# Faire des calculs

Dans le chapitre précédent, vous avez appris à utiliser `cout` pour afficher des messages. Les messages sont des chaînes de caractères encadrées par des guillemets droits ". Une valeur écrite directement dans le code C++ est appelée une littérale. Il existe d'autres types de littérales :

- les nombres entiers : `0`, `1`, `2`, `-1`, `-2`, etc.
- les nombres réels : `1.0`, `2.1`, `-5.12`, `1.457`, etc.
- les booléens, qui représente une valeur à deux états : `true` (vrai) et `false` (faux).
- les caractères : 'a', 'b', 'c', etc.
- et les chaînes de caractères que vous avez déjà vu : "hello, world", "salut tout le monde", etc.

Vous pouvez afficher ces littérales directement en utilisant `cout`, comme vous l'avez vu pour les chaînes de caractères.

### Les nombres entiers

## Écrire des nombres entiers

Les nombres entiers ne devraient pas vous poser de problème, ce sont les nombres que vous utiliser pour compter depuis l'école maternelle. La forme la plus simple pour écrire un nombre entier est d'écrire une série continue de chiffres entre 0 et 9. Les chiffres zéro au début seront ignorés.

main.cpp

```
#include <iostream>
```

```cpp
int main() {
    std::cout << 1 << std::endl; // affiche le nombre 1
    std::cout << 2 << std::endl; // affiche le nombre 2
    std::cout << 3 << std::endl; // affiche le nombre 3
    std::cout << 4 << std::endl; // affiche le nombre 4
    std::cout << 0004 << std::endl; // affiche le nombre 4
}
```

Pour écrire un nombre entier négatif, vous devez ajouter le signe moins devant le nombre.

main.cpp

```cpp
#include <iostream>

int main() {
    std::cout << -1 << std::endl; // affiche le nombre -1
    std::cout << -2 << std::endl; // affiche le nombre -2
    std::cout << -3 << std::endl; // affiche le nombre -3
    std::cout << -4 << std::endl; // affiche le nombre -4
}
```

Lorsque vous écrivez de très grands nombres, il est difficile de le lire. Par exemple :

main.cpp

```cpp
#include <iostream>

int main() {
    std::cout << 123456789123456789 << std::endl;
}
```

La raison est que le cerveau humain est capable de reconnaître des groupes de quelques caractères, mais pas un seul bloc de 18 caractères. Il n'arrive donc pas, en un coup d'oeil, à identifier si ce nombre est de l'ordre du million, du milliard ou autre.

Pour faciliter la lecture, il est donc possible d'ajouter un caractère guillemet droit ' pour séparer un grand nombre en plusieurs groupes. Par habitude, on séparera en groupes de trois chiffres :

main.cpp

```
#include <iostream>

int main() {
    std::cout << 123'456'789'123'456'789 << std::endl;
}
```

Vous pouvez écrire des nombres très grand de cette manière, mais il existe une limite. Si vous écrivez un nombre trop grand, vous aurez un message d'erreur signalant que ce nombre est trop grand.

main.cpp

```
#include <iostream>

int main() {
    std::cout << 123456789123456789 << std::endl; // affiche le nombre 123456789123456789
}
```

Produira l'erreur :

```
main.cpp:4:18: error: integer constant is larger than the largest unsigned integer type
    std::cout << 123456789123456789123456789123456789 << std::endl; // affiche le nombre 123456789123456789
                 ^
1 error generated.
```

L'existence d'une valeur maximale limite est liée à la représentation des nombres dans la mémoire des ordinateurs et à la notion de type de données. Vous verrez cela dans les prochains chapitres.

Il est bien sûr possible d'utiliser des nombres avec autant de chiffre que vous souhaitez, mais il ne sera pas possible d'utiliser dans ce cas les nombres entiers tel que définit dans ce chapitre. Vous apprendrez à créer dans un travail pratique dans un prochain chapitre comment créer ce type de nombres entiers.

## Les nombres décimaux, hexadécimaux, octaux et binaires

### les nombres réels

flotting point

représentation scientifique, + et -, chiffres décimaux

### Les caractères et les chaînes de caractères

### Mettre en forme la sortie

utiliser setw, setprecision. tabulation, retour à la ligne

### Les caractères spéciaux

Au cours de vos essais, vous avez peut-être essayé d'afficher un backslash (\) ou des guillemets (”). Si ce n'est pas le cas, je vous propose de le faire maintenant:

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

int main()
{
  cout << "Je fais des tests pour apprendre le C++ !" <<
endl;
  cout << """ << endl;
  cout << "\" << endl;
  return 0;
}
```

Le compilateur ne va pas aimer cela du tout et il un message d'erreur devrait s'afficher dans la zone au bas de votre fenêtre Code::Blocks. La

raison est simple, pour afficher des guillemets, il faut utiliser la combinaison `\"` et pas juste `"`, idem pour le backslash qu'il faut doubler (Pourquoi ? Et parler des Raw String). Il faut donc écrire:

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

int main()
{
  cout << "Je fais des tests pour apprendre le C++ !" <<
endl;
  cout << "\"" << endl;
  cout << "\\" << endl;
  return 0;
}
```

Équivalent C++11 :

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

int main()
{
  cout << "Je fais des tests pour apprendre le C++ !" <<
endl;
  cout << R"(")" << endl;
  cout << R"(\)" << endl;
  return 0;
}
```

Je vous laisse faire le test pour vérifier que cela fonctionne. Maintenant que vous avez vu ces deux petites exceptions, vous êtes prêt à écrire tout ce qui vous passera par la tête dans la console. Voyons maintenant ce qui se passe à la fin de notre programme.

| Chapitre précédant | Sommaire principal | Chapitre suivant |
| --- | --- | --- |

Cours, C++