Feuille de TP N°01 – Rappel et complexité

Exercice 01 : Un petit rappel …

L'objectif de cet exercice est de faire un petit rappel sur les bases de la programmation en C (conditions, boucles, enregistrements, tableaux, listes chaînées, fonctions, manipulation de données, etc). Pour ce faire, nous allons travailler sur la séparation des éléments pairs et impairs d'un tableau d'entiers, les trier puis les stocker dans deux listes chaînées différentes (une pour les pairs et une pour les impairs).

1. Déclarer la structure ***Liste*** qui permettra de créer une liste chaînée d'entiers. Cette structure contiendra un entier ***val*** et un pointeur vers une autre structure ***Liste*** nommé ***suiv***.
2. Ecrire une fonction qui prend en entrée un tableau d'entiers et sa taille et qui donne comme sortie deux listes d'entiers, une contenant les éléments pairs, et l'autre les éléments impairs du tableau. Les deux listes contiennent les éléments dans un ordre croissant.
3. Ecrire une fonction ***afficher_liste*** qui affiche les éléments d'une liste chaînée passée comme paramètre.
4. Ecrire un programme principal qui demande à l'utilisateur de remplir un tableau d'entiers, puis affiche les éléments pairs et impairs du tableau dans un ordre croissant (utiliser les deux fonctions précédentes).

Indication : rappelez-vous de la fonction permettant l'insertion dans une liste triée.

Exercice 02 : Mesurer le temps d'exécution ?

Le but de cet exercice est d'apprendre comment mesurer le temps d'exécution d'une partie de votre programme C. Cette mesure de temps est utile pour comparer les performances de différents algorithmes, comparer les plusieurs implémentations d'un même algorithme, mesurer l'impact d'une optimisation, ou de s'assurer que votre programme ne dépasse pas un temps d'exécution maximal spécifié.

Pour mesurer le temps d'exécution d'une partie de votre programme, vous pouvez utiliser la fonction ***clock*** de la bibliothèque ***'time.h'***. Cette fonction retourne le nombre de cycles d'horloge écoulés depuis le lancement du programme. Il faut donc appeler cette fonction deux fois :

1. Une fois avant le début de la partie du programme dont on veut mesurer le temps d'exécution.
2. Une fois après la fin de cette partie.

La différence entre les deux valeurs retournées par la fonction ***clock*** est le nombre de cycles d'horloge écoulés pendant l'exécution de la partie du programme.

Pour convertir ce nombre de cycles d'horloge en temps (en secondes), il faut diviser ce nombre par la fréquence du processeur. Par exemple, un processeur cadencé à une fréquence de 4.5 GHz effectue 4.5 milliards de cycles d'horloge par seconde. Pour convertir le nombre de cycles d'horloge en temps, il faut donc diviser le nombre de cycles d'horloge par 4.5 milliards. Vu que la fréquence du processeur peut varier, il est préférable d'utiliser la constante ***CLOCKS_PER_SEC*** de la bibliothèque ***'time.h'*** pour obtenir la fréquence du processeur,

plutôt que d'utiliser une valeur fixe. Cela permet de s'assurer que le même programme s'exécutera correctement sur des processeurs différents.

Soit le code C suivant qui calcule la suite de Fibonacci en utilisant une fonction récursive et une autre itérative :

```
#include <stdio.h>
#include <stdlib.h>
int fibo_rec(int n) {
    if (n <= 1) return n;
    return fibo_rec(n - 1) + fibo_rec(n - 2);
}
int fibo_iter(int n) {
    int a = 0;
    int b = 1;
    int c;
    for (int i = 0; i < n; i++) {
        c = a + b;
        a = b;
        b = c;
    }
    return a;
}
int main() {
    int n;
    scanf(
    printf("fibo_rec(%d) = %d\n", n, fibo_rec(n));
    printf("fibo_iter(%d) = %d\n", n, fibo_iter(n));
    return 0;
}
```

1. Modifier ce code pour mesurer le temps d'exécution de la fonction ***fibo_rec*** et de la fonction ***fibo_iter*** et afficher ces temps en seconde.
2. Tester les deux fonctions pour calculer l'image des valeurs suivantes : 10, 20, 30, 40.
3. Qu'observez-vous ? Expliquez ?

Exercice 03 : Pourquoi réduire la complexité ?

Dans cet exercice, nous allons écrire calculer la somme des n premiers entiers positifs, tel que n est donné par l'utilisateur. Nous allons écrire deux versions de la fonction qui calcule cette somme

1. La première version naïve, qui utilise une boucle for et qui calcule la somme en additionnant les entiers les uns après les autres (voir algorithme 1.2 sur le support de cours).
2. La deuxième version, qui utilise la formule mathématique de la somme des n premiers termes d'une suite arithmétique (voir algorithme 1.3 sur le support de cours).

Implémenter les deux fonctions et comparer leurs temps d'exécutions pour les valeurs de ***n*** suivantes : 1000, 1000000, 1000000000.

Indication : utiliser le type ***long long*** pour éviter les erreurs de dépassement de capacité.