<C++>

Les fonctions

12 | APR | 2020

Dorian.H Mekni

C++

Fonction basique

```cpp
#include <iostream>

int total(int x, int y)
{
     return x + y;

}
int main()

{
    auto n1{10};
    auto n2{19};

    std::cout << total(n1, n2) << std::endl;

    return 0;

}
```

```
29
Program ended with exit code: 0
```

C++

Fonction | &

```cpp
#include <iostream>

void change_number(int &n1) // & fera reference à la valeur contenur dans la fonction -> alt aux pointeurs
{
     n1 = 16;

}
int main()

{
    auto n1{0};

    std::cout << "Before : " << n1 << std::endl;
    change_number(n1);

    std::cout << "After : " << n1 << std::endl;

    return 0;

}
```

```
Before : 0
After : 16
Program ended with exit code: 0
```

C++

Fonction | Multiple

Il est possible de réécrire la même fonction mais avec des types différents et c'est le compilateur qui jugera laquelle prendre en compte comme c'est le cas dans notre exemple ->

```cpp
#include <iostream>

int total(int x, int y)
{
     return x + y;
}

double total(double x, double y)
{
    return x + y;
}


int main()

{
    auto n1{10.5};
    auto n2{19.7};

    std::cout << total(n1, n2) << std::endl;

    return 0;

}
```

```
30.2
Program ended with exit code: 0
```

C++

Fonction | Templates

Encore + court, il y a les templates qui allègent cette surcharge de fonctions et contiennent les différents elements de types à prendre en compte pour un retour intuitif ->
Voir ex pour les int et pour les double :

```cpp
#include <iostream>

template<typename T> // Le trype sera stoqué dans le T
T total(T x, T y)
{
    return x + y;
}

int main()

{
    auto n1{10};
    auto n2{19};

    std::cout << total(n1, n2) << std::endl;

    return 0;

}
```

```
29
Program ended with exit code: 0
```

```cpp
#include <iostream>

template<typename T> // Le trype sera stoqué dans le T
T total(T x, T y)
{
    return x + y;
}

int main()

{
    auto n1{10.5};
    auto n2{19.7};

    std::cout << total(n1, n2) << std::endl;

    return 0;

}
```

```
30.2
Program ended with exit code: 0
```

C++

Fonction | <...>

Dans les templates au moment de l'affichage via std::cout,
on peut faire appel au type que l'on désire voir s'afficher
et via l'ajout <> en y plaçant le type voulu.

```cpp
#include <iostream>

template<typename T> // Le trype sera stoqué dans le T
T total(T x, T y)
{
    return x + y;
}

int main()

{
    auto n1{10.1};
    auto n2{19.4};

    std::cout << total<int>(n1, n2) << std::endl;
    //Pour forcer le type <int> au moment de la compilation
    return 0;

}

```

```
29
Program ended with exit code: 0
```

C++

Fonction | lambda

```cpp
#include <iostream>


int main()

{

    auto total = [](auto x, auto y){return x + y;};

    std::cout << total(3, 2) << std::endl;

    return 0;

}
```

```
5
Program ended with exit code: 0
```

C++

Fonction | lambda &

une fonction lambda en C++ peut également contenir |
[&](auto x, auto y){return x + y + number1 + number2;};

```cpp
#include <iostream>

int main()

{

    int number1 = 10;
    int number2 = 5;

    auto total = [&](auto x, auto y){return x + y + number1 + number2;};

    std::cout << total(3, 2) << std::endl;

    return 0;

}
```

```
20
Program ended with exit code: 0
```

C++

Fonction | lambda -> Mémo

```
/*

    FONCTION LAMBDA

    [&] -> Les variables en référence
    [=] -> Les variables par valeur

*/
```

C++

->