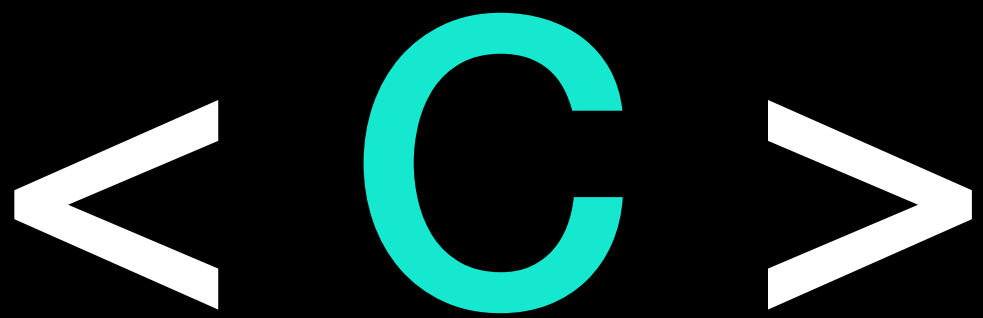


La récursivité

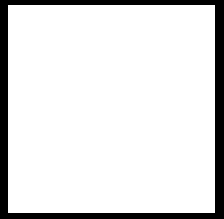
Dorian.H Mekni

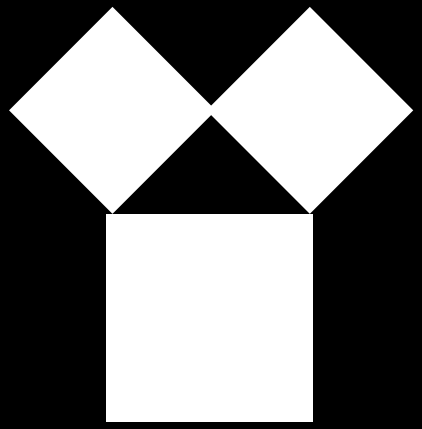
30 | MAR | 2020

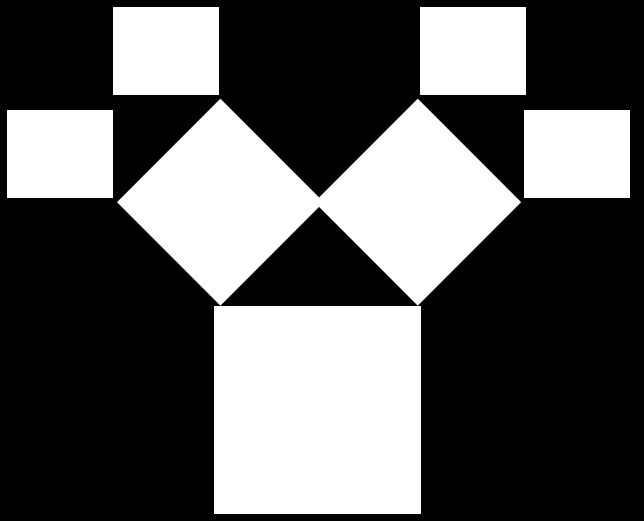
La récursivité

- La récursivité est un problème algorithmique.
- il s'agit de programmes ou de fonctions d'un programme qui ont la faculté de s'appeler eux-mêmes.
- La récursivité est une manière minimaliste de résoudre des problèmes algorithmiques.
- Quand on a pas de récursivité on parle d'iteration.
- Il y a deux types de fonctions récursives: les fonctions récursives et les fonctions récursives terminales.

La récursivité







La récursivité

- C’est une fonction qui se rappelle elle-même

```c
void recursionFunction(void)
{
// C'est une fonction qui se rappelle elle-même
printf("Recursion is the key !. \n");
recursionFunction();
}
```

```c
int main(void)
{

recursionFunction();
return 0;
}
```

Récursivité sous condition

```
void recursionFunction(int i)
                {

                if(i == 10)
                   return;

printf("Recursion under conditions !.
                \n");
            recursionFunction(i+1);
                }

             int main(void)
                {

            recursionFunction(0);
                return 0;

                }
```

```
Recursion under conditions !.
Recursion under conditions !.
Recursion under conditions !.
Recursion under conditions !.
Recursion under conditions !.
Recursion under conditions !.
Recursion under conditions !.
Recursion under conditions !.
Recursion under conditions !.
Recursion under conditions !.
Program ended with exit code: 0
```

La récursivité intègre une condition afin de contraindre la boucle à s'arrêter une la repetition atteinte au bout de 10 fois.

récursive | itérative

- Une fonction iterative sera plus longue qu'une fonction recursive dans son architecture et donc écriture.
- Néanmoins une fonction recursive sera plus demandante car elle toquera plus d'information donc plus de place en mémoire.
- Test de fonction en mode récursif et implementation en itératif pour alléger le code

Suite factorielle

```
/*

suite factorielle de 6 -> 6 * 5 * 4 * 3 * 2 *
1 = 720

suite factorielle de 4 -> 4 * 3 * 2 * 1 =
24

*/
```

Fonction récursive terminale

- Celle-ci crée un empilage mais sans remontée
- Cela veut dire qu'une fois empilé nous aurons un retour d'affichage finale de notre résultat.

```c
#include <stdio.h>
#include <stdlib.h>

unsigned long factoriel(int a)
{

    if(a < 0)
       exit(EXIT_FAILURE);

    if(a == 0 || a == 1)
       return 1L;

    return a * factoriel(a - 1);
}

int main(void)
{

    printf("%lu\n", factoriel(30));
    return 0;
}

```

```
9682165104862298112
Program ended with exit code: 0
```

Suite de Fibonacci

Le résultat du 3ème chiffre - en partant de la gauche est la somme des deux termes précédents et ainsi de suite…

**/*
La suite fibonacci
<——————>
1 1 2 3 5 8 13 21
*/**

Suite de Fibonacci | A

Ici une fonction plus courte <2 lignes> mais plus lourde puisque la récursivité à savoir le rappel de la fonction à l'intérieur de cette même fonction se fait à 2 reprises

```c
int fibonacci(int fibo)
{

    if(fibo < 2) return 1;
    return fibonacci(fibo - 1) + fibonacci(fibo - 2);

}

int main(void)
{
    printf("%d\n", fibonacci(10));
    return 0;
}
```

```
89
Program ended with exit code: 0
```

Suite de Fibonacci | B

Fonction Fibonacci plus longue mais plus performante puisque sans récursivité multiple

```c
#include <stdio.h>
#include <stdlib.h>

int fibonacci(int fibo)
{
    int i = 1, element0 = 1, element1 = 1, temp;
    if(fibo < 2) return 1;
    while(i < fibo)
    {
        temp = element0 + element1;
        element0 = element1;
        element1 = temp;
        i++;
    }
    return(element1);
}

int main(void)
{
    printf("%d\n", fibonacci(10));
    return 0;
}

```

```
89
Program ended with exit code: 0
```