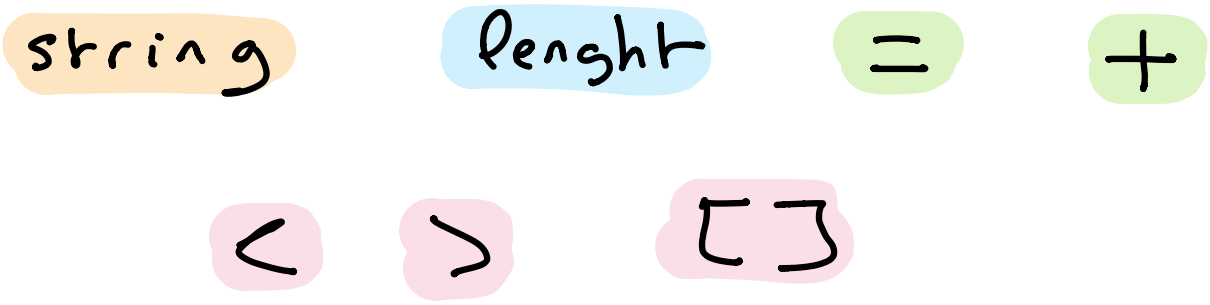
La classe string :

Chaînes en C++



Enseignement de programmation en langages C et C++.

Olivier Baiffeux

Maitre de conférences HDR
à l'université de Bourgogne.

Version 2021.

C et C++ Partie G

Olivier Bailleux, version 2021

Prérequis : Être capable de mettre en application les notions des parties A, B, C, D, E, F et de prévoir le comportement de programmes utilisant ces notions.

Notions abordées : classe standard **`string`** du C++, assignation, copie, concaténation, longueur, fonctions exploitant des instances de **`string`**.

Classe string du C++

La représentation C des chaîne est très rudimentaire et demande beaucoup de précautions pour éviter les bugs. Le C++ dispose d'une classe standard **string** plus simple d'utilisation et plus sécurisée.

```
void presentation()
{
    std::string s = "abcdef";

    char c = s[3];

    std::cout << c << " " << (int)c << " " << std::hex << (int)c << std::endl;

    std::cout << s << std::endl;

    s[3] = 0;

    std::cout << s << std::endl;
}
```

A gauche du = on a une variable de type **std::string**. **std** représente un espace de nom. On peut éviter de le répéter à chaque fois en déclarant **using namspace std;** en début du fichier source.
A droite du = on a une chaîne en format C. L'opérateur d'initialisation = fait la conversion du format C en format C++.

On voit ici comment afficher une valeur de type **char** sous différents formats en C++.

d 100 64

Pour afficher une instance de **string** il suffit de l'envoyer directement dans le flux **cout**.

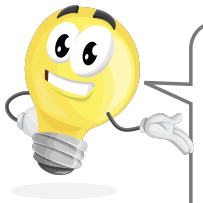
abcdef

abcef

Une instance de **string** n'est PAS un tableau, mais on peut quand même utiliser les crochet [ ] pour lire ou modifier un caractère de la chaîne représentée. On dit que les crochets sont **surchargés** dans la classe **string**.

Dans les chaînes au format C++, la valeur 0 n'est PAS un marqueur de fin de chaîne mais juste un caractère qui ne s'imprime pas.
Nous ne rentrerons pas dans le détail de la représentation interne des chaînes dans la classe string.

Longueur, copie, concaténation de chaînes en C++



Parce-que les chaînes au format C sont représentées par des tableaux, on ne peut les assigner avec = (il faut utiser **strcpy**). Avec la classe **string**, on doit utiliser = et surtout PAS **strcpy**. Ne confondez pas les 2 représentations.

pour comparer des instances de string (ordre lexicographique) on peut utiliser les opérateurs ==, !=, <, >, <=, >=

```
void presentation()
{
    std::string s = "abcdef";

    std::string t = "xxx";

    s = t;

    std::cout << s << std::endl;

    s = s + t;

    std::cout << s << " -> " << s.length() << std::endl;
}
```

L'assignation par = fonctionne avec les instances de **string** : le contenu de la chaîne **t** est recopié dans **s**.

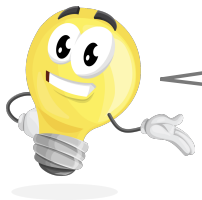
xxx

L'opérateur + permet de réaliser la concaténation de deux chaînes représentées par des instances de **string**.

La méthode **length** retourne la longueur de la chaîne représentée par l'instance courante de **string**.

xxxxxx → 6

Utilisation de la classe string du C++



Ces exercices de découverte vont vous permettre d'assimiler les notions que nous venons de présenter.

XG01

Réalisez une fonction **main** qui :
- crée un tableau de 4 cellules contenant chacune une instance de **string**,
- assigne aux cellules les chaînes "abc", "def", "fgh", et "ijk"
- répète 5 fois (à l'aide d'une boucle **for**) l'échange des contenus de deux cellules dont les indices (entre 0 et 3) sont produits aléatoirement avec la fonction **rand**,
- place dans une variable de type **string** le résultat de la concaténation des chaînes contenues dans les 4 cellules du tableau,
- affiche le contenu de cette variable.

XG02

Réalisez une fonction **main** qui :
- crée une variable de type **string**, et l'initialise avec la chaîne de votre choix,
- parcourt la chaîne, en utilisant une boucle **for**, pour compter le nombre de chiffres (caractères de code ascii compris entre '0' et '9'),
- affiche le nombre de chiffres.

Par exemple, si la chaîne est "abc 12 c34", la valeur affichée doit être 4.

Fonctions exploitant des instances de string

Nous allons examiner plusieurs moyens de réaliser une fontion qui construit le miroir d'une chaîne représentée par une instance de **string**.

Version 1 : On passe une instance de **string** en paramètre par copie, on retourne l'instance de **string** résultat. On aurait pu aussi utiliser un paramètre de type référence sur **string** (**&string**) avec le même code dans le corps de la boucle.

```
std::string miroir1(std::string s)
{
    std::string r = "";
    int n = (int)s.size();
    for(int i=n-1; i >= 0; i--)
    {
        r = r + s[i];
    }
    return r;
}
```

Version 2 : On passe en paramètre une référence sur **string** et on ne retourne rien. La fonction remplace directement la chaîne référencée par son miroir.

```
void miroir2(std::string& s)
{
    int n = (int)s.length();
    for(int i=0; i < n/2; i++)
    {
        char tmp = s[i];
        s[i] = s[n-1-i];
        s[n-1-i] = tmp;
    }
}
```

Version 3 : On passe en paramètre un pointeur sur **string** et on ne retourne rien. La fonction remplace directement la chaîne pointée par son miroir.

```
void miroir3(std::string* s)
{
    int n = (int)(*s).size();
    for(int i=0; i < n/2; i++)
    {
        char tmp = (*s)[i];
        (*s)[i] = (*s)[n-1-i];
        (*s)[n-1-i] = tmp;
    }
}
```

Mise en oeuvre des trois versions.

```
void test()
{
    std::string s = "Ana et Tim";
    s = miroir1(s); std::cout << s << std::endl;
    miroir2(s); std::cout << s << std::endl;
    miroir3(&s); std::cout << s << std::endl;
}
```

miT te anA
Ana et Tim
miT te anA

Qu'avons nous appris ?

- classe **std::string** — représente une chaîne
- Initialisation avec une chaîne au format C — par =
- Assignation d'une variable de type **string** — par =
- Concaténation — avec +
- Accès aux caractères — avec []
- Longueur d'une chaîne — méthode d'instance size
- Passage en paramètre par référence — paramètre de type string&
- Passage en paramètre par copie — paramètre de type string

Exercices d'assimilation

Ces exercices vous permettront de monter en compétence et de vérifier vos acquis. Ne vous précipitez pas. Si possible, testez vos solutions sur machine. Essayez de comprendre vos erreurs. Au besoin demandez des indices ou explications complémentaires.

XG03

Réalisez une fonction **`string pgpc(string& s1, string& s2)`** qui retourne le plus grand préfixe commun aux chaînes désignées par les paramètres **`s1`** et **`s2`**.
Par exemple, si **`s1`** représente la chaîne "abcdef" et **`s2`** représente la chaîne "abcxy", alors le plus grand préfixe commun est "abc".
Indice : on peut construire une chaîne caractère par caractère en partant d'une chaîne vide "" et en utilisant le signe **+** pour lui ajouter des caractères.

Il existe en C++ un type **`bool`** qui représente les Booléens et deux constantes prédéfinies **`true`** et **`false`**. Le type **`bool`** est compatible avec la représentation des Booléens par des entiers utilisée par le langage C. **`true`** est l'entier **1** et **`false`** l'entier **0**.

XG04

Réalisez une fonction **`string mdp(string& caract, int n)`** qui produit et retourne une chaîne aléatoire de **`n`** caractères choisis au hasard dans la chaîne désignée par le paramètre **`caract`**. Il est possible que certains caractères de **`caract`** soient utilisés plusieurs fois et que certains ne soient pas utilisés.
Cette fonction pourrait par exemple être utilsée comme générateur rudimentaire de mots de passe.

XG05

Réalisez une fonction **`bool similar(string& s1, string& s2)`** qui retourne **`true`** si les chaînes désignées par **`s1`** et **`s2`** sont similaires, et **`false`** dans le cas contraire.
Deux chaînes sont considérées comme similaires si et seulement si elle sont de même longeur et si elle sont identiques ou diffèrent par la valeur d'un seul caractère.
Par exemple, "abcd" et similaire à "abvd" mais pas à "acbd" ni à "abc".

Ces exercices vous permettront de monter en compétence et de vérifier vos acquis. Ne vous précipitez pas. Si possible, testez vos solutions sur machine. Essayez de comprendre vos erreurs. Au besoin demandez des indices ou explications complémentaires.

Exercices d'assimilation

XG-06

Pour déterminer si une chaîne représente un nombre fractionnaire écrit en base 10, on peut utiliser un automate fini tel que celui décrit ci-dessous. Au départ l'automate est dans l'état 1. On parcours la chaîne à analyser caractère par caractère. Selon le caractère lu, on peut réaliser une transition vers un autre état, indiquée par une flèche, ou, si une telle transition n'existe pas, arrêter l'analyse (même s'il reste des caractères dans la chaîne). Avec notre automate, la chaîne est reconnue comme valide si et seulement si au moment de l'arrêt, on est dans l'état 2 ou dans l'état 4 (on appelle ces états des états acceptants) et que la fin de la chaîne a été atteinte.
Par exemple si la chaîne est "33.5", la lecture de '3' fait passer en état 2, puis la lecture de '3' reste en état 2, puis '.' fait passer en état 3, puis '5' fait passer en état 4 et l'analyse ses termine parceque la fin de la chaîne est atteinte. Comme l'état final est acceptant, la chaîne est valide.
Mais si la chaine est "33.", à la fin de l'analyse on est dans l'état 3 qui n'est pas acceptant, donc la chaîne n'est pas valide.

Vous devez réaliser une fonction **`bool isFrac(string& str)`** qui retourne le booléen **`true`** si **`str`** est valide au sens décrit ci-dessus et false dans le cas contraire. Le type **`bool`** du C++ représente les constantes **`true`** et **`false`** par les entiers 0 et 1 et est compatible avec la représentations des Booléens par des entiers du langage C.

Cet exercice est plus compliqué que les précédents. Ne le faites que si vous avez un bon niveau en algorithmique et que vous maîtrisez la récursivité.

Exercice de consolidation

XG07

Le but de cet exercice est de trancoder un tableau d'octets en une chaîne de caractères représentant ces octets sous une forme imprimable ou pouvant être transmise ou stockée sous forme d'un texte.
Par exemple, si le tableau d'entrée est le suivant :

```
unsigned char input[] = {0x00, 0x1A, 0xB2}; // Ceci n'est pas une chaîne.
```

alors la chaîne qui représente son contenu sous forme de texte sera "001AB2".

Vous devez réaliser une fonction **`string bin2string(char* input, int n)`** qui transode les **`n`** premiers octets situés dans le tableau désigné par le paramètre **`input`** en une chaîne de longueur 2n comme expliqué plus haut.

XG08

Réalisez une fonction **`string* decompose(string& str, string& sep)`** qui décompose la chaîne désignée par **`str`** en une suite de mots. Tous les caractères de la chaîne **`sep`** sont considérés comme des séparateurs de mots. Les mots résultants de la décomposition doivent être placés sous forme d'instances de **`string`** dans un tableau dynamique retourné par la fonction.
Par exemple si la chaîne à analyser est `"Tim, Tom et Ploum."` et que la chaîne des séparateur est `". ,"`, la fonction doit retourner un tableau dynamique de trois cellules contenant des instances de **`string`** représentant les chaînes `"Tim"`, `"Tom"` et `"Ploum"`.