c+))`, `(a+)`, `(b+)`, and `(c+)`. The total result is the 0th capture group.

C++ has four types of synonyms of type `std::match_results`:

```
typedef match_results<const char*> cmatch;
typedef match_results<const wchar_t*> wcmatch;
typedef match_results<string::const_iterator> smatch;
typedef match_results<wstring::const_iterator> wsmatch;
```

The result of search function `std::smatch smatch` has a powerful interfaces:

Method	Description
`smatch.size()`	Returns the number of capture groups

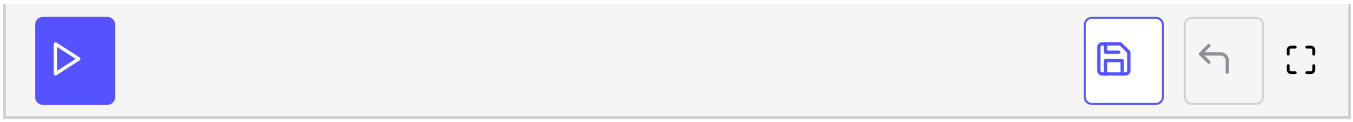
	groups.
`smatch.empty()`	Returns if the search result has a capture group.
`smatch[i]`	Returns the ith capture group.
`smatch.length(i)`	Returns the length of the ith capture group.
`smatch.position(i)`	Returns the position of the ith capture group.
`smatch.str(i)`	Returns the ith capture group as string.
`smatch.prefix()` and `smatch.suffix()`	Returns the string before and after the capture group.
`smatch.begin()` and `smatch.end()`	Returns the begin and end iterator for the capture groups.
`smatch.format(...)`	Formats `std::smatch` objects for the output.

Interface of std::smatch

The following program shows the output of the first four capture groups for different regular expressions.

```
#include <regex>

#include <iomanip>
#include <iostream>
#include <string>

void showCaptureGroups(const std::string& regEx, const std::string& text){

  // regular expression holder
  std::regex rgx(regEx);
```

```
  // result holder
  std::smatch smatch;


  // result evaluation
  if (std::regex_search(text, smatch, rgx)){
    std::cout << std::setw(10) << regEx << std::setw(30) << text << std::setw(30) << smatch[0
  }

}

int main(){

  std::cout << std::endl;

  std::cout << std::setw(10) << "reg Expr" << std::setw(30) << "text" << std::setw(30) << "sm

  showCaptureGroups("abc+", "abccccc");

  showCaptureGroups("(a+)(b+)(c+)", "aaabccc");

  showCaptureGroups("((a+)(b+)(c+))", "aaabccc");

  showCaptureGroups("(ab)(abc)+", "ababcabc");

  std::cout << std::endl;

}
```



Capture groups

std::sub_match

The capture groups are of type `std::sub_match`. As with `std::match_results` C++ defines the following four types of synonyms.

```
typedef sub_match<const char*> csub_match;
typedef sub_match<const wchar_t*> wcsub_match;
typedef sub_match<string::const_iterator> ssub_match;
typedef sub_match<wstring::const_iterator> wssub_match;
```

We can further analyze the capture group `cap`.

Method	Description
`cap.matched()`	Indicates if this match was successful.

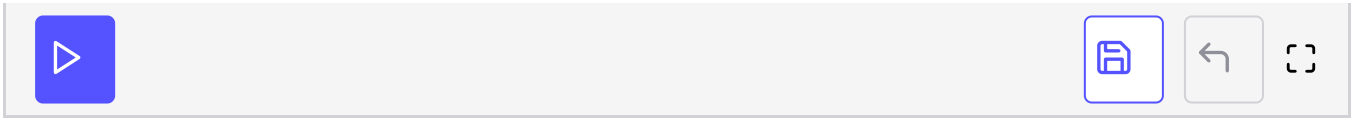
`cap.first()` and `cap.end()`	Returns the begin and end iterator of the character sequence.
`cap.length()`	Returns the length of the capture group.
`cap.str()`	Returns the capture group as string.
`cap.compare(other)`	Compares the current capture group with another capture group.

The std::sub_match object

Here is a code snippet showing the interplay between the search result `std::match_results` and its capture groups `std::sub_match`.

```
#include <iostream>
#include <string>
#include <regex>

int main()
{
    // Simple regular expression matching
    std::string fnames[] = {"foo.txt", "bar.txt", "baz.dat", "zoidberg"};
    std::regex txt_regex("[a-z]+\\.txt");

    for (const auto &fname : fnames) {
        std::cout << fname << ": " << std::regex_match(fname, txt_regex) << '\n';
    }

    // Extraction of a sub-match
    std::regex base_regex("([a-z]+)\\.txt");
    std::smatch base_match;

    for (const auto &fname : fnames) {
        if (std::regex_match(fname, base_match, base_regex)) {
            // The first sub_match is the whole string; the next
            // sub_match is the first parenthesized expression.
            if (base_match.size() == 2) {
                std::ssub_match base_sub_match = base_match[1];
                std::string base = base_sub_match.str();
                std::cout << fname << " has a base of " << base << '\n';
            }
        }
    }

    // Extraction of several sub-matches
```

```
    std::regex pieces_regex("([a-z]+)\\.([a-z]+)");
    std::smatch pieces_match;

    for (const auto &fname : fnames) {
        if (std::regex_match(fname, pieces_match, pieces_regex)) {
            std::cout << fname << '\n';
            for (size_t i = 0; i < pieces_match.size(); ++i) {
                std::ssub_match sub_match = pieces_match[i];
                std::string piece = sub_match.str();
                std::cout << "  submatch " << i << ": " << piece << '\n';
            }
        }
    }
}
```



std::sub_match

In the next lesson, we will look at one of the functions which allows us to send data to `match_results` which we discussed in this lesson.