Les chaînes de caractères en détail

Fondamentalement, l'informatique englobe tout ce qui concerne le traitement de l'information. Vous avez vu dans les chapitres précédents les bases du calcul numérique. Un autre type de données qui souvent manipulé en informatique sont les chaînes de caractères. Les chapitres suivants détaillent leurs manipulations en C++.

Les fonctions membres et non membres

parler plus tôt des fonctions. Et mieux expliquer, parce que là, c'est caca...

partie trop détaillée et pas à sa place

Les fonctions sont des traitements spécifiques que l'on peut appliquer sur les données. Vous avez déjà utilisé de telles fonctions dans les chapitres précédents, comme par exemple les fonctions mathématiques ou la fonction `size` pour connaître la taille d'une chaîne de caractères `string`.

Les fonctions peuvent s'écrire selon deux syntaxes différentes. Par exemple :

```
double const d { 123.456 };
std::cout << std::exp(d) << std::endl; // utilisation de la fonction exp

std::string s{ "hello, world!" };
std::cout << s.size() << std::endl; // utilisation de la fonction size
```

Le premier type de fonction, représenté par la fonction mathématique exponentiel `exp`, s'appelle une fonction non membre. Le service qu'elle rend (calculer l'exponentielle d'un nombre réel) s'applique à la valeur qui est passée en argument entre les parenthèses.

Le résultat du calcul est retourné directement par la fonction. Cela signifie que lors de l'évaluation de la fonction, la fonction est remplacée par le résultat du calcul. Ainsi, le code suivant :

```
double const result { std::exp(1.0) };
std::cout << std::exp(1.0) << std::endl;
```

sont équivalent au code suivant, après évaluation de la fonction `exp` :

```
double const result { 2.71828182845905 };
std::cout << 2.71828182845905 << std::endl;
```

Chaque fonction possède une signature spécifique, qui définit son nom, le nombre d'arguments qu'elle prend et leur type et le type de valeur qu'elle retourne. Une fonction peut également ne pas prendre d'argument ou ne pas retourner de valeur.

Le second type de fonction s'appelle une fonction membre et s'applique à un objet spécifique. Un exemple d'une telle fonction est la fonction `size` de la classe `string` :

```
std::string s{ "hello, world!" };
std::cout << s.size() << std::endl; // utilisation de la
fonction size
```

Ce type de fonction permet d'obtenir des informations sur un objet (c'est le cas de `size`) ou de modifier cet objet. Un par exemple de fonction qui modifie un objet pourrait être la fonction `clear`, qui efface une chaîne de caractères.

```
std::string s{ "hello, world!" };
s.clear(); // on efface s
std::cout << s << std::endl;
```

Ce code n'affiche rien, puisque la variable `s` ne contient plus aucune chaîne après appel de la fonction `clear`.

Une fonction membre peut également prendre aucun, un ou plusieurs arguments et retourner ou non une valeur.

Une fonction membre ne peut pas être appelée comme une fonction non membre. Si vous essayer d'appeler la fonction `size` seule ou en lui passant une chaîne en argument, cela produira une erreur :

```
size(s); // erreur
```

Cependant, dans certain cas, il existe une fonction membre et une fonction non membre qui possède ne même nom. C'est par exemple le cas de la fonction `begin`, qui vous verrez par la suite. Vous pouvez donc écrire :

```
string s { "hello, world!" };
s.begin();
begin(s);
```

Les deux lignes avec `begin` sont équivalentes et font exactement la même chose. Par contre, n'oubliez pas qu'il s'agit bien de deux fonctions différentes (cela sera important lorsque vous écrirez vous même vos propres fonctions si c'est pas important maintenant, faut il en parler maintenant ?).

Dans tous les cas, il ne faut pas hésiter à se référer à la documentation, pour connaître la signature de chaque fonction.

Exercices

Lire la documentation de quelques fonctions, membre et non membre. Ecrire le code correcte pour utiliser ces fonctions, meme si on les connait pas, en fonction de la doc.

Les manipulations de base d'une chaîne

Initialisation et assignation d'une chaîne

Les chaînes de caractères en C++ sont manipulées via la classe `std::string` de la bibliothèque standard. L'initialisation ou la modification d'une chaînes se fait en utilisant l'initialisation avec des crochets et l'opérateur d'affectation `=` :

main.cpp

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

int main() {
    std::string const s1 { "hello, world!" }; //
initialisation avec une littérale
    std::string s2 {};                        //
initialisation par défaut
    s2 = "bonjour tout le monde !";           // affectation

    std::cout << "hello, world!" << std::endl; // afficher
une littérale
    std::cout << s2 << std::endl;              // afficher
une variable
}
```

Un point important concernant les littérales chaînes de caractères : elles ne sont pas de type `string`, mais du type `const char *` hérité du C. L'utilisation de ce type n'est pas recommandé, sauf pour initialiser une chaîne de type `string`. La classe `string` apporte des garanties plus forte que `const char *` (fuite mémoire) et offre beaucoup plus de fonctionnalités (que vous avez voir dans la suite de ce chapitre et dans les chapitres suivants).

Les littérales string

Dans la prochaine norme du C++14, il sera possible de créer directement un littérale chaîne de caractères de type `string`, en ajoutant le suffixe “s” après la chaîne. Cela permettra d'éviter les conversions de `'const char *` en `string` et permettra d'utiliser directement `auto` :

```
string s1 { "hello, world!"s };
auto s2 = "hello, world!"s;
```

Remplir une chaîne avec des caractères

Il est possible d'initialiser une chaîne en la remplissant avec un caractère répété n fois. Dans ce cas, la syntaxe pour initialiser la chaîne est différentes de la syntaxe que vous avez déjà utilisée. Lorsque l'on initialise une variable avec une littérale ou avec une expression, vous avez utilisé la syntaxe avec des crochets :

```
string s { "hello, world!" };
```

Ici, il faut passer des arguments

ou modifier une chaîne existante (en utilisant la fonction `assign`),

size, empty, clear, concaténation

Conversion