Rechercher et remplacer dans les collections

Pas question de détailler tous les algorithmes dans le cours. Faut les utiliser en pratique et s'habituer à la page de documentation pour avoir une vue d'ensemble.

Rechercher dans une collection

La recherche d'un caractère dans une chaîne peut également être réalisée en utilisant des fonctions membres et des fonctions libres (comme pour les fonctions `begin` et `end` dans le chapitre précédent). C'est le cas de la fonction de base pour faire une recherche : `find` (*trouver* en anglais). D'autres fonctions de recherche ne sont utilisable qu'en fonction membre :

- `rfind` recherche à partir de la fin (*reverse find*) ;
- `find_first_of` recherche la première occurrence d'un caractère ;
- `find_first_not_of` recherche la première abscence de caractère ??? ;
- `find_last_of` recherche la dernière occurrence d'un caractère ??? ;
- `find_last_not_of` recherche la dernière absence d'un caractère ???.

D'autres fonctions ne sont utilisables que sous forme de fonction libre (ce sont les algorithmes de la bibliothèque standard) :

- `count` pour compter le nombre d'occurrence d'un caractère ;

Comme pour les algorithmes de la bibliothèque standard vu dans le chapitre précédent (`std::equal` et `std::lexicographical_compare`), il existe également une version de ces algorithmes précédents utilisant un

prédicat personnalisé. Cependant, ici, les algorithmes change de nom, mais dans tous les cas, le prédicat est le dernier argument donné.

- `find_if`
- `find_if_not`
- `count_if`

Différence entre membre et libre : libre retourne une position (itérateur), membre retourne entier (numéro de position)

position : begin et end représente début et fin. find retourne une position (concept d'itérateur, détiallé ensuite). Utilisation :

```
#include <iostream>
#include <string>
#include <algorithm>

int main() {
    std::string const s { "azertyazerty" };
    auto position = std::find(begin(s), end(s), 'e');
    auto const first = std::string(begin(s), position);
    auto const second = std::string(position, end(s));
    std::cout << first << std::endl;
    std::cout << second << std::endl;

    position = std::find(position+1, end(s), 'e');
    std::cout << std::string(begin(s), position) <<
std::endl;
    std::cout << std::string(position, end(s)) << std::endl;
}
```

affiche :

```
az
ertyazerty
azertyaz
erty
```

second find : recherche à partir du 'e' trouvé

problème de fin de collection. recherche 'e', mais string s'arrête avant le

'e' (on n'a pas “aze” et “rtyazerty”). Raison : position de début (élément inclue) et fin (exclu). Si begin correspond à 'a', alors 'a' est dans la string. Si end correspond à 'e', alors 'e' est exclu.

Pour end(s) : correspond pas à 'y' (sinon, celui ci ne serait pas dans la chaîne), mais à l'élément suivant de 'y' (qui n'existe pas). Cela permet de tester si un itérateur retournée par une algo est valide ou pas. Si position == end(s), alors on est en dehors de la chaine, et pas d'élément trouvé

remarque, pour collection vide ({} ou ””), pas d'élément et begin == end

Autres algos

- recherche de caractère
  - find, find_if, find_if_not
  - all_of, any_of, none_of
  - count, count_if
  - adjacent_find
  - search_n
- rechercher un sous chaine
  - mismatch
  - find_end, find_first_of
  - search
- remplacement
  - replace, replace_if
  - replace_copy, replace_copy_if
- suppression
- copie et déplacement
  - copy, copy_if, copy_n, copy_backward
  - move, move_backward

regex

parsing : extraire une information

parcourir toutes les correspondances

token : split une chaîne en sous chaines

Chapitre précédent	Sommaire principal	Chapitre suivant

Cours, C++