Reguläre Ausdrücke in C++

Beispiel Code:

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

using namespace std;

int main()
{
   regex reg("mehr (.*) (.*)\.\nSo habe er Pech.", regex_constants::icase |
regex_constants::extended);
   smatch m;
   string str = "Er habe geflucht.\nSo wird ihm der Teufel gewehrt sein.\nUnd keiner ihn
mehr retten kann.\nSo habe er Pech. So habe er die Nacht zu sein gemacht.So habe er
bereut die Tat, dann sei auch der Tag s[...]"; // from mk16.de - Die Strafe der Nacht
   bool res = regex_search(str, m, reg);
   if(res and m.size() > 1)
   {
      for(int i = 0; i < m.size(); i++)
         cout << "x=" << i << ": " << m.str(i) << endl;
      cout << "Ausgabe 1: " << m.str(1) << " " << m.str(2) << endl;
      cout << "Ausgabe 2: " << "Ich " << m.str(2) << " ihn " << m.str(1) << "." << endl;
   }
   return 0;
}
```

Um die Regexp-Funktionen in C++ nutzen zu können müssen wir <regex> importieren(einbetten).

Als nächstes legen wir den Regulären Ausdruck fest mit „regex reg(„Regulärer Ausdruck“, regex_constants);“. „regex“ ist der Typ, „reg“ ist der Name, „„Regulärer Ausdruck““ ist unser Regex, regex_constants sind Angaben(Multiline) wie mit dem Regex verfahrt werden soll.

regex_constants::icase	Groß- und Kleinschreibung ignorieren
regex_constants::extended	Multiline

Weitere Angaben findet man auf(10.12.2017):

http://www.cplusplus.com/reference/regex/regex_constants/

http://en.cppreference.com/w/cpp/regex/syntax_option_type

Als nächstes legen wir „smatch m“ an. „smatch“ ist der Typ, „m“ ist der Name.

Es gibt auch noch „cmatch“. „cmatch“ verwenden wir wenn wir es als Rohe-Zeichenkette angeben.

smatch	smatch m;
	string str = "Er habe geflucht.\nSo wird ihm der Teufel gewehrt sein.\nUnd keiner ihn mehr retten kann.\nSo habe er Pech. So habe er die Nacht zu sein gemacht.So habe er bereut die Tat, dann sei auch der Tag s[...]"; // from mk16.de - Die Strafe der Nacht
	bool res = regex_search(str, m, reg);

cmatch	cmatch m; bool res = regex_search("Er habe geflucht.\nSo wird ihm der Teufel gewehrt sein.\nUnd keiner ihn mehr retten kann.\nSo habe er Pech. So habe er die Nacht zu sein gemacht.So habe er bereut die Tat, dann sei auch der Tag s[...]", m, reg);

Als nächstes geben wir den Befehl die Regex-Auswertung zu starten:

```
bool res = regex_search(str, m, reg);
```

„bool“(Boolean) ist der Typ, „res“(Result) ist der Name, „regex_search“ ist die Funktion für die Auswertung der Regex, „str“(String) ist der Text der „durchsucht“ wird, „m“(match) ist die „match“ in der die Ergebnisse gespeichert werden, „reg“(RegEx) ist der Regex.

Als nächstes überprüfen wir ob die Auswertung erfolgreich war. Wenn ja dann ist res=True sonst ist res=False. Mit „ m.size() > 1“ überprüfen wir ob das Match mehr als ein Ergebnis hat.
Danach geben wir es in einer for-Schleife(for-loop) aus. Danach bilden wir Sätze mit den Ergebnissen.

Die ganze Ausgabe des Codes ist:

```
x=0: mehr retten kann.
So habe er Pech.
x=1: retten
x=2: kann
Ausgabe 1: retten kann
Ausgabe 2: Ich kann ihn retten.
```