Les chapitres suivants sont encore en cours de rédaction, voire à l'état d'ébauche. N'hésitez pas a faire des remarques ou poser des questions sur le forum de Zeste de Savoir ou de OpenClassroom.

Constantes et énumérations

constantes

litterale = valeur ecrite directement dans le code. Parfois, une litterale a un sens particulier, par exemple une constante de physique ou de mathematique. Il est classique de nommer cette valeur pour la lisibilite = constante.

Syntaxe tres specifique :

```
template<typename T>
constexpr T pi = T(3.1415926535897932385);
```

Signifie que l'on cree un constante "pi", qui prend la valeur "3.1415...". Le type de cette constante depend de T, qui représente un type fondamental, comme int, double, etc...

Une constante est simplement une variable dont la valeur ne change pas.

Anciennes syntaxes

Sans template :

```
const double pi { 3.1415926535897932385 };
```

Très très old school : avec #define

Les enumérations

Une enumeration est une liste de valeurs constantes. Chaque valeur a un nom et l'ensemble des noms sont regroupes dans le nom de l'enumeration.

Pour declarer une enumeration et des valeurs, la syntaxe generale est la suivante :

```
enum class NOM_ENUMERATION {
    LISTE_DE_VALEURS
};
```

Le nom de l'enumeration est “NOM_ENUMERATION”. Par exemple, pour definir une enumeration de couleurs (vide pour le moment) :

```
enum class Couleur {
    ...
};
```

La liste de valeurs est une liste d'identifiants separees par des virgules.

```
NOM_VAlEUR_1, NOM_VALEUR_2, ..., NOM_VALEUR_N
```

Par exemple :

```
enum class Couleur {
    Noir, Blanc, Rouge, Jaune, Bleu, Orange, Vert, Violet
};
```

(Note : pour rappel, les retours a la ligne sont utilises librement en C++. Cette liste est definie sur une seule ligne, mais ca serait equivalent d'ecrire chaque valeur sur une ligne differente).

Dans cet exemple, l'enumeration s'appelle `Couleur`. Elle contient 8 valeurs : `Noir` a `Violet`.

Une enumeration definie un nouveau type, il est donc possible de definir une variable du type “NOM_ENUMERATION”. Pour utiliser les valeurs definies dans une enumeration, il faut simplement utiliser l'operateur de

portee `::` (scope operator).

```
NOM_ENUMERATION NOM_VARIABLE {};
NOM_VARIABLE = NOM_ENUMERATION :: NOM_VALEUR;
```

Par exemple :

```
enum class Couleur {
    Noir, Blanc, Rouge, Jaune, Bleu, Orange, Vert, Violet
};

Couleur une_couleur { Couleur::Rouge };  // initialisation
une_couleur = Couleur::Bleu;             // affectation
```

Les valeurs des enumerations peuvent etre converties en nombre entier, en utilisant l'operateur de conversion “static_cast” :

```
const auto c = static_cast<int>(une_couleur);  // convertie
le type "Couleur" en type "int"
```

Par defaut, les valeurs de l'enumeration sont converties en entier selon leur ordre dans l'enumeration, en commencant de 0 :

```
std::cout << static_cast<int>(Couleur::Noir) << std::endl;
std::cout << static_cast<int>(Couleur::Blanc) << std::endl;
std::cout << static_cast<int>(Couleur::Rouge) << std::endl;
std::cout << static_cast<int>(Couleur::Vert) << std::endl;
```

affiche :

```
0
1
2
6
```

Il est possible de definir la valeur entiere correspondant a chaque valeur de l'enumeration en utilisant =. Les valeurs qui ne sont pas explicitement definies prennent la valeur precedente + 1.

```
enum class Couleur {
    Noir,         // prend automatiquement la valeur 0
```

```
    Blanc,        // prend automatiquement la valeur 1
    Rouge = 100,  // prend explicitement la valeur 100
    Jaune = 200,  // prend explicitement la valeur 200
    Bleu,         // prend automatiquement la valeur 201
    Orange,       // prend automatiquement la valeur 202
    Vert,         // prend automatiquement la valeur 203
    Violet        // prend automatiquement la valeur 204
};

std::cout << static_cast<int>(Couleur::Rouge) << std::endl;
std::cout << static_cast<int>(Couleur::Vert) << std::endl;
```

affiche :

```
100
203
```

Il est également possible de convertir un nombre entier en valeur d'une enumeration avec “static_cast”.

```
const auto c = static_cast<Couleur>(1);
assert(c == Couleur::Blanc);
```

Ancienne syntaxe

Meme syntaxe, sans le mot-clé `class`.

```
enum Couleur {
    Noir, Blanc, Rouge, Jaune, Bleu, Orange, Vert, Violet
};

Couleur une_couleur { Rouge };  // initialisation
une_couleur = Bleu;             // affectation
```

Definit les enum dans la portee globale (*unscoped enum*). “Dans la portée globale” signifie qu'il n'est pas nécessaire d'indiquer la portée `Couleur::`. Cela peut provoquer des conflits, par exemple :

```
enum Pion {
    Noir, Blanc
```

```
};

enum Couleur {
    Noir, Blanc, Rouge, Jaune, Bleu, Orange, Vert, Violet //
confit
};
```

affiche les erreurs suivantes :

```
main.cpp:8:5: error: redefinition of enumerator 'Noir'
    Noir, Blanc, Rouge, Jaune, Bleu, Orange, Vert, Violet //
confit
    ^
main.cpp:4:5: note: previous definition is here
    Noir, Blanc
    ^
main.cpp:8:11: error: redefinition of enumerator 'Blanc'
    Noir, Blanc, Rouge, Jaune, Bleu, Orange, Vert, Violet //
confit
          ^
main.cpp:4:11: note: previous definition is here
    Noir, Blanc
          ^
```

Ce type d'énumération ne nécessite pas de conversion explicite avec `static_cast` pour convertir en entier, les valeurs sont convertie implicitement (promotion) entiers.

Chapitre précédent	Sommaire principal	Chapitre suivant