std::regex

https://wg21.link/n4606
http://en.cppreference.com/w/cpp/regex/basic_regex
http://en.cppreference.com/w/cpp/regex/regex_match

Patterns for matching against inputs

```
/hello/ matches "hello"
/./ matches any single character
/a*/ matches zero or more consecutive "a"s
/b+/ matches one or more consecutive "b"s
/[a-z0-9]/ matches any of a range of characters
/(ab|cd)/ matches either "ab" or "cd"
```


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

Shortcuts supported by C++ regex

C++'s default regex syntax is "Modified ECMAScript."

```
These regexes are all equivalent:
  /[0-9]/   /[[:digit:]]/   /\d/
```

```
\d digit        [[:digit:]]
\s whitespace   [[:space:]]
\w identifier   [[:alnum:]_]
```


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

Shortcuts supported by C++ regex

C++'s default regex syntax is "Modified ECMAScript."

These regexes are all equivalent:

```
/[0-9]/   /[[:digit:]]/   /\d/
```

		Capitalize to invert:
\d digit	[[:digit:]]	*non*digit \D
\s whitespace	[[:space:]]	*non*space \S
\w identifier	[[:alnum:]_]	*non*ident \W

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

Putting it all together

`/\d\d\d\d-\d\d-\d\d/` matches `"2017-01-10"`

Parentheses do double duty; they also help extract pieces of the matched string.

`/(\d\d\d\d)-(\d\d)-(\d\d)/` matches `"2017-01-10"`

and has three *capturing groups*

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

Using regexes in Perl

```
sub ymd_from_date {
    my ($s) = @_;
    if ($s =~ /(\d{4})-(\d\d)-(\d\d)/) {
        return ($1, $2, $3);
    }
    die;  # or whatever
}
```

Using regexes in Python

```
import re

def ymd_from_date(s):
    m = re.match(r'(\d{4})-(\d\d)-(\d\d)', s)
    if m:
        return (
            int(m.group(1)),  # year
            int(m.group(2)),  # month
            int(m.group(3)),  # day
        )
    assert False  # or whatever
```


https://docs.python.org/2/library/re.html

Using regexes in C++

```
auto ymd_from_date(const std::string& s) -> std::array<int, 3> {
    std::regex rx("(\\d{4})-(\\d\\d)-(\\d\\d)");
    std::smatch m;
    if (std::regex_match(s, m, rx)) {
        return {
            std::stoi(m.str(1)),  // year
            std::stoi(m.str(2)),  // month
            std::stoi(m.str(3)),  // day
        };
    }
    assert(false);  // or whatever
}
```

Raw string literals

```
auto ymd_from_date(const std::string& s) -> std::array<int, 3> {
    std::regex rx(R"((\d{4})-(\d\d)-(\d\d))");
    std::smatch m;
    if (std::regex_match(s, m, rx)) {
        return {
            std::stoi(m.str(1)),  // year
            std::stoi(m.str(2)),  // month
            std::stoi(m.str(3)),  // day
        };
    }
    assert(false);  // or whatever
}
```


http://en.cppreference.com/w/cpp/language/string_literal

Raw string literals (C++11)

The syntax for raw string literals in Python is

```
r'raw text'  # or
"""raw text"""
```

The syntax for raw string literals in C++ is

```
R"delimiter(raw text)delimiter"
```

delimiter may be empty, or it may be a string of up to 16 characters, as long as it doesn't contain '(', '\', or spaces.
Unfortunately the parens are required.
Notice that

```
R"""(raw text)"""
```

is a valid raw string literal!

http://en.cppreference.com/w/cpp/language/string_literal

User-defined literals (C++11)

In Python, because you can use a plain old string as a regex, and because `r'foo'` is a plain old string, it's common to see `r'foo'` used as a "regex literal".

In C++, you cannot use a plain old string as a regex (the relevant constructor of `regex` is marked `explicit`). And there's no UDL for `regex` even in C++17. But you can easily make a UDL for your own codebase if you really want to:

```
auto operator ""_rx(const char *str, size_t n) {
    return std::regex(str, n);
}
```


http://en.cppreference.com/w/cpp/language/user_literal

Escape special chars with backslash

The following special chars must be backslash-escaped in order to match themselves:

```
\ . *+? ^$ ()[]{} |
```

Remember that unless you're using raw string literals, every `'\'` must be escaped! So `"\\\\"` matches a literal backslash.

```
std::cmatch m;
assert(std::regex_match("$10", m, "\\$\\d+"_rx));
```

Most characters lose their special meanings inside `[]`, too.

```
assert(std::regex_match("$10+", m, "[$]\\d+\\+$"_rx));
assert(std::regex_match("hi?[\\]", m, "^hi[?][[].\\]$"_rx));
```


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

Sample AoC skeleton

```
#include <stdio.h>
#include <iostream>
#include <regex>
#include <string>

int process_line(const std::string& s) {
    std::smatch m;
    std::regex_match(s, m, std::regex("([0-9]*)[+]([0-9a-f]*)"));
    return std::stoi(m.str(1)) + std::stoi(m.str(2), nullptr, 16);
}

int main() {
    std::string line;
    while (std::getline(std::cin, line)) {
        int sum = process_line(line);
        printf("%d\n", sum);
    }
}
```


goo.gl/KDBl2t

Sample AoC skeleton

Sample input:

```
1+2
10+10
123+abc
```

```
#include <stdio.h>
#include <iostream>
#include <regex>
#include <string>

int process_line(const std::string& s) {
    std::smatch m;
    std::regex_match(s, m, std::regex("([0-9]*)[+]([0-9a-f]*)"));
    return std::stoi(m.str(1)) + std::stoi(m.str(2), nullptr, 16);
}

int main() {
    std::string line;
    while (std::getline(std::cin, line)) {
        int sum = process_line(line);
        printf("%d\n", sum);
    }
}
```


goo.gl/KDBl2t