# *Algorithmique avancee Fiche de TD Numero 2*

## Exercice 1 :

Ecrivez un algorithme permettant à l'utilisateur de saisir un nombre quelconque de valeurs, qui devront être stockées dans un tableau. L'utilisateur doit donc commencer par entrer le nombre de valeurs qu'il compte saisir. Il effectuera ensuite cette saisie. Enfin, une fois la saisie terminée, le programme affichera le nombre de valeurs négatives et le nombre de valeurs positives.

## Exercice 2 :

Ecrivez un algorithme calculant la somme des valeurs d'un tableau (on suppose que le tableau a été préalablement saisi).

## Exercice 3 :

Ecrivez un algorithme qui permette la saisie d'un nombre quelconque de valeurs, sur le principe de l'exercice 1 Toutes les valeurs doivent être ensuite augmentées de 1, et le nouveau tableau sera affiché à l'écran.

## Exercice 4 :

Ecrivez un algorithme permettant, toujours sur le même principe, à l'utilisateur de saisir un nombre déterminé de valeurs. Le programme, une fois la saisie terminée, renvoie la plus grande valeur en précisant quelle position elle occupe dans le tableau. On prendra soin d'effectuer la saisie dans un premier temps, et la recherche de la plus grande valeur du tableau dans un second temps.

## Exercice 5 :

Toujours et encore sur le même principe, écrivez un algorithme permettant, à l'utilisateur de saisir les notes d'une classe. Le programme, une fois la saisie terminée, renvoie le nombre de ces notes supérieures à la moyenne de la classe.

## Exercice 6 :

Ecrivez un algorithme qui permette de saisir un nombre quelconque de valeurs, et qui les range au fur et à mesure dans un tableau. Le programme, une fois la saisie terminée, doit dire si les éléments du tableau sont tous consécutifs ou non.

Par exemple, si le tableau est :

| | | | | | | |
|---|---|---|---|---|---|---|
| 12 | 13 | 14 | 15 | 16 | 17 | 18 |

Ses éléments sont tous consécutifs. En revanche, si le tableau est :

| | | | | | | |
|---|---|---|---|---|---|---|
| 9 | 10 | 11 | 15 | 16 | 17 | 18 |

Ses éléments ne sont pas tous consécutifs.

## Exercice 7 :

Ecrivez un algorithme qui trie un tableau dans l'ordre décroissant.

Vous écrirez bien entendu deux versions de cet algorithme, l'une employant le tri par sélection, l'autre le tri à bulles.

## Exercice 8 :

Ecrivez un algorithme qui inverse l'ordre des éléments d'un tableau dont on suppose qu'il a été préalablement saisi (« les premiers seront les derniers… »)

## Exercice 9 :

Ecrivez un algorithme qui permette à l'utilisateur de supprimer une valeur d'un tableau préalablement saisi. L'utilisateur donnera l'indice de la valeur qu'il souhaite supprimer. Attention, il ne s'agit pas de remettre une valeur à zéro, mais bel et bien de la supprimer du tableau lui-même ! Si le tableau de départ était :

| | | | | | | |
|---|---|---|---|---|---|---|
| 12 | 8 | 4 | 45 | 64 | 9 | 2 |

Et que l'utilisateur souhaite supprimer la valeur d'indice 4, le nouveau tableau sera :

| | | | | | |
|---|---|---|---|---|---|
| 12 | 8 | 4 | 45 | 9 | 2 |

## Exercice 10 :

Ecrivez l'algorithme qui recherche un mot saisi au clavier dans un dictionnaire. Le dictionnaire est supposé être codé dans un tableau préalablement rempli et trié.

## Exercice 11 :

Écrivez un algorithme qui fusionne deux tableaux (déjà existants) dans un troisième, qui devra être trié.

Attention ! On présume que les deux tableaux de départ sont préalablement triés : il est donc irrationnel de faire une simple concaténation des deux tableaux de départ, puis d'opérer un tri : comme quand on se trouve face à deux tas de papiers déjà triés et qu'on veut les réunir, il existe une méthode bien plus économique (et donc, bien plus rationnelle...)

Pour i ← 0 à 1

```
  Pour j ← 0 à 2
    X(i, j) ← Val
    Val ← Val + 1
  j Suivant
i Suivant
Pour i ← 0 à 1
  Pour j ← 0 à 2
    Ecrire X(i, j)
  j Suivant
i Suivant
Fin
```

## Exercice 12 :

Quel résultat produira cet algorithme ?

```
Tableau X(1, 2) en Entier
Variables i, j, val en Entier
Début
Val ← 1
Pour i ← 0 à 1
  Pour j ← 0 à 2
    X(i, j) ← Val
    Val ← Val + 1
  j Suivant
i Suivant
Pour j ← 0 à 2
  Pour i ← 0 à 1
    Ecrire X(i, j)
  i Suivant
j Suivant
Fin
```

## Exercice 13 :

Quel résultat produira cet algorithme ?

```
Tableau T(3, 1) en Entier
```

```
Variables k, m, en Entier
Début
Pour k ← 0 à 3
  Pour m ← 0 à 1
    T(k, m) ← k + m
  m Suivant
k Suivant
Pour k ← 0 à 3
  Pour m ← 0 à 1
    Ecrire T(k, m)
  m Suivant
k Suivant
Fin
```

## Exercice 14 :

Ecrivez un algorithme qui demande une phrase à l'utilisateur. Celui-ci entrera ensuite le rang d'un caractère à supprimer, et la nouvelle phrase doit être affichée (on doit réellement supprimer le caractère dans la variable qui stocke la phrase, et pas uniquement à l'écran).

## Exercice 15 : Cryptographie 1

Un des plus anciens systèmes de cryptographie (aisément déchiffrable) consiste à décaler les lettres d'un message pour le rendre illisible. Ainsi, les A deviennent des B, les B des C, etc. Ecrivez un algorithme qui demande une phrase à l'utilisateur et qui la code selon ce principe. Comme dans le cas précédent, le codage doit s'effectuer au niveau de la variable stockant la phrase, et pas seulement à l'écran.

## Exercice 16 : Cryptographie 2 - le chiffre de César

Une amélioration (relative) du principe précédent consiste à opérer avec un décalage non de 1, mais d'un nombre quelconque de lettres. Ainsi, par exemple, si l'on choisit un décalage de 12, les A deviennent des M, les B des N, etc.

Réalisez un algorithme sur le même principe que le précédent, mais qui demande en plus quel est le décalage à utiliser. Votre sens proverbial de l'élégance vous interdira bien sûr une série de vingt-six "Si...Alors"

## Exercice 17 : Cryptographie 3

Une technique ultérieure de cryptographie consista à opérer non avec un décalage systématique, mais par une substitution aléatoire. Pour cela, on utilise un alphabet-clé, dans lequel les lettres se succèdent de manière désordonnée, par exemple : HYLUJPVREAKBNDOFSQZCWMGITX

C'est cette clé qui va servir ensuite à coder le message. Selon notre exemple, les A deviendront des H, les B des Y, les C des L, etc.

Ecrire un algorithme qui effectue ce cryptage (l'alphabet-clé sera saisi par l'utilisateur, et on suppose qu'il effectue une saisie correcte).

## Exercice 18 : Cryptographie 4 - le chiffre de Vigenère

Un système de cryptographie beaucoup plus difficile à briser que les précédents fut inventé au XVIe siècle par le français Vigenère. Il consistait en une combinaison de différents chiffres de César.

On peut en effet écrire 25 alphabets décalés par rapport à l'alphabet normal :

- l'alphabet qui commence par B et finit par …YZA
- l'alphabet qui commence par C et finit par …ZAB
- etc.

Le codage va s'effectuer sur le principe du chiffre de César : on remplace la lettre d'origine par la lettre occupant la même place dans l'alphabet décalé.

Mais à la différence du chiffre de César, un même message va utiliser non un, mais plusieurs alphabets décalés. Pour savoir quels alphabets doivent être utilisés, et dans quel ordre, on utilise une clé.

Si cette clé est "VIGENERE" et le message "Il faut coder cette phrase", on procèdera comme suit :

La première lettre du message, I, est la 9e lettre de l'alphabet normal. Elle doit être codée en utilisant l'alphabet commençant par la première lettre de la clé, V. Dans cet alphabet, la 9e lettre est le D. I devient donc D.

La deuxième lettre du message, L, est la 12e lettre de l'alphabet normal. Elle doit être codée en utilisant l'alphabet commençant par la deuxième lettre de la clé, I. Dans cet alphabet, la 12e lettre est le S. L devient donc S, etc.

Quand on arrive à la dernière lettre de la clé, on recommence à la première.

Ecrire l'algorithme qui effectue un cryptage de Vigenère, en demandant bien sûr au départ la clé à l'utilisateur.

## Exercice 19 :

Soient Toto.txt et Tata.txt deux fichiers dont les enregistrements ont la même structure. Ecrire un algorithme qui recopie tout le fichier Toto dans le fichier Tutu, puis à sa suite, tout le fichier Tata (concaténation de fichiers).

## Exercice 20 :

Ecrivez une fonction qui purge une chaîne d'un caractère, la chaîne comme le caractère étant passés en argument. Si le caractère spécifié ne fait pas partie de la chaîne, celle-ci devra être retournée intacte. Par exemple :

- Purge("Bonjour","o") renverra "Bnjur"
- Purge("J'ai horreur des espaces"," ") renverra "J'aihorreurdesespaces"
- Purge("Moi, je m'en fous", "y") renverra "Moi, je m'en fous"

## Exercice 21 :

On va à présent réaliser une application complète, en utilisant une architecture sous forme de sous-procédures et de fonction.

Cette application a pour tâche de générer des grilles de Sudoku. Une telle grille est formée de 81 cases (9 x 9), contenant un chiffre entre 1 et 9, et dans laquelle aucune ligne, aucune colonne et aucune "sous-grille" de 3x3, ne contient deux fois le même chiffre.

Pour parvenir à nos fins, on va utiliser une méthode particulièrement barbare et inefficace : la génération aléatoire des 81 valeurs de la grille. On vérifiera alors que la grille satisfait aux critères ; si tel n'est pas le cas... on recommence la génération jusqu'à ce que la grille convienne. En pratique, la probabilité de générer une grille adéquate est si faible que cette méthode prendra sans doute beaucoup de temps, mais passons.

Tout le truc est de piger que vérifier que les neuf cases d'une ligne, d'une colonne, ou d'une sous-grille, sont toutes différentes, c'est en réalité du pareil au même. On va donc factoriser le code procédant à cette vérification sous la forme d'une fonction booléenne TousDifférents, à qui on passera un tableau de 9 valeurs en argument. La fonction renverra donc VRAI si les 9 valeurs du tableau sont toutes différentes, et FAUX sinon.

a. Ecrire la fonction TousDifferents

Maintenant, bien que ce ne soit pas indispensable (car ce code n'est pas spécialement répété), on choisit également par pure commodité de confier la génération au hasard de la grille de 81 cases à un module dédié, RemplitGrille. (ce module, à qui on passera notre tableau de 81 cases en argument, est forcément une procédure, puisqu'il a pour tâche d'en modifier les 81 valeurs).

b. Ecrire la procédure RemplitGrille

Il faut à présent vérifier que l'ensemble des lignes correspond à la condition voulue, à savoir qu'il n'y existe pas de doublons.

On réalise donc une fonction, VerifLignes, qui va vérifier les neuf lignes de notre grille une par une (en utilisant bien sûr la fonction TousDifférents, déjà écrite) et renvoyer VRAI si toutes les lignes sont correctes, FAUX dans le cas contraire.

c. Ecrire la fonction Veriflignes

On procède alors de même avec une fonction chargée de vérifir les colonnes, VérifColonnes.

d. Ecrire la fonction Verifcolonnes

...et encore à nouveau, avec cette fois la vrification des neuf "sous-grilles" 3x3.

e. Ecrire la fonction VerifSousGrilles (

Il ne reste plus qu'à écrire la procédure principale, et l'affaire est dans le sac !

f. Ecrire la procédure principale de l'application