TD 6 - Programmation Avancée

Hugo Bolloré[1]

[1]hugo.bollore@uvsq.fr

Dans ce TD nous utiliserons plusieurs langages, le C, le Javascript et le Python.

Pour le langage Javascript, pour rappel vous avez 2 choix :

1. Dans votre navigateur, allez sur le site https://playcode.io/javascript
2. Téléchargez le package *nodejs*, écrivez le code dans un fichier script_name.js et le lancer avec :

   ```
   node script_name.js
   ```

Pour langage Python :

1. Dans votre navigateur, allez sur le site https://www.online-python.com/

Exercice 6.1 Iterative et Recursive

Question 6.1.1 Boucle itérative

Développer en C, une fonction 'facto' avec une boucle itérative qui prendra en entrée un entier n positif et retournera la factorielle de n.

Question 6.1.2 Boucle récursive

Transformer la fonction que vous venez d'écrire pour qu'elle utilise une récursion.

Question 6.1.3 Optimisation

Il y a-t-il une optimisation possible ? Si oui, modifier votre code et appliquer cette optimisation.

Exercice 6.2 Impératif vs Fonctionnel

Question 6.2.1 Langage impératif

Écrivez en Python, un code impératif qui effectuera les actions suivantes sur une liste d'entier:

- Effectue le carré sur tout les élements de la liste,
- Récupère tous les nombres pairs de la nouvelle liste,
- Effectue une reduction sur la nouvelle liste.

```python
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# CODE HERE

print("La somme des carrés pairs est :", sum)
```

Question 6.2.2 Fonction reduce

Écrivez en Python, une fonction *reduce* qui prend en entrée 3 arguments :

- func : une fonction qui sera appliqué a chaque élements de l'iterable,
- iterable : un iterable (list, tuple, etc.),

- initial : un entier qui sera la valeur initial de la reduction.

La fonction *reduce* effectuera la reduction d'un iterable et renverra donc la somme de tous les élements + la valeur initiale.

Question 6.2.3 Langage fonctionnel

Transformer le code impératif de la première question en code fonctionnel. Pour cela, vous ne devriez pas utiliser de boucle et vous aurez besoin des fonctions *map*, *filter* et de la fonction *reduce* que vous venez d'écrire.

Bonus : Utilisez des fonctions anonymes dans votre code.

Exercice 6.3 Cast

En utilisant le fichier *data.set* qui vous sera fourni, calculez la différence entre les 50 valeurs des tableaux *uintVals* et *intVals*, stockez celles-ci dans des *long int*, et calculez et affichez la moyenne de toutes les différences qui sont inférieures à 888.

Pour pouvoir lire les données binaires contenues dans le fichier *data.set* vous devrez partir du code suivant (le fichier data.set devra être copié dans le même répertoire que votre programme compilé) :

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

#define NB_VALS 50

void main()
{
    FILE* fp;
    unsigned int uintVals[NB_VALS];
    int intVals[NB_VALS];
    int returnValue = 0;

    fp = fopen("data.set", "rb");
    if (fp == NULL) {
        fprintf(stderr, "Cannot open file data.set\n");
        exit(-1);
    }

    returnValue = fread(uintVals, sizeof(unsigned int), NB_VALS, fp);
    if (returnValue != NB_VALS) {
        fprintf(stderr, "Cannot read %d blocks in file data.set\n", NB_VALS);
        exit(-1);
    }

    returnValue = fread(intVals, sizeof(int), NB_VALS, fp);
    if (returnValue != NB_VALS) {
        fprintf(stderr, "Cannot read %d blocks in file data.set\n", NB_VALS);
        exit(-1);
    }
}
```

Vous devriez obtenir une moyenne de -453110824.230769.

Exercice 6.4 Closure

```
function factorial(n) {
    if (n <= 1) return 1;
    let result = 1;
    for (let i = 2; i <= n; i++) {
        result *= i;
    }
    return result;
}
```

Refactorisez la fonction de calcul de la factorielle, ci dessus, en utilisant une closure pour mémoriser les résultats précédemment calculés. Ainsi, lorsque la même valeur est demandée, vous pourrez renvoyer le résultat directement depuis le cache sans avoir à refaire le calcul.

- Écrivez une fonction create_optimized_factorial qui créé une fonction de calcul de la factorielle.
- La fonction retournée par create_optimized_factorial doit mémoriser les résultats précédents dans un cache (un tableau dont les indexes seront les valeurs de n et cache[n] = factorial(n)).
- Si la fonction est appelée avec un nombre pour lequel la factorielle a déjà été calculée, elle doit renvoyer directement le résultat du cache et afficher un message disant que le resultat vient du cache.
- Si la fonction est appelée avec un nouveau nombre, elle doit calculer la factorielle, la stocker dans le cache, et renvoyer le résultat.

```
const factorial = create_optimized_factorial();

console.log(factorial(6));  // Affiche "720"
console.log(factorial(5));  // Affiche "120"
console.log(factorial(5));  // Affiche "Utilisation du cache : 120"
console.log(factorial(7));  // Affiche "5040"
console.log(factorial(6));  // Affiche "Utilisation du cache : 720"
```

Bonus : Si ce n'est pas déjà le cas, modifiez votre code pour que soit mis en cache les résultats intermédiaires du calcul d'une factorielle. Votre affichage devrait être celui-ci.

```
console.log(factorial(6));  // Affiche "720"
console.log(factorial(5));  // Affiche "Utilisation du cache : 120"
console.log(factorial(5));  // Affiche "Utilisation du cache : 120"
console.log(factorial(7));  // Affiche "5040"
console.log(factorial(6));  // Affiche "Utilisation du cache : 720"
```