Modern C++

<regex>, <ratio>, <chrono>

by Mario Völker

Lecturer: Dr. Ahrens

Content

- `<regex>` for *regular expressions*
- `<ratio>` for a typesafe way of converting values
- `<chrono>` as an implementation of `<ratio>` for time values

Regular Expressions

short Introduction

- Way to specify and recognize patterns in a text
- Used to search and manipulate text based patterns
- Patterns based on *grouping, OR, Quantification*
- To be found *grep, Perl, Ruby, AWK .....*

Regular Expressions
Implementation

- Excerpt of ECMAScript-Regular Expression Syntax (**not** the POSIX standard)

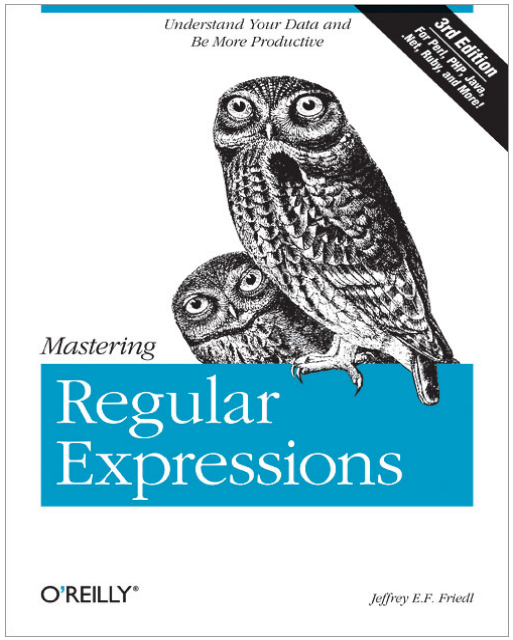
ab	a followed by b	a{n.m}	n to m times a
a|b	Alternative	( )	Grouping
.	Any character	^$	Stringbeginning and -end
a?	0 or 1 a	[class]	Character out of class
a*	0 or more times a	\d \w \s	class-shorts
a+	1 or more times a	\b	word boundary

Pattern	Text	ECMAScript
a|ab	"xaby"	"a"
.*(a|xayy)	"zzxayyzz"	"zzxa"
.*([[:alnum:]]+).*	" abc def xyz "	m[1] = "z"
<[^>]*>	"A link"	m[1] = "" m[2] = ""

Further information on http://v2.cplusplus.com/reference/std/regex/ECMAScript/

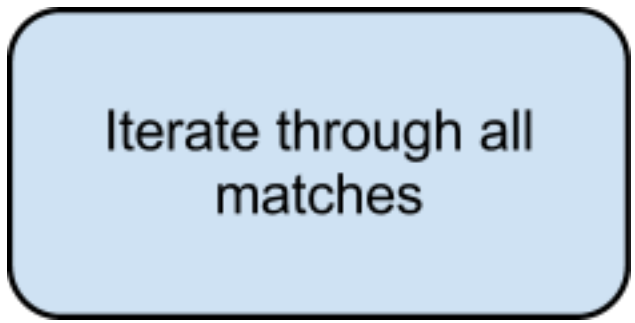
Regular Expressions in C++

- `std::regex`
- std::match_results
- `std::regex_search`
- `std::regex_replace`
- **"Mastering Regular Expressions",** Jeffrey E.F. Friedl
- **Examples**

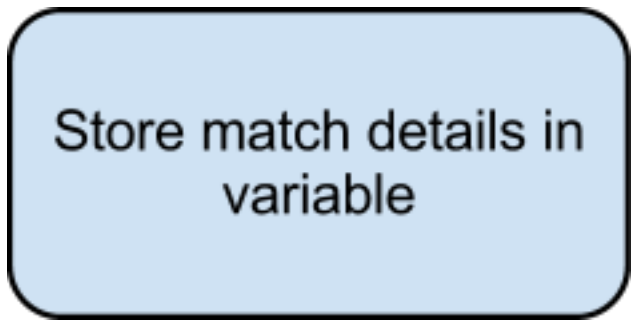


```
#include <regex>

static const regex rgxMobile("(01[567][[:digit:]]{6,10})");


bool isMobilephone(const string& text) {
  return regex_match(text, rgxMobile);
}

bool containsMobilephone(const string &text) {
  return regex_search(text, rgxMobile);
}




void replace() {
  string text = "Title;Album;Artist";
  regex pattern(";");
  string new = regex_replace(text, pattern, string(","));
}
```

```
...

void listMobilephones(const string& text) {

  sregex_iterator begin( text.cbegin(), text.cend(),
                                        rgxMobile );
  sregex_iterator end;

  for(auto it = begin; it != end; it++) {
    cout << it-> str() << " ";
  }
}
```



```
...

void cmatchRes() {
    cmatch res;
    string text = "<h2>Ergebnis und Teil davon</h2>";
    regex muster("<h(.)>([^<]+)");

    regex_search(text, res, muster);
    cout << res[1] << ". " // H-Ebene
         << res[2] << endl; // H-Text
}
```



Compile-Time Arithmetic support
<ratio>

- `template<`
  `std::intmax_t Num,`
  `std::intmax_t Denom = 1`
  `> class ratio;`

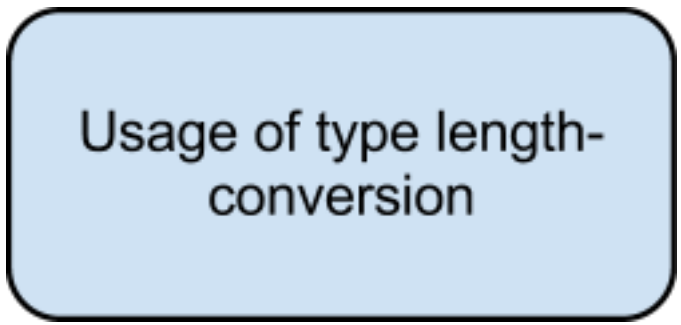
- `std::ratio`
  - `milli std:ratio <1,1000>`
  - `deci std:ratio <1,10>`
  - `kilo std:ratio <1000,1>`

- **Examples**

Implement typesafe conversions for lengths

```
#include <ratio>

..
struct Length {
  long long val_;
  Length(long long val) : val_(val) {}
  Length() = default;
  Length(const Length&) = default;
  Length& operator=(const Length&) = default;

  //converting
  template<typename Scale2>
  Length(const Length<Scale2> & other)
  : val_( other.val_*
    (Scale2::num*Scale::den)/(Scale2::den*Scale::num) ) {}
  // access
  long long value() const { return val_; }
};
```

```
...

typedef Length<ratio<1>> m;
typedef Length<kilo> km;
typedef Length<milli> mm;

int main() {
  km len_km = 300;
  mm len_mm = len_km;
  cout << " millimeter:" << len_mm.value() << endl;
}
```



Time Intervals
<chrono>

- `std::chrono`
- Save way of manipulating specific timespans and moments
- `system_clock` - "real" precision (vulnerable to "timejumps")
- `steady_clock` - constant period
- `high_resolution_clock` - highest precision
- **Examples**

```
#include <chrono>
#include <iostream>

using namespace std;
using namespace chrono;

int main() {
  seconds sec1 = hours(2)
               + minutes(35)
               + seconds(9);
  cout << sec1.count() << endl; // 9309
}
```

typesafe time-conversion via <chrono>

```
..
void f()
{
   this_thread::sleep_for(milliseconds(200));
}

int main()
{
 auto t1 = high_resolution_clock::now();
 f();
 auto t2 = high_resolution_clock::now();

 cout << "f() took "
  << duration_cast<chrono::milliseconds> (t2-t1).count()
  << " milliseconds\n";
}
```

give and measure timespans via <chrono> and clocks

Questions?