# TD10 : éléments de correction

## Exercice 4

```
Fonction : DonnerIndice
Précondition : c appartient à l'ensemble {'a', ... , 'z', ' '}
Entrée :        - c : caractère
Local :         /
Sortie :        - entier
Début
|       Si c == ' ' Alors
|       |       Retourner 26
|       Fin Si
|       Retourner ASCII(c) - ASCII('a')         // donne le code ASCII du caractère, ex : ASCII('a') = 97
Fin
```

```
Fonction : Code2Gramme
Précondition : message ne contient que des caractères appartenant à l'ensemble {'a', ... 'z', ' '}
Entrée :        - grille : tableau 2D de 27*27 chaînes de caractères
                - message : chaîne de caractères
Local :         - indice1, indice2 : entier
Sortie :        - messageCode : chaîne de caractères
Début
|       Soit messageCode = ""
|       Soit indice1 = -1
|       Soit indice2 = -1
|       Pour i de 0 à Longueur(message) - 1 avec un pas de +2 Faire    // lecture des caractères 2 par 2
|       |       Soit indice1 = DonnerIndice(message[i])
|       |       Soit indice2 = DonnerIndice(message[i + 1])
|       |       Soit messageCode = messageCode + grille[indice1, indice2]
|       Fin Pour
|       Retourner messageCode
Fin
```

```
Fonction : DonnerCaractere
Précondition : indice appartient à l'intervalle [0, 26]
Entrée :        - indice : entier
Local :         /
Sortie :        - caractère
Début
|       Si indice == 26 Alors
|       |       Retourner ' '
|       Fin Si
|       Retourner Caractere(indice + ASCII('a'))        // conversion en caractère, ex : Caractere(97) = 'a'
Fin
```

```
Procédure : RechercheDansGrille
Précondition : le message ne contient que des caractères appartenant à l'ensemble {'a', ... 'z', ' '}
Entrée :        - grille : tableau 2D de 27*27 chaînes de caractères
                - message : chaîne de caractères
                - indice1 : entier (modifiable)
                - indice2 : entier (modifiable)
Local :         /
Sortie :        void
Début
|       Soit indice1 = -1
|       Soit indice2 = -1
|       Pour i de 0 à 26 avec un pas de +1 Faire
|       |       Pour j de 0 à 26 avec un pas de +1 Faire
|       |       |       Si grille[i, j] est égale à message Alors
|       |       |       |       Soit indice1 = i
|       |       |       |       Soit indice2 = j
|       |       |       |       Retourner       // sortir de la procédure
|       |       |       Fin Si
|       |       Fin Pour
|       Fin Pour
Fin
```

```
Fonction : Decode2Gramme
Précondition : c appartient à l'ensemble {'a', ... , 'z', ' '}
Entrée :        - grille : tableau 2D de 27*27 chaînes de caractères
                - message : chaîne de caractères
Local :         - indice1, indice2 : entier
Sortie :        - messageDecode : chaîne de caractères
Début
|       Soit messageDecode = ""
|       Soit indice1 = -1
|       Soit indice2 = -1
|       Pour i de 0 à Longueur(message) - 1 avec un pas de +3 Faire    // lecture des caractères 3 par 3
|       |       Soit msg = SousChaine(message, i, 3)    // sous-chaîne de message de l'indice i à i+3 (*)
|       |       RechercherDansGrille(grille, msg, indice1, indice2)        // modifie indice1 et indice2 !
|       |       Si indice1 ≠ -1 ET indice2 ≠ -1 Alors
|       |       |       Soit messageDecode = messageDecode   + DonnerCaractere(indice1)
|       |       |                                            + DonnerCaractere(indice2)
|       |       Fin Si
|       Fin Pour
|       Retourner messageDecode
Fin
```

(*) en C# on utilisera la fonction Substring() qui s'applique sur une chaîne, prend en paramètre la position de départ et le nombre de caractère à extraire depuis cet indice, et retourne la sous-chaine correspondante.
Ex :

```
string message = "hello world !"
string s = message.Substring(6, 5) // s vaut "world"
```

```
Procédure : SauverGrille
Entrée :        - grille : tableau 2D de 27*27 chaînes de caractères
                - chemin : chaîne de caractères
Local :         - f : flux d'écriture
Sortie :        void
Début
|       Soit f = flux d'écriture initialisé sur chemin
|       // écrire les dimensions de la grille sur la première ligne
|       Écrire(f, "{Longueur(grille, 0)},{Longueur(grille, 1)}\n") // ex : 3,3
|       // écrire la grille ligne par ligne (séparateur '\n'), colonne par colonne (séparateur ',')
|       Pour ligne de 0 à Longueur(carreMagique, 0) − 2 avec un pas de +1 Faire
|       |       Pour colonne de 0 à Longueur(grille, 1) − 1 avec un pas de +1 Faire
|       |       |       Soit valeur = grille[ligne, colonne]
|       |       |       Écrire("{valeur},")
|       |       Fin Pour
|       |       Soit valeur = grille[ligne, Longueur(grille, 1) − 1]
|       |       Écrire("{valeur}\n")
|       Fin Pour
|       FermerFlux(f)
Fin


Fonction : ChargerGrille
Entrée :        - chemin : chaîne de caractères
Local :         - f : flux de lecture
Sortie :        - grille : tableau 2D de chaînes de caractères
Début
|       Soit f = flux de lecture initialisé sur chemin
|       // lire les dimensions de la grille sur la première ligne
|       Soit ligne = LireLigne(f)
|       Soit details = Séparer(ligne, ',') // ex : tableau ["3", "3"]
|       Soit lignes = Entier(details[0])
|       Soit colonnes = Entier(details[1])
|       // création de la grille
|       Soit grille = tableau 2D de lignes*colonnes chaînes de caractères
|       // lecture de la grille ligne par ligne (séparateur '\n'), colonne par colonne (séparateur ',')
|       Soit i = 0
|       Faire
|       |       Soit ligne = LireLigne(f)
|       |       Soit details = Séparer(ligne, ',')
|       |       Pour j de 0 à colonnes avec un pas de +1 Faire
|       |       |       Soit grille[i, j] = details[j]
|       |       Fin Pour
|       |       Soit i = i + 1
|       Tant Que ligne n'est pas nulle
|       FermerFlux(f)
|       Retourner grille
Fin


Procédure : Main
Entrée :        void
Local :         - grille : tableau 2D de 27*27 chaînes de caractères
                - chemin : chaîne de caractères
Sortie :        void
Début
|       Soit creerGrille = SaisirBooléen()
|       Si creerGrille est VRAI Alors
|       |       Soit grille = tableau 2D de 27*27 chaînes de caractères
|       |       Soit chemin = SaisieChaîne()
|       |       SauverGrille(grille, chemin)
|       Sinon
|       |       Soit chemin = SaisieChaîne()
|       |       Soit grille = ChargerGrille(chemin)
|       Fin Si
Fin
```