Todo : tabulation, retour à la ligne, quoted. Caractère spéciaux sous forme hexa.

Les littérales chaînes de caractères

Chaînes de caractères et caractères

Comme vous pouvez remarquer, on distingue en C++ les caractères uniques et les chaînes de caractères (un ensemble de caractères). Un caractère unique sera écrit avec un guillemet simple droit, une chaîne de caractères sera écrite avec des guillemets doubles droits :

main.cpp

```
#include <iostream>

int main() {
    std::cout << "hello";  // une chaîne
    std::cout << 'h';      // un caractère
}
```

Si vous essayez de mettre plusieurs caractères entre guillemets droits, vous obtiendrez généralement un message d'avertissement. Par exemple, le message est avec Clang dans Coliru :

main.cpp

```
#include <iostream>

int main() {
    std::cout << 'hello, world!' << std::endl;
}
```

affichera :

```
main.cpp:4:18: warning: character constant too long for its
type
```

```
    std::cout << 'hello, world!' << std::endl;
                 ^
1919706145
```

Remarquez bien que cela provoque un avertissement (*warning*) et non une erreur : le programme s'exécute quand même. La chaîne est interprétée comme un nombre, qui est affiché en dessous de l'avertissement (`1919706145` dans l'exemple précédent). Les avertissements signifient que le compilateur ne sait pas si le code contient une erreur ou si c'est intentionnel de la part du développeur.

Vous ne devez jamais ignorer les avertissements. Dans ce cours, un programme qui affiche des avertissements ne sera pas considéré comme correct.

Les chaînes de caractères ne sont qu'une suite de caractères, sans aucun sens particulier pour le C++. Cela implique que ce que vous mettez dedans ne sera pas évalué. Si vous écrivez par exemple :

main.cpp

```
#include <iostream>

int main() {
    std::cout << "std::endl" << std::endl;  // code dans une chaîne
    std::cout << std::endl << std::endl;    // code hors d'une chaîne

    std::cout << "1+2" << std::endl;        // expression dans une chaîne
    std::cout << 1+2 << std::endl;          // expression hors d'une chaîne
}
```

affiche :

```
std::endl


1+2
```

```
3
```

On voit dans cet exemple simple que les chaînes “std::endl” et “1+2” s'affichent exactement tel quel, alors que le même code en dehors de la chaîne est évalué.

Les caractères spéciaux

Si vous vous êtes amusé à tester différents messages à afficher, vous avez peut-être essayé d'afficher une barre oblique inversée `\` (*backslash*) ou des guillemets `”`. Si ce n'est pas le cas, essayez maintenant :

main.cpp

```cpp
#include <iostream>

int main() {
  std::cout << "hello " world" << std::endl;
}
```

Le compilateur ne va pas du tout aimer ce code et produira des erreurs et avertissements :

```
main.cpp:4:28: error: expected ';' after expression

      std::cout << "hello " world" << std::endl;

                           ^

                           ;

main.cpp:4:34: warning: missing terminating '"' character
[-Winvalid-pp-token]

      std::cout << "hello " world" << std::endl;

                                  ^

main.cpp:4:29: error: use of undeclared identifier 'world'
```

```
      std::cout << "hello " world" << std::endl;
                                ^

1 warning and 2 errors generated.
```

L'erreur est simple à comprendre avec un peu de logique. En C++, une chaîne de caractères est délimitée par des guillemets. Le compilateur va rencontrer le premier guillemet et va l'interpréter comme le début d'une chaîne (et donc tout ce qui suit sera considéré comme des caractères faisant partie de la chaîne).

```
"hello " world"
^      ^      ^
1      2      3
```

Le compilateur continue de parcourir le code et va arriver au second guillemet. Le problème est qu'il ne sait pas qu'il y a encore un guillemet ensuite, il va considérer que ce guillemet est la fin de la chaîne. Tout ce qui suit sera donc considéré comme du code C++ et non comme des caractères à afficher. Or, comme la suite n'est pas du code C++ valide, le compilateur produit une erreur.

Pour résoudre ce problème, il faut pouvoir dire au compilateur que l'on souhaite utiliser le caractère guillemet et non indiquer la fin de la chaîne. Pour cela, il faut utiliser un caractère spécial, le caractère d'échappement `\`. Ce caractère sera interprété de la façon suivante par le compilateur lorsqu'il est rencontré dans une chaîne : “attention, le caractère suivant est un caractère spécial”.

En particulier, le caractère `\”` est interprété comme étant le caractère guillemet et non la fin d'une chaîne. Avec cette correction, le code devient :

main.cpp

```cpp
#include <iostream>

int main() {
  std::cout << "hello \" world" << std::endl;
```

```
}
```

affiche :

```
hello " world
```

Remarquez que l'ensemble des deux caractères `\` et `"` est interprété comme étant un seul caractère.

Un autre problème se présente. Puisque le caractère `\` est considéré comme étant le caractère d'échappement, comment afficher le caractère `\` dans une chaîne ? La solution est simple, il suffit d'utiliser le caractère '\\'.

Encore une fois, il s'agit bien d'une séquence d'échappement, qui correspond à un seul caractère dans la chaîne et qui est constituée du caractère d'échappement `\` puis de l'identifiant `\`.

main.cpp

```cpp
#include <iostream>

int main() {
  std::cout << "hello \\ world" << std::endl;
}
```

affiche :

```
hello \ world
```

La liste des séquences d'échappement est donnée dans la documentation : Escape sequences.

Les chaînes de caractères brutes

Dans certains cas, une chaîne va contenir de nombreux caractères spéciaux (par exemple, le chemin d'un fichier sous Windows, qui peut contenir plusieurs `\` ou des expressions régulières). Dans ce cas, il devient fastidieux (c'est pas faux !) de devoir ajouter systématiquement le caractère d'échappement `\` devant chaque caractère spécial.

Pour éviter cela, il est possible d'utiliser une chaîne brute, dans laquelle les caractères spéciaux sont ignorés. Une chaîne brute doit commencer (par défaut) par `R"(` et se terminer par `)"`, tout ce qui se trouve entre les deux sera interprété comme des caractères à afficher.

main.cpp
```
#include <iostream>

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

affiche :

```
Je fais des " tests pour \ apprendre le C++ !
```

Remarquez bien que les parenthèses en début et fin de chaîne (celles correspondantes à `R"(` et `)"`) ne font pas partie de la chaîne et ne sont pas affichées.

Si vous êtes attentif (et que vous cherchez un peu les problèmes...), vous avez peut être testé un code similaire au suivant :

main.cpp
```
#include <iostream>

int main() {
  std::cout << R"(Un autre )" test pour apprendre le C++ ! )"
<< std::endl;
}
```

On retrouve le problème précédent, à savoir que l'on a une séquence de caractères dans la chaîne qui est similaire à la séquence de caractères utilisée pour indiquer la fin de la chaîne. Ce code retourne, comme on peut s'y attendre, des erreurs :

```
main.cpp:4:30: error: expected ';' after expression
  std::cout << R"(Un autre )" test pour apprendre le C++ !
)" << std::endl;
```

```
                                ^
                                ;
main.cpp:4:31: error: unknown type name 'test'
  std::cout << R"(Un autre )" test pour apprendre le C++ !
)" << std::endl;
                                 ^
main.cpp:4:40: error: expected ';' at end of declaration
  std::cout << R"(Un autre )" test pour apprendre le C++ !
)" << std::endl;
                                          ^
                                          ;
main.cpp:4:61: warning: missing terminating '"' character
[-Winvalid-pp-token]
  std::cout << R"(Un autre )" test pour apprendre le C++ !
)" << std::endl;

^
1 warning and 3 errors generated.
```

Heureusement, il existe une solution : il est possible d'ajouter une séquence de caractères quelconque dans les guillemets et les parenthèses délimitant la chaîne. Cette séquence de caractères permet de spécifier des délimiteurs différents de n'importe quelle séquence contenu dans la chaîne et donc d'éviter les ambiguïtés. Par exemple :

main.cpp

```cpp
#include <iostream>

int main() {
  std::cout << R"*(Un autre )" test pour apprendre le C++ ! )
*" << std::endl;
}
```

affiche le résultat attendu :

```
Un autre )" test pour apprendre le C++ !
```

Ici, la chaîne brute est délimitée par les séquences `R"*(` et `)*"`.

Exercices

Modifier le code suivant pour afficher les messages demandés. Faire l'exercice

main.cpp

```cpp
#include <iostream>

int main() {
    // Modifier la ligne suivante pour afficher "Bienvenue
tout le monde !"
    std::cout << "Hello, world!" << std::endl;

    // Ajouter UNE ligne pour afficher "Bienvenue !" et
"Tout le monde !" sur DEUX lignes
    std::cout << @@@@

    // Ajouter DEUX lignes pour afficher "Bienvenue !" et
"Tout le monde !" sur UNE ligne
    std::cout << @@@@
    std::cout << @@@@

    return 0;
}
```

Exos : utiliser tabulation pour afficher un tableau

Exos : donner des chaînes à afficher et écrire le code correspondant

Exos : Afficher un tableau avec bordures

main.cpp

```cpp
#include <iostream>

int main() {
  std::cout << "┌────┬────┬────┐" << std::endl;
  std::cout << "│ 12 │ 34 │ 56 │" << std::endl;
  std::cout << "├────┼────┼────┤" << std::endl;
  std::cout << "│ ab │ cd │ ef │" << std::endl;
  std::cout << "└────┴────┴────┘" << std::endl;
```

```
}
```

affiche :

```
┌────┬────┬────┐
│ 12 │ 34 │ 56 │
├────┼────┼────┤
│ ab │ cd │ ef │
└────┴────┴────┘
```

Faire pareil, avec double bordure.

Exos : ASCII art

Chapitre précédent	Sommaire principal	Chapitre suivant

Cours, C++