<C++>

Les pointeurs

14 | APR | 2020

Dorian.H Mekni

C++

Les pointeurs | déclaration -> C

Déclaration de pointeurs en C++ relativement standard
si une étude approfondi du C a été au préalable.

```cpp
#include <iostream>

int main()
{

    int x = 12;
    int *pExample{nullptr};

    pExample = &x;

    std::cout << *pExample << std::endl;

    return 0;

}
```

```
12
Program ended with exit code: 0
```

C++

Les pointeurs | allocation dynamique

voilà en C++ la syntaxe la plus standard,
voir ancienne pour l'allocation dynamique.

```cpp
#include <iostream>

int main()
{

    int *pExample{new int(12)}; // Le new Serait en C un malloc()

    std::cout << *pExample << std::endl;

    delete pExample;

    return 0;

}

```

```
12
Program ended with exit code: 0
```

C++

Les pointeurs | Allocation dynamique ++

Cette méthode permettait de remplacer les fameux malloc() héritée du C

```cpp
#include <iostream>

int main()
{

    int *pExample{new int(12)};
    int *tableau{new int[100]};

    std::cout << *pExample << std::endl;
    std::cout << *tableau << std::endl;

    delete[] tableau;
    delete pExample;
    // Cela permet de vider l'espace mémoire allouée

    return 0;

}
```

```
12
0
Program ended with exit code: 0
```

C++

Les pointeurs | Mémo

```
/*

 Le pointeur unique -> Un seul propriètaire ayant accès à une ressource. Une fois le pointeur
     détruit, la ressource est libérée de la mémoire.
 Le pointeur partagé -> plusieurs propriètaires peuvent accèder à la même ressource. Aussi
     longtemps que des éléments utilisent la ressource, cette dernière est mobilisée. Dès que
     les pointeurs sont détruits, la ressource est libérée.
 Le pointeur faible -> Il vérifie que notre ressource existe, si c'es tle car on pourra le
     transformé en pointeur partagé

 */
```

C++

Les pointeurs | <memory>

Méthode dans laquelle il n'y aura pas de fuite mémoire
et grâce à laquelle nous n'avons pas à nous occuper de libérer de l'espace mémoire.
Ne pas oublier d'ajouter l'entête #include<memory>

```
3  int main()
4  {
5
6      std::unique_ptr<int> ptOperator{std::make_unique<int>(12)};
7
8      std::cout << *ptOperator << std::endl;
9
10     return 0;
11
12 }
13
```

```
12
Program ended with exit code: 0
```

C++

Les pointeurs | auto

Toujours dans le même exemple,
mais cette fois en allégeant la syntaxe avec l'utilisation d' auto ->

```
3  int main()
4  {
5
6      auto ptOperator{std::make_unique<int>(12)};
7
8      std::cout << *ptOperator << std::endl;
9
10     return 0;
11
12 }
13
```

```
12
Program ended with exit code: 0
```

C++

Les pointeurs | [ ]

En revanche pour l'initialisation du tableau dans ce cas de figure, il nous faudra insérer les crochets pour que le compilateur comprenne qu'il ne s'agit pas d'une variable

```cpp
#include <iostream>
#include <memory>
int main()
{

    auto tableau{std::make_unique<int[]>(100)};

    return 0;

}

```

C++

Les pointeurs | pointeur partagé

Voilà la syntaxe pour l'initialisation d'un pointeur partagé

```cpp
#include <iostream>
#include <memory>
int main()
{

    std::shared_ptr<int> ptrNumber{std::make_shared<int>(12)};

    return 0;

}
```

C++

Les pointeurs | La conversion

Pour la conversion d'un pointeur unique à un pointeur brut,
on utilise la méthode .release() au moment d'initialiser le pointeur en question
sans oublier de l'effacer avec delete

```
1  #include <iostream>
2  #include <memory>
3  int main()
4  {
5
6      auto ptrNumber{std::make_unique<int>(12)};
7      int *pt = ptrNumber.release();
8
9      delete pt;
10     pt = nullptr;
11
12     return 0;
13
14 }
15
```

C++

->