

Les expressions régulières

<http://www.informit.com/articles/article.aspx?p=2079020>

Création et initialisation

Fichier d'en-tête : `#include <regex>`

Initialisation : `std::regex pattern { "bla bla bla" }; Utilisation des raw string`

Fonctionnalités principales

Sera détaillé par la suite.

- `match` : vérifier qu'une chaîne donnée correspond à un motif
- `recherche` : trouve une sous-chaîne correspondant à un motif dans une chaîne
- `remplacement` : recherche une sous-chaîne et la remplace

Parcourir : `regex_iterator` et `regex_token_iterator`. Sera vu plus tard ?

Introduction au langage des regex

Définitions :

une expression régulière est un motif représenté par une chaîne

- séquence cible (*Target sequence*) : chaîne sur laquelle est appliqué la recherche
- motif (*pattern*) : séquence de caractères représentant ce que vous recherchez

- correspondance (*match*) : sous-chaîne de la séquence cible qui correspond au motif

Syntaxe

Plusieurs syntaxes possible, par défaut ECMAScript. Autre : basic posix, extended posix, awk, grep, egrep

```
std::regex("meow", std::regex::ECMAScript|std::regex::icase)
```

Vérifier qu'une chaîne correspond à un motif. Par exemple, vérifier un code postal, une date, un nombre, etc.

```
std::regex pattern { R"(\d{5})" }; -> 5 chiffres
```

Caractères spéciaux : `^ $ \ . * + ? ( ) [ ] { } |` ont un sens spécifique. Pour utiliser le caractère "normal" correspondant à un caractère spécial, précéder de `\ : \^ \$`

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

Caractères génériques :

- `.` 1 caractère quelconque
- `*` 0 ou plus
- `+` 1 ou plus
- `?` optionnel (0 ou 1)

Ancres début et fin :

- `^` début : `"^abc"` = doit commencer par "abc"
- `$` fin : `"abc$"` = doit terminer par "abc"

Comptage :

- `{n}` exactement n fois
- `{n,}` au moins n fois
- `{n,m}` entre n et m fois

Groupe et classes [] ()

Classes de caractères (et abréviation)

main.cpp

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

void search(std::string const& target, std::string const&
pattern_str) {
    std::regex pattern { pattern_str };
    std::cout << R"(search ")" << pattern_str << R"(" in ")"
<< target << R"(" = )"
    << std::boolalpha << std::regex_search(target,
pattern) << std::endl;
}

void match(std::string const& target, std::string const&
pattern_str) {
    std::regex pattern { pattern_str };
    std::cout << R"(")" << pattern_str << R"(" match with ")"
<< target << R"(" = )"
    << std::boolalpha << std::regex_match(target,
pattern) << std::endl;
}

int main()
{
    std::cout << "Caractères de base" << std::endl;
    match("", R"(a)");
    match("a", R"(a)");
    match("b", R"(a)");
    match("ab", R"(a)");
    match("ab", R"(ab)");
    match("abb", R"(ab)");
    std::cout << std::endl;

    std::cout << "différence entre match et search" << std::
endl;
    match("ab", R"(a)");
}
```

```

search("ab", R"(a)");
std::cout << std::endl;

std::cout << "Caractères générique (wildcard)" << std::
endl;
match("", R"(.)");
match("a", R"(.)");
match("abc", R"(.)");
std::cout << std::endl;

match("abc", R"(abc)");
match("a5c", R"(abc)");
match("abc", R"(a.c)");
match("a5c", R"(a.c)");
std::cout << std::endl;

match("a", R"([[:alpha:]])");
match("1", R"([[:alpha:]])");
match("&", R"([[:alpha:]])");
match("a", R"([[:alnum:]])"); // alnum = alphanumeric
match("1", R"([[:alnum:]])");
match("&", R"([[:alnum:]])");
std::cout << "Autres : alnum ou w, alpha, blank, cntrl,
digit ou d, graph, lower,\n"
"      print, punct, space ou s, upper, xdigit (digit
hexa)" << std::endl;
std::cout << std::endl;

std::cout << "Ensemble de caractères (character set)" <<
std::endl;
match("", R"([ab])");
match("a", R"([ab])");
match("b", R"([ab])");
match("ab", R"([ab])");
std::cout << std::endl;

match("", R"([a-e])"); // range specification
match("a", R"([a-e])");
match("e", R"([a-e])");
match("f", R"([a-e])");
match("ab", R"([a-e])");
std::cout << std::endl;

```

```

match("", R"(ab[c-f])");
match("abc", R"(ab[c-f])");
std::cout << std::endl;

// escape notation
// \d = [[:d:]]
// \D = [^[:d:]]
// \s = [[:s:]]
// \S = [^[:s:]]
// \w = [[:w:]]
// \W = [^[:w:]]

std::cout << "Répétition" << std::endl;
match("", R"(a*)"); // zero or more
match("a", R"(a*)");
match("aa", R"(a*)");
match("aaaaaa", R"(a*)");
match("b", R"(a*)");
search("jdsihbhvds", R"(a*)");
std::cout << std::endl;

match("", R"(a+)"); // one or more
match("a", R"(a+)");
match("aa", R"(a+)");
match("aaaaaa", R"(a+)");
match("b", R"(a+)");
std::cout << std::endl;

match("", R"(a?)"); // zero or one
match("a", R"(a?)");
match("aa", R"(a?)");
std::cout << std::endl;

match("", R"(a{3})"); // bounded repeat
match("a", R"(a{3})");
match("aaa", R"(a{3})");
match("aaaaa", R"(a{3})");
match("", R"(a{3,})");
match("a", R"(a{3,})");
match("aaa", R"(a{3,})");
match("aaaaa", R"(a{3,})");

```

```

match("", R"(a{3,5})");
match("a", R"(a{3,5})");
match("aaa", R"(a{3,5})");
match("aaaaa", R"(a{3,5})");
match("aaaaaaa", R"(a{3,5})");
std::cout << std::endl;

// non-greedy (non vorace)
search("aaaaa", R"(a{3})"); // -> aaaaa = plus
longue correspondance
search("aaaaa", R"(a{3}?)"); // -> aaa = plus courte
correspondance
std::cout << std::endl;

std::cout << "Ancres (anchor)" << std::endl;
search("", R"(^a)"); // début
search("a", R"(^a)");
search("abc", R"(^a)");
search("cba", R"(^a)");
std::cout << std::endl;

search("", R"(a$)"); // fin
search("a", R"(a$)");
search("abc", R"(a$)");
search("cba", R"(a$)");
std::cout << std::endl;

std::cout << "Groupes (group)" << std::endl;
std::cout << "Priorité des opérateurs (precedence)" <<
std::endl;
}

```

affiche :

Caractères de base

```

"a" match with "" = false
"a" match with "a" = true
"a" match with "b" = false
"a" match with "ab" = false
"ab" match with "ab" = true
"ab" match with "abb" = false

```

différence entre match et search

"a" match with "ab" = false

search "a" in "ab" = true

Caractères générique (wildcard)

"." match with "" = false

"." match with "a" = true

"." match with "abc" = false

"abc" match with "abc" = true

"abc" match with "a5c" = false

"a.c" match with "abc" = true

"a.c" match with "a5c" = true

"[:alpha:]" match with "a" = true

"[:alpha:]" match with "1" = false

"[:alpha:]" match with "&" = false

"[:alnum:]" match with "a" = true

"[:alnum:]" match with "1" = true

"[:alnum:]" match with "&" = false

Autres : alnum ou w, alpha, blank, cntrl, digit ou d, graph, lower,

print, punct, space ou s, upper, xdigit (digit hexa)

Ensemble de caractères (character set)

"[ab]" match with "" = false

"[ab]" match with "a" = true

"[ab]" match with "b" = true

"[ab]" match with "ab" = false

"[a-e]" match with "" = false

"[a-e]" match with "a" = true

"[a-e]" match with "e" = true

"[a-e]" match with "f" = false

"[a-e]" match with "ab" = false

"ab[c-f]" match with "" = false

"ab[c-f]" match with "abc" = true

Répétition

"a*" match with "" = true

"a*" match with "a" = true

```
"a*" match with "aa" = true
"a*" match with "aaaaaa" = true
"a*" match with "b" = false
search "a*" in "jdsihbhvds" = true
```

```
"a+" match with "" = false
"a+" match with "a" = true
"a+" match with "aa" = true
"a+" match with "aaaaaa" = true
"a+" match with "b" = false
```

```
"a?" match with "" = true
"a?" match with "a" = true
"a?" match with "aa" = false
```

```
"a{3}" match with "" = false
"a{3}" match with "a" = false
"a{3}" match with "aaa" = true
"a{3}" match with "aaaaa" = false
"a{3,}" match with "" = false
"a{3,}" match with "a" = false
"a{3,}" match with "aaa" = true
"a{3,}" match with "aaaaa" = true
"a{3,5}" match with "" = false
"a{3,5}" match with "a" = false
"a{3,5}" match with "aaa" = true
"a{3,5}" match with "aaaaa" = true
"a{3,5}" match with "aaaaaaa" = false
```

```
search "a{3}" in "aaaaa" = true
search "a{3}?" in "aaaaa" = true
```

Ancres (anchor)

```
search "^a" in "" = false
search "^a" in "a" = true
search "^a" in "abc" = true
search "^a" in "cba" = false
```

```
search "a$" in "" = false
search "a$" in "a" = true
search "a$" in "abc" = false
search "a$" in "cba" = true
```

Groupes (group)
Priorité des opérateurs (precedence)

<http://stormimon.developpez.com/dotnet/expressions-regulieres/>

Chapitre précédent	Sommaire principal	Chapitre suivant
---------------------------	---	-------------------------