CHAINES DE CARACTÈRES

INTRODUCTION AUX CHAÎNES DE CARACTÈRES

QU'EST-CE QU'UNE CHAÎNE DE CARACTÈRES ?

DÉFINITION

Une **chaîne de caractères** est une séquence de caractères, généralement utilisée pour représenter du **texte**.

Exemple :

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

int main() {
    std::string maChaine = "Bonjour tout le monde!";
    std::cout << maChaine << std::endl;
    return 0;
}
```

En C++, on utilise la classe `std::string` pour créer et manipuler des chaînes de caractères.

EXEMPLES

Voici quelques exemples de **chaînes de caractères** en C++ :

```
"Bonjour"
"12345"
"Je suis une IA."
```

En C++, il est important d'utiliser des guillemets doubles pour déclarer les chaînes de caractères. Les guillemets simples sont utilisés pour les caractères individuels.

TYPES DE CHAÎNES DE CARACTÈRES EN C++

ARRAY DE CHARS (TABLEAUX DE CARACTÈRES)

SYNTAXE

Pour déclarer un tableau de caractères en C++ :

```
char chaine[] = "Bonjour";
```

Les tableaux de caractères sont utilisés pour stocker des chaînes de caractères en C++. N'oubliez pas que la taille du tableau inclut le caractère de fin de chaîne '\0'.

EXEMPLES D'UTILISATION

```
char chaine1[] = "Bonjour";
char chaine2[] = {'B', 'o', 'n', 'j', 'o', 'u', 'r', '\0'};
```

Ici, nous montrons deux façons de déclarer et initialiser des chaînes de caractères en C++. La première utilise des guillemets doubles, tandis que la seconde utilise un tableau de caractères individuels suivi d'un caractère de fin (null), '\0'.

CLASSE STRING C++ (STANDARD LIBRARY)

SYNTAXE

Pour déclarer une **chaîne de caractères** en utilisant la classe `std::string` de la **bibliothèque standard C++** :

```cpp
#include <string>

std::string chaine = "Bonjour";
```

La classe std::string est plus facile à manipuler que les tableaux de caractères (char[]). Elle offre des méthodes pour gérer les chaînes de caractères.

EXEMPLES D'UTILISATION

```cpp
#include <string>

std::string chaine_cpp = "Je suis une IA.";
```

Dans cet exemple, nous déclarons une variable de type `std::string` et lui attribuons une valeur "Je suis une IA.".

OPÉRATIONS SUR LES CHAÎNES DE CARACTÈRES

CONCATÉNATION

MÉTHODE 1 : OPÉRATEUR +

La **concaténation** permet de combiner deux **chaînes de caractères** en une seule.

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

int main() {
    std::string str1 = "Bonjour, ";
    std::string str2 = "comment ça va ?";
    std::string str_concat = str1 + str2;

    std::cout << str_concat << std::endl;

    return 0;
}
```

L'opérateur + est surchargé pour fonctionner avec des chaînes de caractères, il n'additionne pas les caractères, mais les combine pour former une nouvelle chaîne.

SYNTAXE

```
string s1, s2, resultat;
resultat = s1 + s2;
```

Concaténation de chaînes de caractères :

```
string s1, s2, resultat;
resultat = s1 + s2;
```

Cette syntaxe permet de concaténer deux chaînes de caractères en C++. La variable `resultat` contiendra la chaîne de caractères résultante après l'exécution de cette ligne de code.

EXEMPLES D'UTILISATION

```
string prenom = "John";
string nom = "Doe";
string nom_complet = prenom + " " + nom; // "John Doe"
```

MÉTHODE 2 : MÉTHODE `append()`

La méthode `append()` permet également de concaténer deux **chaînes de caractères**.

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

int main() {
  std::string str1 = "Bonjour";
  std::string str2 = " tout le monde!";
  str1.append(str2);
  std::cout << str1 << std::endl;
  return 0;
}
```

La méthode `append()` modifie la chaîne sur laquelle elle est appelée, en ajoutant la chaîne passée en argument à la fin de celle-ci.

SYNTAXE

```
string s1, s2;
s1.append(s2);
```

La fonction `append()` permet de concaténer deux chaînes de caractères (strings) en C++. On prend la chaîne `s1`, à laquelle on ajoute la chaîne `s2` après sa fin.

EXEMPLES D'UTILISATION

```
string prenom = "John";
string nom = "Doe";
prenom.append(" ").append(nom); // "John Doe"
```

La méthode **append()** permet de concaténer deux chaînes de caractères. Ici, on ajoute un espace puis le nom de famille à la variable prenom.

COMPARAISON

OPÉRATEUR ==

L'opérateur == permet de vérifier si deux **chaînes de caractères** sont égales.

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

int main() {
  std::string chaine1 = "Bonjour";
  std::string chaine2 = "Bonjour";

  if (chaine1 == chaine2) {
    std::cout << "Les chaînes sont égales." << std::endl;
  } else {
    std::cout << "Les chaînes ne sont pas égales." << std::endl;
  }

  return 0;
}
```

Évitez d'utiliser == pour comparer des chaînes de caractères en C et utilisez plutôt la fonction `strcmp`.

SYNTAXE

```
string s1, s2;
bool egal = (s1 == s2);
```

Cette syntaxe permet de comparer deux chaînes de caractères **s1** et **s2** en C++. La variable **egal** sera **true** si les deux chaînes sont égales, sinon elle sera **false**.

EXEMPLES D'UTILISATION

```
string s1 = "Bonjour";
string s2 = "Hello";
bool egal = (s1 == s2); // false
```

Il est possible de comparer directement des chaînes de caractères en C++ à l'aide de l'opérateur ==.

OPÉRATEUR !=

L'opérateur != permet de vérifier si deux **chaînes de caractères** sont différentes.

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

int main() {
    std::string texte1 = "Hello";
    std::string texte2 = "Bonjour";

    if (texte1 != texte2) {
        std::cout << "Les deux chaînes sont différentes" << std::endl;
    } else {
        std::cout << "Les deux chaînes sont identiques" << std::endl;
    }

    return 0;
}
```

SYNTAXE

```
string s1, s2;
bool different = (s1 != s2);
```

Cette slide montre comment comparer deux chaînes de caractères pour savoir si elles sont différentes en C++.

EXEMPLES D'UTILISATION

```
string s1 = "Bonjour";
string s2 = "Hello";
bool different = (s1 != s2); // true
```

Les opérateurs de comparaison tels que '!=' sont utilisés pour comparer deux valeurs. Dans cet exemple, on compare deux chaînes de caractères.

MÉTHODE `compare()`

La méthode `compare()` permet de comparer deux **chaînes de caractères** et de retourner un **entier** indiquant le résultat de la comparaison.

```cpp
#include <iostream>
#include <string>
using namespace std;

int main() {
    string s1 = "abc";
    string s2 = "def";
    int resultat = s1.compare(s2);

    cout << "Le resultat de la comparaison est : " << resultat << endl;
    return 0;
}
```

La méthode `compare()` renvoie un entier positif, négatif ou zéro selon que la chaîne est lexicographiquement supérieure, inférieure ou égale à l'autre chaîne.

SYNTAXE

```
string s1, s2;
int resultat = s1.compare(s2);
```

La méthode compare() retourne 0 si les deux chaînes sont égales, un nombre négatif si s1 est inférieure à s2 et un nombre positif si s1 est supérieure à s2.

EXEMPLES D'UTILISATION

```
string s1 = "Bonjour";
string s2 = "Hello";
int resultat = s1.compare(s2); // != 0 car les chaînes sont différentes
resultat = s1.compare("Bonjour"); // == 0 car les chaînes sont identiques
```

La méthode **compare()** est utilisée pour comparer deux chaînes de caractères. Elle renvoie 0 si les chaînes sont identiques, un nombre négatif si la chaîne de gauche est "plus petite" et un nombre positif si la chaîne de gauche est "plus grande".

FONCTIONS UTILES POUR LES CHAÎNES DE CARACTÈRES

LONGUEURS DES CHAÎNES DE CARACTÈRES

MÉTHODE `length()`

Pour obtenir la **longueur** d'une chaîne de caractères en C++, on peut utiliser la méthode `length()` de la classe `string` :

```cpp
#include <iostream>
#include <string>
using namespace std;

int main() {
    string chaine = "Bonjour";
    int longueur = chaine.length();
    cout << longueur << endl; // Affiche 7
}
```

`length()` donne le nombre total des caractères, y compris les espaces, et qu'il est important d'inclure la bibliothèque `<string>` pour utiliser cette méthode.

MÉTHODE `size()`

La méthode `size()` donne également la **longueur** de la chaîne :

```cpp
#include <iostream>
#include <string>
using namespace std;

int main() {
    string chaine = "Bonjour";
    int longueur = chaine.size();
    cout << longueur << endl; // Affiche 7
}
```

La méthode `size()` est similaire à la méthode `length()`, qui peut également être utilisée pour obtenir la longueur d'une chaîne.

ACCÈS AUX CARACTÈRES

OPÉRATEUR []

On peut accéder aux caractères individuels d'une **chaîne** en utilisant l'opérateur [] :

```
#include <iostream>
#include <string>
using namespace std;

int main() {
    string chaine = "Bonjour";
    char c = chaine[1];
    cout << c << endl; // Affiche o
}
```

Rappeler aux stagiaires que l'indexation commence à 0, donc chaine[1] correspond au deuxième caractère de la chaîne.

MÉTHODE `at()`

On peut également accéder aux caractères en utilisant la méthode `at()` :

```cpp
#include <iostream>
#include <string>
using namespace std;

int main() {
    string chaine = "Bonjour";
    char c = chaine.at(1);
    cout << c << endl; // Affiche o
}
```

La méthode `at()` renvoie le caractère à la position spécifiée. Les positions commencent à 0. Si l'index est hors limites, une exception std::out_of_range est lancée.

RECHERCHE DE SOUS-CHAÎNES

MÉTHODE `find()`

Pour trouver la première occurrence d'une sous-chaîne dans une chaîne de caractères, utilisez la méthode `find()`. Si la sous-chaîne est trouvée, elle renvoie l'**indice** du début de la sous-chaîne, sinon elle renvoie `string::npos`.

```cpp
#include <iostream>
#include <string>
using namespace std;

int main() {
    string chaine = "Bonjour tout le monde !";
    size_t index = chaine.find("tout");

    if (index != string::npos) {
        cout << "La sous-chaîne est à l'indice : " << index << endl; // Affiche 8
    } else {
        cout << "Sous-chaîne non trouvée" << endl;
    }
}
```

`string::npos` est une constante spéciale représentant une position invalide dans une chaîne.

CONVERSION DE CHAÎNES DE CARACTÈRES

CONVERSION EN MAJUSCULES ET MINUSCULES

MÉTHODES `toupper()` ET `tolower()`

Pour convertir une chaîne de caractères en **majuscules** ou en **minuscules**, on utilise les fonctions `toupper()` et `tolower()` de la bibliothèque `<cctype>`.

Exemple d'utilisation :

```cpp
#include<iostream>
#include<cctype>
using namespace std;

int main() {
    char caractere = 'a';
    char majuscule = toupper(caractere);
    char minuscule = tolower(majuscule);

    cout << "Caractere original : " << caractere << endl;
    cout << "Caractere en majuscule : " << majuscule << endl;
    cout << "Caractere en minuscule : " << minuscule << endl;

    return 0;
}
```

Il est important de rappeler que `toupper()` et `tolower()` ne sont pas limitées aux variables de type `char`, elles fonctionnent aussi sur les `strings`.

SYNTAXE

```
#include <cctype>

char toupper(char c);
char tolower(char c);
```

Les fonctions `toupper` et `tolower` permettent de convertir un caractère en majuscule ou en minuscule, respectivement. Elles font partie de la bibliothèque `<cctype>`.

EXEMPLES D'UTILISATION

```cpp
#include <iostream>
#include <cctype>
#include <algorithm>

int main() {
    std::string str = "Exemple de Chaîne";

    std::transform(str.begin(), str.end(), str.begin(), ::toupper);
    std::cout << str << std::endl;  // Affiche "EXEMPLE DE CHAÎNE"

    std::transform(str.begin(), str.end(), str.begin(), ::tolower);
    std::cout << str << std::endl;  // Affiche "exemple de chaîne"

    return 0;
}
```

Les fonctions `::toupper` et `::tolower` permettent de convertir une chaîne en majuscules et en minuscules respectivement. Cet exemple illustre l'utilisation de ces fonctions de la bibliothèque `<cctype>` pour manipuler des chaînes de caractères en C++.

CONVERSION DE CHAÎNES EN NOMBRES

FONCTIONS `stoi()`, `stof()` ET `stod()`

Pour convertir une chaîne de caractères en nombre, on utilise les fonctions `stoi()` (**string to int**), `stof()` (**string to float**) et `stod()` (**string to double**) de la bibliothèque `<string>`.

Exemple d'utilisation :

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

int main() {
  std::string str_int = "123";
  std::string str_float = "123.45";
  std::string str_double = "123.4567";

  int num_int = std::stoi(str_int);
  float num_float = std::stof(str_float);
  double num_double = std::stod(str_double);

  std::cout << "Integer: " << num_int << "\\n";
  std::cout << "Float: " << num_float << "\\n";
  std::cout << "Double: " << num_double << "\\n";
```

Les fonctions `stoi()`, `stof()` et `stod()` peuvent lancer une exception `std::invalid_argument` si la chaîne de caractères ne peut pas être convertie en nombre. Assurez-vous de gérer ces exceptions si nécessaire.

EXEMPLES D'UTILISATION

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

int main() {
    std::string str1 = "42";
    int num1 = std::stoi(str1);
    std::cout << num1 << std::endl;  // Affiche "42"

    std::string str2 = "3.14";
    float num2 = std::stof(str2);
    std::cout << num2 << std::endl;  // Affiche "3.14"

    std::string str3 = "2.718";
    double num3 = std::stod(str3);
    std::cout << num3 << std::endl;  // Affiche "2.718"
```

CONVERSION DE NOMBRES EN CHAÎNES

FONCTIONS `to_string()`

Pour convertir un nombre en chaîne de caractères, on utilise la fonction `to_string()` de la bibliothèque `<string>`.

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

int main() {
  int nombre = 42;
  std::string chaine = std::to_string(nombre);
  std::cout << "Nombre converti en chaîne : " << chaine << std::endl;
  return 0;
}
```

`to_string()` est une fonction utile pour combiner des nombres et des chaînes de caractères, et pour afficher des nombres sous forme de texte.

SYNTAXE

```cpp
#include <string>

std::string to_string(int value);
std::string to_string(float value);
std::string to_string(double value);
```

Ces fonctions permettent de convertir des valeurs numériques (int, float, double) en chaînes de caractères (std::string).

EXEMPLES D'UTILISATION

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

int main() {
    int num1 = 42;
    std::string str1 = std::to_string(num1);
    std::cout << str1 << std::endl;  // Affiche "42"

    float num2 = 3.14;
    std::string str2 = std::to_string(num2);
    std::cout << str2 << std::endl;  // Affiche "3.140000"

    double num3 = 2.718;
    std::string str3 = std::to_string(num3);
    std::cout << str3 << std::endl;  // Affiche "2.718000"
```

Les fonctions `std::to_string` permettent de convertir des types numériques, tels que `int`, `float` et `double`, en chaînes de caractères `std::string`.

MANIPULATION DE CHAÎNES DE CARACTÈRES

INSERTION ET SUPPRESSION DE CARACTÈRES

MÉTHODES `insert()` ET `erase()`

SYNTAXE

```
string.insert(pos, str);
string.erase(pos, n);
```

string.insert(pos, str) : insère une chaîne (str) à une position spécifiée (pos) dans la chaîne principale (string).

string.erase(pos, n) : supprime n caractères à partir de la position (pos) dans la chaîne principale (string).

EXEMPLES D'UTILISATION

```
std::string s = "Hello, World!";
s.insert(7, " C++"); // "Hello, C++ World!"
s.erase(7, 4); // "Hello, World!"
```

REMPLACEMENT DE CARACTÈRES

MÉTHODE `replace()`

SYNTAXE

```
string.replace(pos, n, str);
```

string.replace() est utilisé pour remplacer une sous-chaîne dans une chaîne principale.

- **pos** : le point de départ du remplacement dans la chaîne principale.
- **n** : le nombre de caractères à remplacer à partir du point de départ.
- **str**: la sous-chaîne qui doit remplacer la partie existante.

EXEMPLES D'UTILISATION

```
std::string s = "Hello, World!";
s.replace(7, 5, "C++"); // "Hello, C++!"
```

La méthode replace permet de remplacer une partie d'une chaîne de caractères. Les arguments sont la position de début, la longueur de la partie à remplacer et la nouvelle chaîne à insérer.

INVERSION DE CHAÎNES

MÉTHODE `reverse()`

SYNTAXE

```
std::reverse(string.begin(), string.end());
```

std::reverse : fonction permettant d'inverser un conteneur en C++.

Cette fonction est disponible dans la bibliothèque `<algorithm>`. N'oubliez pas d'ajouter `#include <algorithm>` en haut du code source pour pouvoir l'utiliser.

EXEMPLES D'UTILISATION

```
std::string s = "Hello, World!";
std::reverse(s.begin(), s.end()); // "!dlroW ,olleH"
```

Cet exemple montre comment utiliser la fonction `std::reverse` de la bibliothèque standard pour inverser une chaîne de caractères.