POLYTECHNIQUE
MONTRÉAL

INF1600
Architecture des micro-ordinateurs

TP3

Groupe 02 (B2)

Soumis par:
Charles de Lafontaine - **2076524**
Geneviève Pelletier-Mc Duff - **2088742**

Le 24 mars 2021

Barème de correction

TP 2		**/4,00**
Exercice 1 :		
	`serie_s_iter.s`	…/0,25
	`serie_s_rec.s`	…/0,5
	Discussion 2.3	…/0,25
Exercice 2 :		
	`admisasm.s`	… /1,00
	`moyenneasm.s`	… /1,00
Partie 2		
	`admisasm1.s`	… /1,00

1.3 - Discussion des avantages et inconvénients de l'approche itérative vis-à-vis l'approche récursive

Nous allons comparer l'implémentation d'une fonction itérative à celle récursive sous différents aspects, soit la longueur du code, la facilité d'implémentation et la performance.

Tout d'abord, pour ce qui est de la longueur du code, nous pouvons remarquer qu'une fonction récursive s'écrit, en général, en un nombre inférieur de lignes de code, comparativement à une fonction itérative. Par contre, moins de lignes de code ne signifient pas nécessairement une meilleure performance.

Pour tout dire, une fonction itérative est généralement plus simple et facile à implémenter que son homologue récursif. En effet, en assembleur, pour implémenter une fonction récursive, nous devons passer plus de temps pour réfléchir aux accès à la mémoire, éviter les erreurs de segmentation et nous assurer du bon fonctionnement du programme. Ainsi, si nous avons des contraintes strictes de temps et que nous cherchons une méthode plus efficace, un programmeur aurait intérêt à préférer le type itératif.

Pour ce qui est de la performance, suite à des recherches, nous avons découvert qu'une fonction récursive est généralement moins performante, moins efficace et demande un plus long temps d'exécution qu'une fonction itérative [1]. Aussi, il faut souvent plus d'accès en mémoire en utilisant une version récursive étant donné que la fonction s'appelle elle-même et a besoin d'accéder plus fréquemment au contenu de la pile.

Par ailleurs, nous avons écrit un petit script en Python qui calcule la différence de temps entre la manière itérative et celle récursive pour calculer les 500 premières factorielles (voir **Figure 1**).

```python
from time import time

def factorial_iter(n: int) -> int:
    f = 1
    for i in range(1, n + 1):
        f *= i
    return f

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

temps1 = time()
for i in range(499):
    factorial_iter(i)
print(time() - temps1, " secondes se sont écoulés pour factorial_iter")

print("VERSUS")

temps2 = time()
for i in range(499):
    factorial_cur(i)
print(time() - temps2, " secondes se sont écoulés pour factorial_cur\n")
print("La manière itérative est donc plus rapide de ", ((temps2 - temps1) * 1000), " ms !")
```

Figure 1. Script écrit en Python qui calcule les 500 premières factorielles de manière itérative et récursive et qui les compare.

À l'exécution de ce script, nous voyons clairement que la manière itérative est à privilégier, puisque cette dernière est environ 12 ms plus rapide que son homologue récursive (voir **Figure 2**). Plus les calculs sont importants, plus cette différence de temps est remarquable.

```
0.011968135833740234  secondes se sont écoulés pour factorial_iter
VERSUS
0.020943403244018555  secondes se sont écoulés pour factorial_cur

La manière itérative est donc plus rapide de  11.968135833740234  ms !
```

Figure 2. Affichage du script lors de l'exécution.

Bref, en considérant les différents aspects présentés soit le nombre de lignes de code, la facilité d'implémentation et la performance, nous remarquons une préférence pour une fonction de type itérative dans un contexte d'assembleur. Par contre, selon le contexte et nos besoins, une fonction récursive peut s'avérer utile et une piste de solution.

Référence :

[1] (2005) Recursion. In: Introduction to Assembly Language Programming. Texts in Computer Science. Springer, New York, NY. https://doi.org/10.1007/0-387-27155-4_16