

4

Tout ça est bien variable

Vous savez désormais afficher des messages à l'écran. C'est un bon début, mais on est quand même assez limités, n'est-ce pas ? Rassurez-vous, C++ permet de faire bien plus que ça. Dans ce chapitre, nous allons découvrir les littéraux et les variables.

4.1. Les littéraux

Les chaînes de caractères

Que sont les littéraux ? Une valeur constante écrite littéralement dans le code, d'où son nom. Nous avons rencontré un littéral dans le chapitre précédent : `"Hello World !"`. Ce littéral est ce qu'on appelle une *chaîne de caractères*. Les chaînes de caractères ne sont rien d'autre que du texte. On les reconnaît parce qu'elles commencent et finissent par des guillemets `"`.

```
import std;

int main()
{
    std::println("Bonjour");
    std::println("Comment vas-tu ?");
    // Le littéral suivant est aussi une chaîne de caractères.
    // C'est une chaîne de caractères vide.
    std::println("");

    return 0;
}
```

Et ce qui est bien, c'est que `std::println` permet non seulement d'afficher du texte, mais également de le *formater*, c'est-à-dire de lui appliquer des transformations avant de l'afficher. Par exemple, on peut afficher de multiples éléments et choisir dans quel ordre les afficher. Le code suivant montre comment s'y prendre.

```
import std;

int main()
{
```

```

std::println("Aujourd'hui, nous sommes {}.", "vendredi");

std::println("Le premier jour de l'année est le {1} {0}.",
            "janvier",
            "premier"
        );

return 0;
}

```

```

Aujourd'hui, nous sommes vendredi.
Le premier jour de l'année est le premier janvier.

```

Notez qu'on a cette fois plusieurs chaînes de caractères séparées par des virgules entre les deux parenthèses et non plus une seule, comme avant. On remarque aussi la présence d'accolades `{}` dans les chaînes de caractères. Ces accolades sont des *emplacements*, qui seront remplacés par les valeurs que l'on a écrit après la première chaîne de caractères. Ainsi, dans la première ligne, `{}` sera remplacé par "vendredi". Et s'il y a des chiffres entre les accolades, c'est pour indiquer l'ordre des valeurs à afficher. Ainsi, dans la seconde ligne, `{0}` sera remplacé par "janvier" et `{1}` par "premier".

- **4.1.** Complétez le code suivant pour afficher le message *Bonjour, je suis un lecteur et j'apprends le C++.*

```

import std;

int main()
{
    std::println("?, je suis ?? et j'apprends le ?.", "C+
+", "un", "Bonjour", "lecteur");

    return 0;
}

```

✧ [Voir la solution](#)

Les caractères

Tout comme les mots de la langue française sont des regroupements de lettres, les chaînes de caractères sont une suite de caractères simples. En C++, un caractère est encadré par des guillemets simples `'`. L'exemple suivant montre que `std::println` gère très bien les caractères simples en plus des chaînes de caractères.

```
import std;

int main()
{
    // Un caractère peut être une lettre.
    std::println("{}", 'A');
    // Ou bien un chiffre.
    std::println("{}", '7');
    // Ou même de la ponctuation.
    std::println("{}", '!');

    return 0;
}
```



J'ai mis plusieurs caractères entre guillemets simples, comme ceci 'abc' et mon code compile. Ça veut dire que ça marche ?

Si vous avez essayé de compiler, vous avez dû remarquer un message en rouge dans la fenêtre de résultat disant `warning: multi-character character constant [-Wmultichar]`, ainsi qu'un nombre s'affiche au lieu de caractères.

Le `warning` est un message d'avertissement que le compilateur vous envoie, car il ne sait pas si ce que vous avez fait est une erreur d'inattention ou bien une manœuvre volontaire de votre part. Dans notre cas, la norme C++ n'interdit pas de faire ça, mais ce n'est pas considéré comme une bonne pratique, comme étant du bon code.

Attention > *N'ignorez pas les warnings ! Les warnings sont des messages importants signalant d'éventuels problèmes. Il ne faut surtout pas les ignorer sous prétexte que le code compile !*

Les caractères spéciaux

Il existe quelques caractères qui sont un peu particuliers et, pour vous les introduire, vous allez afficher un message contenant un chemin de dossier Windows [`C:\Program Files` par exemple].

```
import std;

int main()
{
    std::println("Dossier principal : C:\Program Files (x86)");
    return 0;
}
```

Boum ba da boum ! Le compilateur vous crache à la figure un message de type `warning` : `unknown escape sequence '\P'` (GCC / Clang) ou `warning C4129: « P » : séquence d'échappement de caractères non reconnue` (Visual Studio). Que s'est-il passé ?

Il existe en C++ des séries de caractères, appelées *séquences d'échappement*, qui commencent toutes par `\` et dont vous trouverez la liste complète dans le [manuel de référence](#). Et pour comprendre l'intérêt de ces séquences d'échappement, essayez donc de compiler le code suivant.

```
import std;

int main()
{
    std::println("Il m'a demandé "Comment vas-tu ?");
    return 0;
}
```

Vous devez certainement avoir de nombreuses erreurs qui s'affichent. C'est tout à fait normal. En effet, nous avons vu plus haut que les chaînes de caractères sont délimitées par les doubles guillemets `""`. Donc tout ce qui est entre cette paire de guillemets est considéré par le compilateur comme faisant partie de la chaîne de caractères.

Dans notre exemple, cela veut dire que `Comment vas-tu ?` ne fait plus partie de la chaîne. Donc le compilateur l'interprète comme des instructions C++ et comme il ne les comprend pas, il ne peut pas compiler le programme. D'ailleurs, la coloration syntaxique n'affiche plus les mêmes couleurs que pour les chaînes de caractères, parce que elle aussi est perdue.



Mais quel rapport avec ces fameux caractères commençant par `\` ?

Eh bien, les séquences d'échappement permettent de dire au compilateur *Écoute, ce caractère est spécial, il faut l'afficher et non l'interpréter*, ce qui permet d'afficher des doubles guillemets dans une chaîne de caractères par exemple.

```
import std;

int main()
{
    std::println("Il m'a demandé \"Comment vas - tu ? \");
    std::println("Dossier principal : \"C:\\Program Files (x86)\");
    return 0;
}
```

De toutes les séquences d'échappement qui existent, les plus utilisées et celles que vous verrez le plus souvent sont les suivantes :