Cours de programmation séquentielle

Récursivité

1 Buts

— Les Makefile.
— La récursivité
— La récursivit
— La récursivi
— La récursiv
— La récursi
— La récurs
— La récur
— La récu
— La réc
— La ré
— La r

2 Préambule

Comme vous l'aurez compris, le but de cette petite série d'exercices est de pratiquer la **récursivité**. Elle est séparée en deux parties. Dans la première, dite de *compréhension*, il s'agit de bien comprendre dans quel ordre les appels récursifs sont effectués. Dans la seconde, dite de *programmation*, il faut résoudre un certain nombre d'exercices de façon **récursive** uniquement.

Cette série n'est pas notée. Comme pour le premier semestre, vous pouvez obtenir un bonus de `0.5` sur la note de l'examen en rendant 2/3 des exercices non-notés du semestre. Vous devez mettre le lien de votre repo `git` dans le wiki correspondant sur `Cyberlearn`. **Ne nous rajoutez pas comme reporter sous peine de sanctions!** Vous pouvez à choix laisser le repo public ou alors le mettre en interne. Pour la partie de compréhension écrivez vos réponses dans un fichier `ANSWERS.md` et placez le dans votre repo `git`.

3 Exercices de compréhension

Exercice de compréhension 1 *(The final countdown)*

```c
int countdown(int i) {
    printf("%d ", i);
```

```
    return countdown(i);
}
```

Si nous appelons `countdown(5)`, combien d'appels à la fonction `countdown` seront effectués?

Que fait cette fonction? Qu'affiche-t-elle?

Exercice de compréhension 2 *(Sommons-nous dans les bois)*

```
int sum(int i) {
    return i + sum(i - 1);
}
```

Que fait cette fonction?

Si nous appelons `sum(5)`, combien d'appels à la fonction `sum` seront effectués?

Exercice de compréhension 3 *(Nombre d'appels)*

Soit la fonction suivante:

```
void print(int i) {
    if (i < 1) {
        return;
    }
    print(i - 2);
    print(i - 4);
    printf("%d\n", i);
}
```

Si nous appelons `print(5)` combien d'appels à `print` seront effectués avant que la fonction se termine? Qu'est-ce que ce programme va afficher?

---

4 Exercices de programmation

Pour chacune des fonctionnalités ci-dessous, essayez d'écrire des tests **avant** (à l'aide de `<assert.h>`) de les implémenter. Utilisez une cible `tests` dans votre `Makefile` pour exécuter les tests "automatiquement". Il faudra aussi un `main` qui illustre l'utilisation des fonctions.

Dans ce qui suit il y a des exemples d'utilisation de fonctions, mais il faut des fois changer les signatures.

Exercice 1 *(Affichage)*

Afficher les entiers plus grands de `0` à `N-1`.

Exemple:

```
show(10);

// Affiche: 0 1 2 3 4 5 6 7 8 9
```

Exercice 2 *(Somme)*

Calculer la somme des entiers positifs de `0` à `N`.

Exemple:

```
sum(10);

// Retourne: 55
// Le calul effectué est :
//  0 + 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10
```

Exercice 3 *(Somme d'éléments)*

Calculer la somme des entiers se trouvant dans un tableau.

Exemple:

```
sum({1, 5, 3});

// Retourne: 9
// Le calul effectué est bien : 1 + 3 + 5
```

Exercice 4 *(Compter les digits)*

Compter le nombre de digits d'un nombre.

Exemple:

```
count_digits(8345);

// Retourne: 4
```

Exercice 5 *(Sommer les digits)*

Sommer les digits d'un nombre.

Exemple:

```
sum_digits(8345);

// Retourne: 20
// Le calcul est: 8 + 3 + 4 + 5
```

Exercice 6 *(Palindrome)*

Déterminer si une chaîne de caractères est un palindrome (il s'écrit de façon identique de gauche à droite et de droite à gauche: https://fr.wikipedia.org/wiki/Palindrome). Par simplicité on considérera que les lettres majuscules et accentuées sont différentes des minuscules.

Exemple:

```
is_palindrome("ABBA");

// Retourne: true

is_palindrome("albuchef");

// Retourne: false
```

---

5 Exercice bonus

Pour ceux · celles qui le souhaitent voilà également un exercice supplémentaire un peu plus complexe.

Exercice bonus 1 *(Tours de Hanoi)*

Implémenter une solution récursive du jeu de des tours de Hanoi https://fr.wikipedia.org/wiki/Tours_de_Hano%C3%AF.
