

## Les expressions régulières 3

### Valider qu'une chaîne correspond à un motif

Pour terminer ce chapitre sur la comparaison de chaînes, voyons l'utilisation des motifs pour valider une chaîne. La fonction correspondante, `std::regex_match`, a déjà été utilisée dans les chapitres sur les expressions régulières pour présenter leur syntaxe. Pour rappel, la syntaxe de base prend simplement en arguments la séquence cible et le motif et retourne vrai si la séquence correspond au motif.

```
std::regex pattern { "abc" };
std::string target { "abcdef" };
bool result = std::regex_match(target, pattern);
```

Il est également possible d'utiliser cette fonction en passant en arguments les premier et dernier éléments de la séquence cible, comme pour les algorithmes.

```
std::regex_match(begin(target), end(target), pattern);
```

match flags ? [http://en.cppreference.com/w/cpp/regex/match\\_flag\\_type](http://en.cppreference.com/w/cpp/regex/match_flag_type)

exemples : <https://support.google.com/a/answer/1371417?hl=fr>

main.cpp

```
{
    std::regex pattern("(ab)cd(ef)");    // Find double
word.
    std::string replacement = "le premier groupe est $1
et le second groupe est $2";
    std::string target = "abcdef";
```

```

        std::string output_str = regex_replace(target,
pattern, replacement);
        std::cout << output_str << std::endl;
    }
    {
        std::regex pattern(R
"((\\d{2})[-/](\\d{2})[-/](\\d{4}))");
        std::smatch match;
        std::regex_search(std::string("12-03-2014"), match,
pattern);
        for (size_t i = 0; i < match.size(); ++i)
        {
            std::cout << i << ": " << match[i].str() << '\n';
        }
    }

    std::cout << "Traduction" << std::endl;
    {
        std::regex pattern("([a-zA-Z]+) \\s1");
        std::string replacement = "$1";
        std::string target = "The cat cat bites the dog
dog.";
        std::string output_str = regex_replace(target,
pattern, replacement);
        std::cout << output_str << std::endl;
    }

    std::cout << tr("bla bla $1 bla bla", 123) << std::endl;
    std::cout << tr("bli bli bli bli $1", 123) << std::endl;
    std::cout << std::endl;

    std::cout << "Groupe" << std::endl;
    match("", R"((ab)*)");
    match("a", R"((ab)*)");
    match("b", R"((ab)*)");
    match("ab", R"((ab)*)");
    match("abc", R"((ab)*)");
    match("ababab", R"((ab)*)");
    std::cout << std::endl;

```

```

match("cat", R"(c[a-z]*t)");
std::cout << std::endl;

std::cout << "Exemples de regex" << std::endl;
std::cout << "Date" << std::endl;
match("12-03-2014", R"(\d{2}[-/]\d{2}[-/]\d{4})");
std::cout << "Time" << std::endl;
match("15:17", R"(\d{2}:\d{2})");
std::cout << std::endl;

// vérifier qu'un identifiant C++ est valide
// [a-zA-Z][a-zA-Z_0-9]*

// fichier windows
//[a-zA-Z][a-zA-Z_0-9]*\.[a-zA-Z0-9]+

std::string regex_str = "[a-z_][a-z_0-9]*\\. [a-z0-9]+";
std::regex regl(regex_str, std::regex_constants::icase);
std::string str = "File names are readme.txt and
my.cmd.";
std::sregex_iterator it(str.begin(), str.end(), regl);
std::sregex_iterator it_end;
while(it != it_end) {
    std::cout << it->str() << std::endl;
    ++it;
}
}

```

#### Groups

le premier groupe est ab et le second groupe est ef

0: 12-03-2014

1: 12

2: 03

3: 2014

Traduction

The cat bites the dog.

bla bla 123 bla bla

bli bli bli bli 123

#### Groupe

"(ab)\*" match with "" = true

"(ab)\*" match with "a" = false

```
"(ab)*" match with "b" = false
"(ab)*" match with "ab" = true
"(ab)*" match with "abc" = false
"(ab)*" match with "ababab" = true
```

```
"c[a-z]*t" match with "cat" = true
```

Exemples de regex

Date

```
"\d{2}[/-]\d{2}[/-]\d{4}" match with "12-03-2014" = true
```

Time

```
"\d{2}:\d{2}" match with "15:17" = true
```

readme.txt

my.cmd

## Manque : vorace

A déplacer dans le chapitre sur les recherches ?

main.cpp

```
// 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;
}
```

affiche :

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

|                                    |                                    |                                  |
|------------------------------------|------------------------------------|----------------------------------|
| <a href="#">Chapitre précédent</a> | <a href="#">Sommaire principal</a> | <a href="#">Chapitre suivant</a> |
| <a href="#">Cours, C++</a>         |                                    |                                  |