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 nombres ne doivent pas commencer par zéro.

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
}
```

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 << 1234567891234567890 << 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

```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

```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 entiers que vous utilisez habituellement s'écrivent à partir de dix chiffres (0 à 9). C'est pour cette raison que l'on parle de système décimal (du latin “decimus”, qui signifie “dixième”) et l'on parle de base 10. Mais ce n'est pas la seule façon d'écrire les nombres et il existe d'autres systèmes numériques.

Imaginons par exemple que l'on souhaite écrire les nombres en utilisant que huit chiffres (0 à 7). Dans ce cas, nous pouvons compter de la façon suivante : 0, 1, 2, 3, 4, 5, 6, 7. Arrivé au huitième chiffre, nous ne pouvons pas écrire “8”, puisque ce chiffre n'est pas autorisé dans ce système décimal. Donc, il faut passer à un nombre à deux chiffres : 0, 1, 2, 3, 4, 5, 6, 7, 10, 11, 12, 13, 14, 15, 16, 17, 20, 21, etc.

Vous verrez dans les exercices comment convertir un nombre écrit selon une base dans une autre base.

En C++, il est possible d'écrire et d'afficher des nombres écrit selon des bases différentes de 10. Pour des raisons historiques, les ordinateurs sont habitués à manipuler les nombres en base 2 (binaire), 8 (octal), 16 (hexadécimal). Pour écrire un nombre en base différente de 10, il faut commencer le nombre par le chiffre 0 puis un caractère pour spécifier la base : rien pour octal, x ou X pour l'hexadécimal et b pour le binaire. Les chiffres autorisés pour écrire un nombre dépendent également du mode : 0 à 7 pour l'octal, 0 à 9 et a à f (ou A à) pour l'hexadécimal et 0 et 1 pour le binaire.

Le code suivant permet d'afficher la valeur de 10 selon la base :

main.cpp

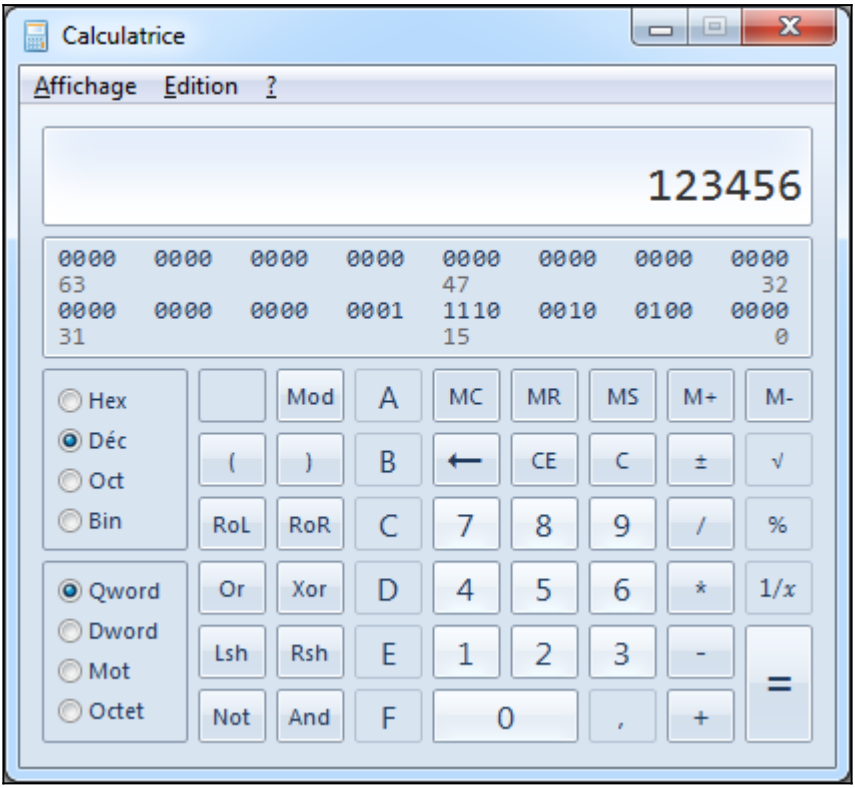
```cpp
#include <iostream>

int main() {
    std::cout << 10 << std::endl;
    std::cout << 010 << std::endl;
    std::cout << 0x10 << std::endl;
    std::cout << 0b10 << std::endl;
```

```
}
```

affiche :

```
10
8
16
2
```

Il est possible calculer les conversions à la main, mais c'est plus simple d'utiliser un convertisseur. Il existe des convertisseurs en ligne, vous pouvez également utiliser la calculatrice de Windows en sélectionnant le mode “Programmeur” dans le menu “Affichage”.



Il faut donc bien faire attention à la base utilisée lorsque l'on écrit un

nombre.

Base	Système	Préfixe	Chiffres	Exemple
2	binaire	0b	0 et 1	`0b01`
8	octal	0	0 à 7	`001234567`
10	décimal		0 à 9	`1234567890`
16	hexadécimal	0x	0 à 9 et a à f	`0x0123456789abcdef`
		0X	0 à 9 et A à F	`0X0123456789ABCDEF`

suffixe ? u l et ll

Comme vous l'avez vu dans les codes précédents, les nombres sont affichés après conversion en base 10. C'est le mode de fonctionnement par défaut de `cout`, mais il est possible de modifier ce comportement. Les directives suivantes permettent de modifier la base utilisée par `cout` pour afficher les nombres :

- `std::oct` pour afficher en utilisant la base 8 ;
- `std::dec` pour afficher en utilisant la base 10 ;
- `std::hex` pour afficher en utilisant la base 16.

Il n'existe pas de directive permettant d'afficher les nombres en binaire, vous verrez dans un exercice dans un prochain chapitre comment afficher des nombres en binaire. Par défaut, `cout` affiche les nombres en utilisant la base 10.

main.cpp

```cpp
#include <iostream>

int main() {
    std::cout << std::hex;
    std::cout << 10 << std::endl;
    std::cout << 010 << std::endl;
    std::cout << 0x10 << std::endl;
    std::cout << 0b10 << std::endl;
}
```

affiche :

```
a
8
10
2
```

Comme vous le voyez, une directive continue de s'appliquer sur toutes les lignes suivantes, tant que vous ne donner pas une directive modifiant une nouvelle fois l'affichage. Vous pouvez écrire les directives sur une ligne différente (comme dans le code précédent), sur la première ligne ou à chaque ligne, cela ne change rien au résultat.

Si vous manipuler et afficher plusieurs bases, il peut être difficile de savoir exactement ce que vous afficher, avec quelle base. Il est dans ce cas possible d'afficher la base utilisée pour l'affichage avec la directive `std::showbase`. Pour ne plus afficher la base, vous pouvez utiliser la directive `std::noshowbase`.

main.cpp

```cpp
#include <iostream>

int main() {
    std::cout << 10 << std::endl;
    std::cout << 010 << std::endl;
    std::cout << 0x10 << std::endl;
    std::cout << 0b10 << std::endl;

    std::cout << std::hex << std::showbase;
    std::cout << 10 << std::endl;
    std::cout << 010 << std::endl;
    std::cout << 0x10 << std::endl;
    std::cout << 0b10 << std::endl;
}
```

affiche :

```
10
8
16
2
0xa
```

```
0x8
0x10
0x2
```

Faire des calculs sur les entiers

Pouvoir écrire des nombres entiers est une chose

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

Exercices

Conversion en nombre de n'importe quelle base

http://fr.wikipedia.org/wiki/Syst%C3%A8me_d%C3%A9cimal

opérations à faire pour les calculs ?

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++