

Constantes et énumérations

constantes

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

Syntaxe très spécifique :

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

Signifie que l'on crée une constante "pi", qui prend la valeur "3.1415...". Le type de cette constante dépend 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

énumérations

Plusieurs constantes, regroupées dans un même scope.

```
enum class MyEnum {
```

```
    value_1 = 123,  
    value_2 = 456  
};
```

Utilisation : avec operateur de portee `::`

```
int x = MyEnum::value_1;
```

Anciennes syntaxes

Sans le mot cle `class`. Definit les enum dans la portee globale.

Notion de portée

```
enum class MyEnum {  
    value_1 = 123  
};  
  
int value_1 = 456;
```

L'identifiant `value_1` est defini 2 fois. Dans les regles de nommage, on a dit que chaque identifiant devait etre defini 1 seule fois, pourquoi cela fonctionne ? Par ce qu'ils sont dans des portees differentes.

`main.cpp`

```
#include <iostream>  
  
int main() {  
    enum class MyEnum {  
        value_1 = 123  
    };  
  
    int value_1 = 456;  
  
    std::cout << value_1 << std::endl;  
    std::cout << static_cast<int>(MyEnum::value_1) << std::endl;  
}
```

