CONTRÔLE CONTINU TP

XLG1IU020 – Algorithmique et programmation pour les sciences

Un notebook bien organisé, identifiant les différentes parties des exercices est attendu. Ce dernier est a déposer sur Madoc dans la zone de dépôt dédiée. Les programmes écrits doivent être commentés, permettant ainsi une meilleure compréhension du code.

Exercice 1 : Compter les mots

L'objectif de cet exercice est d'implémenter un programme comptant le nombre de mots dans un texte. Un mot est défini comme une suite de caractères alphanumériques, séparés par des caractères que l'on appelle des séparateurs. Un caractère alphanumérique est une lettre (majuscule, minuscule, accentuée ou non) ou un chiffre. Tout autre caractère est considéré comme un séparateur. En utilisant le texte ci-dessous comme exemple, le nombre de mots est de 66.

```
Nous sommes le 11 octobre 2008. Avant-hier Jean-Claude a épluché dix-huit pommes de terre. En effet nous étions six convives. Cet homme-là est un spécialiste de l'épluchage. Il se considère lui-même comme le plus rapide du domaine. Aujourd'hui quatre pommes de terre sont en train de pourrir car, voyez-vous, quatorze d'entre elles ont suffi à nous rassasier!
```

1. Écrire une fonction `compte_mots(texte)` qui compte le nombre de mots (selon la défnition donnée précédement) dans le texte sous forme de chaine de caractères, donné en paramètre de la fonction. On utilisera une fonction booléenne `est_alphanum(caractere)` (donnée ci-dessous) qui renvoie vrai si et seulement si le caractère passé en paramètre est un caractère alphanumérique.
2. Au sein d'un programme, tester la fonction `compte_mots(texte)` avec le texte donné en exemple et vérifier que le nombre de mots calculé par votre fonction est correct.
3. Écrire une fonction `MotLePlusLong(texte)` qui détermine le mot le plus long d'un texte donné en paramètre.
4. Écrire une fonction qui stocke tous les mots d'un texte dans un tableau. Ce tableau ne devra pas contenir de doublons (i.e., chaque mot ne doit être présent qu'une seule fois dans le tableau).

```
# Fonction qui détermine si un caratère est alpha-numérique ou non.
# Retourne True si oui, False sinon.
def est_alphanum(caractere):
   return caractere.isalnum()
```

Exercice 2 : Matrice de Toeplitz

En algèbre linéaire, une matrice de Toeplitz (d'après Otto Toeplitz) ou matrice à diagonales constantes est une matrice dont les coefficients sur une diagonale descendant de gauche à droite sont les mêmes[1]. Les matrices de Toeplitz peuvent être utilisées dans divers domaines comme par exemple en traitement du signal, pour filtrer les signaux électriques ou en imagerie, pour reconstruire des images en IRM ou pour compresser des données vidéo. Elles peuvent aussi intervenir en physique statistique pour modéliser les corrélations dans les systèmes quantiques. La matrice A ci-dessous est une matrice de Toeplitz :

$$A = \begin{bmatrix} 1 & 2 & 3 & 5 & 7 \\ 4 & 1 & 2 & 3 & 5 \\ 2 & 4 & 1 & 2 & 3 \\ 0 & 2 & 4 & 1 & 2 \\ 9 & 0 & 2 & 4 & 1 \end{bmatrix}$$

En revanche, la deuxième matrice B ci-dessous n'est pas une matrice de Toeplitz :

$$B = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix}$$

Pour des raisons de simplification, nous considérerons des matrices composées uniquement de chiffres allant de 0 **à** 9**.**

1. Implémenter une fonction `affiche_matrice(matrice)` permettant d'afficher une matrice $n \times m$.
   En utilisant cette fonction pour la matrice A, le résultat doit être :

   ```
   1 2 3 5 7
   4 1 2 3 5
   2 4 1 2 3
   0 2 4 1 2
   9 0 2 4 1
   ```

2. Écrire une fonction `est_matrice_toeplitz(matrice)` qui retourne `True` si la matrice est une matrice de Toeplitz, et retourne `False` sinon.
   Au sein d'un programme, tester cette fonction avec les matrices A et B.
3. Implémenter une fonction `creer_matrice_toeplitz(n)` qui créer et retourne une matrice $n \times n$ de Toeplitz.
4. Au sein d'un programme, créer automatiquement, via les fonctions créées précédemment, une matrice 3×3 de Toeplitz et l'afficher.

[1] source: Wikipédia