[C++17] Structures de controle

Clause d'initialisation

Dans un test `if-else` classique, les variables déclarées dans les blocs ont une portée locale, les variables déclarées en dehors des blocs ont une portée plus globale.

```
if (x < 100) {
    int i { 123 }
} else {
    instruction2;
}
int i { 321 }; // ok
```

Ce code est valide, puisque la variable `i` déclarée dans le bloc n'a pas la même portée que la variable `i` déclarée en dehors de l'instruction `if-else`.

Supposez maintenant que vous souhaitez utiliser une variable qui sera utilisée dans les deux blocs de code. Par exemple, pour afficher la valeur absolue d'un entier :

```
int i { 123 };
if (i < 0) {
    std::cout << (-i) << std::endl;
} else {
    std::cout << i << std::endl;
}
int i { 321 }; // erreur
```

Dans ce code, une variable `i` est utilisée dans les deux blocs d'instructions du test `if-else`. Pour cela, cette variable est déclarée avant le test, et celle-ci est accessible dans les blocs du fait que ce sont des blocs enfants de la portée de la variable.

Le problème est que la variable a maintenant la même portée que la seconde declaration de la variable `i`, ce qui provoque un conflit.

Plusieurs solutions sont possibles pour regler ce probleme :

- changer les noms des variables pour éviter les conflits. Mais cela peut nuire à la lisibilité.
- déclarer la variable `i` dans les blocs.

operateur d'affectation
if (int i { 123 })

```
  ...
```

Une clause d'initialisation permet d'exécuter un code d'initialisation dans les parenthèses de la clause `if`. La syntaxe est la suivante :

```
if (INITIALISATION; CONDITION_BOOLEENNE)
    ....
```

Vous pouvez par exemple déclarer une nouvelle variable dans cette clause d'initialisation :

```
if (int i { 123 }; x < 100)
    instruction1;
else
    instruction2;
```

Dans ce code, vous pouvez reconnaître la déclaration d'une variable `int i { 123 };` et une expression booléenne utilisant un opérateur de comparaison `x < 100`.

Il est également possible de déclarer plusieurs variables, en utilisant l'operateur *comma* (virgule).

```
if (int i{1}, j{2}, k{3}; x < 100)
    ...
```

La déclaration de plusieurs variables dans une même instruction en utilisant l'operateur *comma* diminue la lisibilité du code et augmente les risques d'erreur, c'est donc une pratique déconseillée.

Cependant, cette syntaxe est la seule utilisation dans une clause d'initialisation si vous souhaitez déclarer plusieurs variables. C'est donc une syntaxe acceptable dans ce cas particulier.

constexpr-if [C++17]

Dans une test classique, le code est entièrement compile et valide. Il n'est pas possible d'écrire un code non valide, même si celui-ci n'est jamais exécuté (et sera supprimé par le compilateur). Par exemple si vous écrivez :

```
if (true) {
    std::cout << "hello" << std::endl;
} else {
    hello    // code invalide
}
```

Dans ce code, la clause `else` ne sera jamais exécutée et le compilateur pourrait supprimer cette partie du code. En pratique, cette partie est quand même compilée et doit être valide. Ce type de situation peut se rencontrer en méta-programmation avec des templates.

La syntaxe `constexpr-if` permet de ne pas évaluer le code qui n'est pas utilisé dans le test. Pour cela, il est nécessaire que l'expression booléenne soit évaluée à la compilation.

```
if constexpr (true) {
    std::cout << "hello" << std::endl;
} else {
    hello    // valide, ce code n'est pas compile
}
```

Initialisation dans la clause switch [C++17]

```
switch (INITIALISATION; EXPRESSION) ...
```

Chapitre précédent	Sommaire principal	Chapitre suivant