



Institut Supérieur d'Informatique,
de Modélisation et de leurs Applications

Rapport de management projet ZZ3

Participation à la journée porte ouverte Démo + Poster

Filière :

F2 : Génie Logiciel et Systèmes Informatiques
F5 : Réseaux et sécurité informatique

Présenté par : **Paul Chapuis et Thomas TAMAGNAUD**

Responsable ISIMA : Patrice LAURENCOT

Le samedi 15 février 2025



INTRODUCTION

Notre projet était initialement de proposer un cours d'initiation du langage de programmation Rust à l'ISIMA, mais après discussion avec M.LAURENCOT qui était convaincu qu'il n'y aurait pas grand monde d'intéressé, celui-ci nous a indiqué de partir sur un modèle démo à la jpo + poster.

CONTEXTE

Notre projet de management consiste à présenter brièvement les différences entre les langages de programmation à des étudiants intéressés ou intrigués par l'informatique, lors de la JPO 2025 de l'ISIMA le samedi 15 février 2025, afin qu'ils puissent continuer dans cette voie si ils le souhaitent.

Nous venons donc apporter nos 2 points de vue différents pour la jpo, nous venons de 2 filières différentes, l'un a fait la prépa intégré ISIMA, l'autre un IUT.

Nous souhaitons faire un parallèle avec des bouts de codes écrit dans des langages étudiés au collège/lycée comme Scratch ou Python, et des codes plus avancé comme du C++ ou Rust, afin de montrer que les concepts mis en place sont relativement similaires, même si la syntaxe de ces langages est un peu différente.

DÉROULEMENT

Avant la JPO, nous avons préparé un github pour héberger les différents codes, ainsi que les modèles du poster :

https://github.com/Thomas-Mewily/jpo_isima_2025_code_showcase

Celui-ci a été imprimé à Corep, une agence spécialisée dans l'impression proche de Jaude <https://www.corep.fr/agence/corep-clermont-ferrand/> .

Nous avons programmé les algorithmes suivants pour la Journée Portes Ouvertes (JPO) 2025 de [l'ISIMA](#)

- [Code de César](#)
- [Factorielle](#)
- [Somme d'entier](#)

... écrits dans différents langages de programmation :

- [Scratch](#)
- [Python](#) (juste factorielle...)
- [Rust](#)
- [C++](#) (juste factorielle...)

L'idée est de montrer les différences entre des langages comme [Scratch](#), qui utilisent l'approche visuelle avec des blocs, à des langages de programmation écrits en lignes de code tel que Python ou du C++.

Nous avons été ensuite invités à l'installation du matériel et des différents stands dans le bâtiment A, nous avons donc rencontré Mme Ferrand, qui était au courant de notre venue suite aux différentes prises de contact que nous avons effectué au préalable, nous avons donc installé les tables, nappes, affiches, et rallonges électriques nécessaires pour notre stand et quelques autres stands à proximité.

Le jour J, notre stand était prêt, nous avons accueilli les premiers visiteurs que nous avons conseillés, ils se présentaient majoritairement dans le but de recueillir notre point de vue en tant qu'étudiant, comme nous sommes de deux filières et deux parcours d'admissions différents, nous avons pu fournir des réponses pertinentes aux visiteurs.

Ceci fait, à la fin de l'événement, nous avons aidé à la désinstallation des stands.

ANNEXES

Toute les annexes sont également disponible sur le git :

https://github.com/Thomas-Mewily/jpo_isima_2025_code_showcase

Exemple de bout de code juste pour la factorielle :

Python :

```
def factorielle_recursive(n : int) -> int:
    match n:
        case 0 | 1: return 1
        case _: return n * factorielle_recursive(n-1)

def factorielle_recursive_avec_si(n : int) -> int:
    if (n == 0 or n == 1):
        return 1
    return n * factorielle_recursive_avec_si(n-1)

def factorielle_iterative(n : int) -> int:
    resultat = 1
    while n != 0:
        resultat *= n
        n -= 1
    return resultat

for i in range(10):
    print(f"factorielle({i: >2}) = {factorielle_recursive(i): >7} (recursif) {factorielle_iterative(i): <7} (iteratif)")
print()
```

Rust :

```
pub const fn factorielle_recursive(n : u32) -> u32
{
    match n
    {
        0 | 1 => 1,
        _ => n * factorielle_recursive(n-1)
    }
}

pub const fn factorielle_recursive_avec_si(n : u32) -> u32
{
    if n == 0 || n == 1 { 1 }
    else { n * factorielle_recursive(n-1) }
}

pub const fn factorielle_iterative(mut n : u32) -> u32
{
    let mut resultat = 1;
    while n != 0
    {
        resultat *= n;
        n -= 1;
    }
    resultat
}

pub fn factorielle_exemple()
{
    for i in 0..10
    {
        println!("factorielle({: >2}) = {: >7} (recursif) {: <7}
        (iteratif)", i, factorielle_recursive(i),
        factorielle_iterative(i))
    }
    println!();
}
```

C++ :

```
#include <iostream>
#include <iomanip>

int factorielle_recursive(int n)
{
    switch (n)
    {
        case 0: case 1:
            return 1;
        default:
            return n * factorielle_recursive(n - 1);
    }
}

int factorielle_recursive_avec_si(int n)
{
    if (n == 0 || n == 1)
        return 1;
    return n * factorielle_recursive_avec_si(n - 1);
}

int factorielle_iterative(int n)
{
    int resultat = 1;
    while (n != 0)
    {
        resultat *= n;
        n -= 1;
    }
    return resultat;
}

int main() {
    for (int i = 0; i < 10; ++i)
    {
        std::cout << "factorielle("
                << std::setw(2) << i << ") = "
                << std::setw(7) << std::right <<
factorielle_recursive(i) << " (recursif) "
                << std::setw(7) << std::left <<
```

```

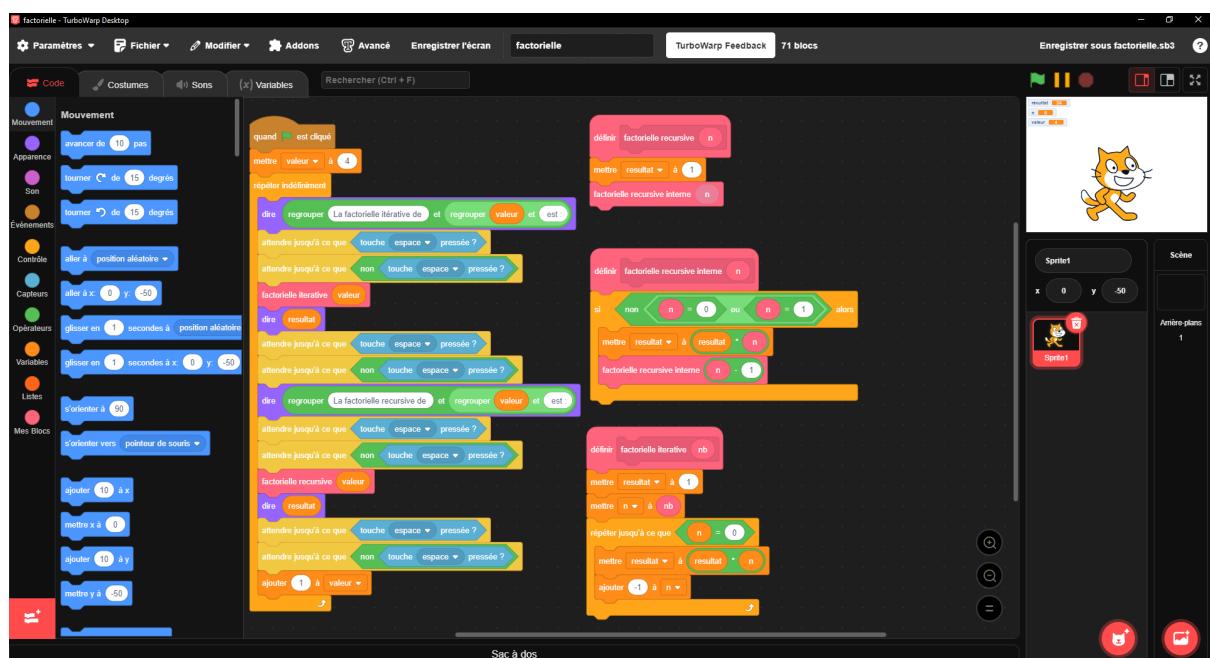
factorielle_iterative(i) << " (iteratif)"
    << std::endl;
}

return 0;
}

/*
g++ -o cpp/factorielle cpp/factorielle.cpp
./cpp/factorielle
*/

```

Scratch :



A chaque la sortie affiché est (modulo Scratch où l'affichage est un peu différent)

<code>factorielle(0) =</code>	<code>1 (recursif) 1</code>	<code>(iteratif)</code>
<code>factorielle(1) =</code>	<code>1 (recursif) 1</code>	<code>(iteratif)</code>
<code>factorielle(2) =</code>	<code>2 (recursif) 2</code>	<code>(iteratif)</code>
<code>factorielle(3) =</code>	<code>6 (recursif) 6</code>	<code>(iteratif)</code>
<code>factorielle(4) =</code>	<code>24 (recursif) 24</code>	<code>(iteratif)</code>
<code>factorielle(5) =</code>	<code>120 (recursif) 120</code>	<code>(iteratif)</code>
<code>factorielle(6) =</code>	<code>720 (recursif) 720</code>	<code>(iteratif)</code>
<code>factorielle(7) =</code>	<code>5040 (recursif) 5040</code>	<code>(iteratif)</code>
<code>factorielle(8) =</code>	<code>40320 (recursif) 40320</code>	<code>(iteratif)</code>
<code>factorielle(9) =</code>	<code>362880 (recursif) 362880</code>	<code>(iteratif)</code>

Le poster de la JPO, imprimé en format A0 (841 x 1189 mm) pour environ 38 euros :



Ouvrez-vous à l'informatique !



```
fn factorielle(mut n : u32) -> u32
{
    let mut resultat: u32 = 1;
    while n != 0
    {
        resultat *= n;
        n -= 1;
    }
    resultat
}
```



```
factorielle_iterative(int):
    push rbp
    mov rbp, rsp
    mov DWORD PTR [rbp-20], edi
    mov DWORD PTR [rbp-4], 1
    jmp .L2
.L3:
    mov eax, DWORD PTR [rbp-4]
    imul eax, DWORD PTR [rbp-20]
    mov DWORD PTR [rbp-4], eax
    sub DWORD PTR [rbp-20], 1
.L2:
    cmp DWORD PTR [rbp-20], 0
    jne .L3
    mov eax, DWORD PTR [rbp-4]
    pop rbp
    ret
```

x86-64 gcc 14.2



```
def factorielle(n : int) -> int:
    resultat = 1
    while n != 0:
        resultat *= n
        n -= 1
    return resultat
```



Python



```
int factorielle(int n)
{
    int resultat = 1;
    while (n != 0)
    {
        resultat *= n;
        n--;
    }
    return resultat;
```

C++