Les expressions régulières 3

Les groupes de capture

Les groupes de capture permettent de réutiliser une sous-chaîne de la séquence cible, identifiée par un motif. La sous-chaîne capturée peut être réutilisée dans le motif ou être récupérée dans le code C++. Pour créer un groupe de capture, vous devez simplement écrire le motif correspondant à ce groupe entre parenthèses. Pour que cela soit plus clair, voyons quelques exemples.

Répéter un groupe de capture

Commençons par un exemple simple de groupe, sans utiliser la capture (donc sans réutiliser la sous-chaîne identifiée par le groupe). Le but est d'écrire un motif qui permet de valider une chaîne constituée de répétition la chaîne `"ab"`, comme par exemple `"ab"` ou `"ababab"`. En première intention, on pourrait être tenté d'écrire le motif suivant : `"ab*"`.

Cependant, le caractère de répétition `*` s'applique que sur le caractère `b`. Pour indiquer que l'on souhaite répéter le motif `"ab"`, il faut donc le mettre entre parenthèses : `"(ab)*"`.

main.cpp

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

int main()
{
    std::regex pattern { "ab*" };
    std::cout << "'ab*' match with '': " << std::boolalpha
<<
```

```
        std::regex_match("", pattern) << std::endl;

    std::cout << "'ab*' match with 'ab': " << std::boolalpha
<<
        std::regex_match("ab", pattern) << std::endl;

    std::cout << "'ab*' match with 'abab': " << std::
boolalpha <<
        std::regex_match("abab", pattern) << std::endl <<
std::endl;

    pattern = "(ab)*";
    std::cout << "'(ab)*' match with '': " << std::boolalpha
<<
        std::regex_match("", pattern) << std::endl;

    std::cout << "'(ab)*' match with 'ab': " << std::
boolalpha <<
        std::regex_match("ab", pattern) << std::endl;

    std::cout << "'(ab)*' match with 'abab': " << std::
boolalpha <<
        std::regex_match("abab", pattern) << std::endl;
}
```

affiche :

```
'ab*' match with '': false
'ab*' match with 'ab': true
'ab*' match with 'abab': false

'(ab)*' match with '': true
'(ab)*' match with 'ab': true
'(ab)*' match with 'abab': true
```

Réutiliser un groupe de capture

Les chaînes identifiées par un groupe de capture peut être réutilisées dans le motif. Chaque groupe est identifié par un numéro, dans l'ordre de

leur déclaration. Pour réutiliser une chaîne, il faut indiquer le numéro de groupe précédé d'une barre oblique inversée : `\1`, `\2`, `\3`, etc.

Par exemple, pour écrire un motif qui permet d'identifier n'importe quelle chaîne commençant et terminant par le même caractère, on pourra écrire le motif suivant :

- pour récupérer le premier caractère : `(.)` ;
- pour les caractères de la chaîne autre que le premier et le dernier caractère : `.*` ;
- pour le dernier caractère, qui doit être identique au premier : `\1`.

Donc le motif final est `(.).*\1`. Pour écrire ce motif en C++, n'oubliez pas qu'il faut ajouter un caractère d'échappement devant la barre oblique inversée (le motif s'écrit alors `"(.).*\\1"`) ou il faut écrire une littérale chaîne brute (le motif s'écrit alors `R"((.).*\1)"`).

main.cpp

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

int main()
{
    std::regex const pattern { R"((.).*\1)" };
    std::cout << "'(.).*\1' match with '': " << std::
boolalpha <<
        std::regex_match("", pattern) << std::endl;

    std::cout << "'(.).*\1' match with 'abc': " << std::
boolalpha <<
        std::regex_match("abc", pattern) << std::endl;

    std::cout << "'(.).*\1' match with 'aba': " << std::
boolalpha <<
        std::regex_match("aba", pattern) << std::endl;

    std::cout << "'(.).*\1' match with 'abcdefa': " << std::
boolalpha <<
        std::regex_match("abcdefa", pattern) << std::endl;
}
```

affiche :

```
'(.).*□' match with '': false
'(.).*□' match with 'abc': false
'(.).*□' match with 'aba': true
'(.).*□' match with 'abcdefa': true
```

De la même façon, si on veut pouvoir identifier une chaîne qui commence par deux caractères et qui se termine par ces deux caractères dans l'ordre inverse, on va pouvoir utiliser deux groupes de capture. Le motif s'écrit alors : `(.)(.).*\2\1`.

Pour terminer avec les groupes de capture, ils vont permettre de récupérer dans le code C++ une sous-chaîne correspondant à un motif complet de la chaîne. En effet, imaginons que l'on souhaite écrire un motif qui permet de reconnaître une date au format `jj/mm/aa` (jour-mois-année, chaque élément étant écrit avec deux chiffres) et qui permet de récupérer le jour. On pourrait écrire simplement le motif `^[[:digit:]]{2}` (ou `^\d{2}`). Le problème est que ce motif peut correspondre à n'importe quelle chaîne commençant par deux chiffre (pour rappel, le caractère `^` est une ancre indiquant le début de la séquence cible), par exemple `"12abcde"` ou `"123456"`.

Pour éviter cela, on va simplement écrire un motif qui correspond à une date, selon le format indiqué : `\d{2}/\d{2}/\d{2}`. On ajoute ensuite un groupe de capture pour les sous-chaînes que l'on souhaite récupérer dans le code C++. Par exemple, pour récupérer les jours, on écrit : `(\d{2})/\d{2}/\d{2}`. Cela permet de récupérer la date du jour, en garantissant que le format de date est respecté.

Nous verrons par la suite le code C++ utilisé pour récupérer les groupes capturés.

Chapitre précédent	Sommaire principal	Chapitre suivant

Cours, C++