Les chaînes de caractères style C

Le type de chaîne char*

Comme vous l'avez vu dans le chapitre sur les littérales, les littérales chaînes de caractères ne sont pas de type `string`, mais de type `const char*`. Ce type est un héritage du C++ historique et du langage C. Pourquoi ne pas utiliser ce type en C++ ?

Faisons un test simple. Essayons de comparer deux chaînes :

main.cpp
```
#include <iostream>

int main() {
    std::cout << std::boolalpha << ("b" < "a") << std::endl;
}
```

Ce code affiche :

```
main.cpp: In function 'int main()':
main.cpp:4:43: warning: comparison with string literal
results in unspecified behaviour [-Waddress]
     std::cout << std::boolalpha << ("b" < "a") << std::endl;
                                           ^
true
```

Premier problème, le compilateur affiche un message d'avertissement pour prévenir que la comparaison de littérales chaînes produit un comportement indéterminé (astuce : si vous ne comprenez pas bien l'anglais, n'hésitez pas à vous servir d'un traducteur en ligne comme Google Translate ou de faire une recherche sur internet en copiant le message d'erreur). On parle de comportement indéterminé (*Undefined Behavior* ou *UB*) lorsque le comportement n'est pas défini dans la norme C++. Donc ce code pourra fonctionner différemment selon le compilateur,

donner le résultat correct ou un résultat aléatoire, produire une erreur, etc. Dans l'idée d'écrire du code C++ moderne (donc de qualité), on évitera bien sûr d'écrire du code qui produit un comportement indéterminé.

Le second problème est que le résultat est vrai, alors que la lettre b est supérieur (dans l'ordre alphabétique) à la lettre a. Vous pouvez essayer avec n'importe quelle lettre (avec Clang sur Coliru), le résultat sera toujours vrai. L'explication de ce comportement nécessite de comprendre le fonctionnement des pointeurs (ce qui sort du cadre de ce cours débutant), mais le principal est de comprendre que cela ne donne pas le résultat attendu.

Le code similaire avec `string` ne posera pas de problème :

main.cpp

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

int main() {
    std::string const s1 { "b" };
    std::string const s2 { "a" };
    std::cout << std::boolalpha << (s1 < s2) << std::endl;
}
```

affiche :

```
false
```

En effet, la classe `string` implémente l'opérateur `<` en respectant la sémantique habituelle de cet opérateur (alors que le type `char*` utilise la sémantique des pointeurs). L'utilisation de `char*` est donc a éviter en C++ moderne.

Le code précédent fonctionne aussi si l'on déclare qu'une seule variable de type `string` :

main.cpp

```
std::string const s { "b" };
```

```
std::cout << std::boolalpha << (s < "a") << std::endl;
```

Dans ce cas, c'est bien la sémantique de `string` qui est utilisée, pas celle de `char*`. Pourquoi le compilateur utilise correctement l'opérateur `<` de `string` et le le fait pas lorsque l'on écrit `“b” < “a”` ?

Le compilateur procède de la façon suivante :

- Il commence par rechercher s'il existe un opérateur `<` qui correspond aux types utilisés. Donc `string` et `char*` dans un cas et deux `char*` dans l'autre cas. Cet opérateur existe pour les deux `char*` (mais s'applique sur les pointeurs, pas sur le contenu de la chaîne de caractères) et est donc utilisé. Dans les cas de `string` et `char*`, aucun opérateur ne convient.
- S'il n'existe pas d'opérateur, le compilateur essaie de convertir l'un des types pour trouver un opérateur `<` qui existe. Toutes les types ne sont pas convertissables dans n'importe quel autre type. Chaque classes spécifient les conversions qui sont autorisée ou non. Elles spécifient également si la conversion peut être implicite (le compilateur décide tout seul s'il réalise la conversion ou non) ou explicite (l'utilisateur doit écrit spécifiquement la conversion).

Dans le cas qui nous intéresse, il existe une conversion implicite de `char*` en `string`. Cela signifie que lorsque le compilateur rencontre un type `char*`, il est autorisé à le convertir en `string` sans demander l'autorisation à l'utilisateur. Le compilateur résout donc le code `s < “a”` en convertissant la littérale chaîne en `string`, puis appelle l'opérateur `<` qui s'applique sur deux `string`. Le code `s < “a”` est donc interprété par le compilateur de la même façon que le code `s1 < s2`.

Il est également possible d'écrire explicitement la conversion d'un type dans un autre en initialisant avec le type. Par exemple :

```
std::string const s { "b" };
std::cout << std::boolalpha << (s < std::string { "a" }) <<
std::endl;
// ou
```

```
std::cout << std::boolalpha << (std::string { "b" } < "a")
<< std::endl;
```

Vous apprendrez dans la partie sur la programmation orientée objet comment créer des classes autorisant les conversions implicites et explicites.

Chapitre précédent	Sommaire principal	Chapitre suivant

Cours, C++