

COURS DE C++

7 - Première classe, la classe string

CHAINES DE CARACTÈRES A LA MODE C++

- En C++ il existe une autre manière de faire des chaînes de caractères : la classe string

```
#include <string>
```

Pour définir une valeur de ce type : `string s;`

Pour l'initialiser : `s = string("hello");`

Les deux en même temps : `string s("hello");`

CONSTRUCTEURS

La syntaxe `string("hello");` s'appelle une construction

A chaque construction correspond une fonction constructeur, ici `string(const char *str);` qui prend en entrée une chaîne à la C et produit une string ayant le même contenu [Attention il n'y jamais de retour pour un constructeur]

- Il existe plusieurs constructeurs :

```
string()
```

```
string(size_type length, const char &ch)
```

```
string(const char *str, size_type length)
```

```
string(const string &str)
```

- ...

Quand on définit `string x;` on appelle en fait le constructeur qui ne prend pas d'arguments (appelé constructeur par défaut).

MÉTHODES

- Une valeur d'une classe est appelée un objet ou une instance de cette classe
- Celle-ci nous fournit des méthodes accessibles depuis cet objet

Par exemple : `s.length()` renvoie la taille de la chaîne

Il existe de nombreuses méthodes prédéfinies dans la classe `string`

OPÉRATEURS

- On peut surcharger des opérateurs pour les définir sur n'importe quelle classe :

```
string s1("debut"),s2("fin");  
cout << s1 << endl;  
cout << s2 << endl;  
cout << s1+s2 << endl;
```

PASSAGE ET RETOUR DE CHAINES

- Comme pour une valeur de base on peut passer une chaîne des trois manière standard : directement, par pointeur, par référence

On peut directement renvoyer une `string`

CAST IMPLICITE

- On peut également surcharger les opérateurs de cast
- Il y a une conversion implicite des chaîne à la C vers des string
- par exemple :

```
string test(const string &s) {  
    return s+"bar"+s;  
}  
cout << test("foo");
```

CRÉATION DYNAMIQUE

- Imaginons que notre programme n'utilise pas tout le temps une chaîne, il est inutile de toujours prendre l'espace mémoire lui correspondant

On définit alors un pointeur `string *s`;

Lorsqu'on a besoin de la string on crée s avec `s = new string("foo");`

De même lorsque l'on n'en a plus besoin, on la détruit avec `delete s`;

- On appelle alors un destructeur qui se charge de libérer toute la mémoire