CC3 - sujet 1

Info1.Algo1 - 2022-2023 Semestre Impair

- Pour chaque exercice vous devez déposer un fichier correspondant à votre travail.
- Vous ne devez pas modifier le nom du fichier.
- Un fichier qui ne compile pas, ou qui ne passe pas les fonctions de tests proposées dans le sujet ne sera pas corrigé et entraînera la note de 0 à l'exercice, sans aucune investigation supplémentaire.
- Il est naturellement interdit de modifier la fonction de tests qui vous est proposée, vous devez être particulièrement attentif à nommer correctement vos fonctions et à respecter les paramètres attendus.

- Le passage des différents tests proposés avec le sujet n'est en aucune façon une garantie de la bonne correction de votre travail, des tests complémentaires seront passés à posteriori.

Exercice 1

> **Rappel :** Étant donnés deux entiers positifs x et y (y étant non nul), le quotient q et le reste r de la division euclidienne de x par y sont les uniques entiers tels que $x = q \cdot y + r$, avec $r \in [0; y[$

Dans le fichier **ex01.py** est définie la fonction `division_euclidienne` qui accepte en paramètre deux entiers `x` et `y` et retourne les entiers `q` et `r`. L'objectif de cet exercice est d'instrumenter le code fourni avec les `assert` nécessaires à la vérification de :

- la **pré-condition**.
- la **post-condition**.
- l'**invariant**
- le **variant**.

Attention Pour permettre l'analyse de votre code nous sommes contraints de vous imposer de n'écrire les asserts que sur **une** seule ligne.

1) Écrire dans le corps de la fonction `division_euclidienne` les assertions de **pré-condition** et de **post-condition** correspondant à la définition rappelée ci-dessus.

2) Écrire dans le corps de la fonction `division_euclidienne` les 2 assertions pour la vérification de l'**invariant** *(on pourra s'aider de la formulation de la*

post-condition et de la condition de boucle afin d'écrire un invariant pertinent).

3) Déterminer un variant (positif ou nul) pour la boucle de la fonction `division_euclidienne`. Ajouter les 4 lignes de codes permettant la vérification expérimentale de la terminaison par contrôle de la décroissance et de la non-négativité du variant.

Il est **indispensable** que le code proposé continue à passer la fonction de test sans erreurs. Dans l'hypothèse contraire, il vous faudra commenter les assertions qui posent problème, et la question correspondante sera considérée comme non traitée. Une fonction qui ne passe plus les tests entraînerait un 0 à l'exercice.

Exercice 2

Une matrice carrée est une matrice de nombres ayant le même nombre `n` de lignes et de colonnes. On dit qu'une matrice carrée est symétrique lorsque les valeurs sont identiques de part et d'autre de sa diagonale principale.

Exemples :

Matrice symétrique :

$$\begin{pmatrix} 1 & 5 & 3 \\ 5 & 6 & 10 \\ 3 & 10 & 12 \end{pmatrix}$$

Matrice non-symétrique :

$$\begin{pmatrix} 1 & 5 & 3 \\ 5 & 6 & 8 \\ 3 & 10 & 12 \end{pmatrix}$$

Dans le fichier **ex02.py**, compléter la fonction `est_symetrique` qui retourne `True` si la matrice donnée en paramètre est symétrique et `False` sinon.

Exercice 3

Dans le fichier **ex03.py**, compléter le corps de la fonction récursive `somme_elements_pairs` qui accepte en paramètre une `liste` et retourne la somme de ses éléments pairs.

On ne manipulera les listes qu'au moyen des 5 fonctions d'interface fournies dans le sujet.

Exemple : `somme_elements_pairs((7,(4,(-2,(3,None)))))` retourne `4 + (-2) = 2`.

Exercice 4

Dans le fichier **ex04.py**, écrire la fonction **récursive** `sont_chiffres_croissants` qui accepte en paramètre un entier positif ou nul `n` et retourne le booléen `True` si les chiffres de `n` (représentation en base 10) sont en ordre croissant (au sens large), et `False` sinon.

Exemple :

- `sont_chiffres_croissants(10)` retourne `False` ($1 < 0$).
- `sont_chiffres_croissants(123)` retourne `True` ($1 < 2 < 3$).