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

Que sont les littéraux ? Une valeur 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*. Mais ce n'est pas le seul littéral possible en C++. Examinons tout ça.

Les caractères

Nous avons fait connaissance avec les chaînes de caractères lors du chapitre précédent. Les chaînes de caractères ne sont rien d'autre que du texte. On les reconnaît parce qu'elles commencent et finissent par des doubles guillemets `"`.

► **4.1.** Entraînez-vous donc à afficher des chaînes de caractères, par exemple un littéral souhaitant une bonne journée.

Voir la solution

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 `'`.

```
#include <iostream>

int main()
{
  // Un caractère peut être une lettre.
  std::cout << 'A' << std::endl;
  // Ou bien un chiffre.
  std::cout << '7' << std::endl;
  // Ou même de la ponctuation.
```

```
  std::cout << '!' << std::endl;

  return 0;
}
```

Les chaînes de caractères permettent de regrouper des caractères et nous facilitent ainsi la vie pour écrire du texte.

```
#include <iostream>

int main()
{
  // Que c'est fastidieux d'écrire ça !
  std::cout << 'S' << 'a'
    << 'l' << 'u' << 't'
    << ' ' << 't' << 'o'
    << 'i' << ' ' << '!'
    << std::endl;

  // C'est tellement mieux comme ça.
  std::cout << "Salut toi !" << std::endl;

  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`, ainsi qu'un nombre au lieu de nos 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

► **4.2.** 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 (x86)` par exemple].

Voir la solution

Boum ba da boum ! Le compilateur vous crache à la figure un message de type `warning: unknown escape sequence: '\P'`. 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.

```
#include <iostream>

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

Vous devez certainement avoir une erreur proche de celle-ci : `'error: expected ';' before 'Comment'`. 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.

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.

```
#include <iostream>

int main()
{
  std::cout << "Il m'a demandé \"Comment vas-tu ?\"" << std::endl;
  std::cout << "Dossier principal : C:\\Program Files (x86)"
    << std::endl;

  return 0;
}
```

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

- `\'` qui permet d'afficher un guillemet simple `'` ;
- `\"` qui permet d'afficher un guillemet double `"` ;
- `\n` qui permet d'aller à la ligne, comme `std::endl` ;
- `\t` qui permet de faire une tabulation horizontale ;
- `\\` qui permet d'afficher un antislash `\`.

► **4.3.** Entraînez-vous donc en affichant un guillemet simple sans échappement, puis avec échappement. Et amusez-vous avec les tabulations et les retours à la ligne.

Voir la solution

Les nombres

Il existe d'autres types de littéraux en C++ : les nombres entiers et les nombres à virgule, appelés *flottants*.

```
std::cout << -1 << std::endl;
std::cout << 0 << std::endl;
std::cout << 1 << std::endl;

std::cout << -1.6027 << std::endl;
std::cout << 3.14159 << std::endl;
std::cout << 2.71828 << std::endl;
```

Vous remarquerez qu'on peut utiliser des nombres négatifs sans aucun problème. Ensuite, dû à l'origine américaine de C++, les flottants s'écrivent avec un point et non une virgule. Même si l'on se fiche des chiffres après la virgule, il faut mettre le point `.` sinon C++ interprétera le nombre comme un entier.