Algorithmes et Structures de Données
Mardi 24 Octobre 2017 – correction

Exercice 1 : Pile (5 pts)

Soit la fonction de Newman-Conway implémentée comme suit :

```
Fonction u (n : entier) : entier
Var res : entier
Début
Si (n <= 1)
    Alors res ← 1
    Sinon res ← u(u(n-1){@1}){@2} + u(n - u(n-1){@3}){@4}
FinSi
retourner(res)
Fin
```

Simuler la pile pour l'appel `écrire(u(3)){@0}` dans le programme principal.

~~@4 ; n=1 r=1~~
~~@4 ; n=1 r=1~~
~~@3 ; n=1 r=1~~
~~@2 ; n=1 r=1~~
~~@1 ; n=1 r=1~~
~~@3 ; n=2 r=1+1=2~~
~~@4 ; n=1 r=1~~
~~@3 ; n=1 r=1~~
~~@2 ; n=1 r=1~~
~~@1 ; n=1 r=1~~
~~@2 ; n=2 r=1+1=2~~
~~@4 ; n=1 r=1~~
~~@3 ; n=1 r=1~~
~~@2 ; n=1 r=1~~
~~@1 ; n=1 r=1~~
~~@1 ; n=2 r=1+1=2~~
~~@0 ; n=3 r=2+1=3~~

17 appels

Exercice 2 : Position (5 pts)

On souhaite écrire la fonction `pos` définie sur le type `chaine`. `pos(c1, c2)` retourne la position d'une chaine `c2` dans une autre `c1` ; 0 si `c2` n'apparaît pas dans `c1`.

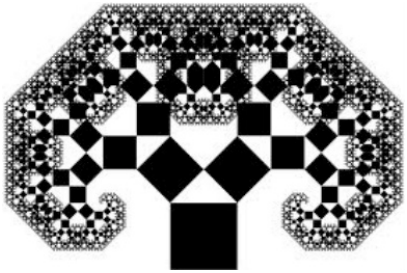
2.1. Expliquer en français le principe **itératif** de cette fonction.

On parcourt c1 pour y trouver le premier caractère de c2. Si on ne le trouve pas on retourne 0, sinon on regarde si les caractères suivants dans c1 correspondent à ceux de c2. Si oui, on retourne l'indice du premier caractère de c2 dans c1, sinon on recommence le principe avec l'occurrence suivante du premier caractère de c2 dans c1.

2.2. Ecrire en pseudo-langage la fonction **itérative** qui réalise cette opération.

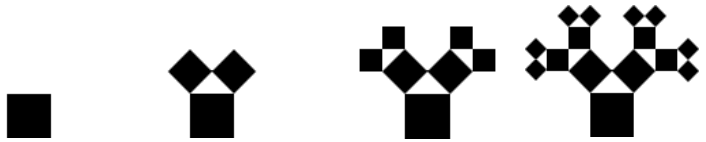
```
Fonction pos (c1,c2 : chaine) : entier
Var trouve : booleen
    i,j,posdeb,res : entier
Début
res←0
j←1
trouve←faux
TantQue (j+lg(c2)-1<=lg(c1)) et not trouve Faire
    i←1
    TantQue (j+lg(c2)-1<=lg(c1)) et (c2[1]<>c1[j]) Faire
       j←j+1
    FinTantQue
    posdeb←j
    TantQue (i<=lg(c2)) and (j<=lg(c1)) et (c2[i]=c1[j]) Faire
       j←j+1
       i←i+1
    FinTantQue
    Si i>length(c2)
       Alors  trouve←vrai
              res←posdeb
       Sinon j←posdeb+1
    FinSi
FinTantque
Retourner(res)
Fin
```


Exercice 3 : Arbre de Pythagore (5 pts)



La construction de l'arbre de Pythagore débute avec un simple carré. Sur ce carré sont construits deux autres carrés, tels que les coins des carrés soient en contact.

La procédure est appliquée récursivement jusqu'à l'infini (ou presque !). L'illustration ci-dessous donne les 4 premiers appels de la construction.



3.1. Montrer que les égalités suivantes sont vraies quelle que soit l'orientation des carrés :

```
* Si le carré est horizontal :
c.x = b.x  car a.y-b.y=0
c.y = b.y  - coté du carré (= b.x-a.x)
d.x = a.x  car a.y-b.y=0
d.y = a.y - coté du carré (= b.x-a.x)
* Si le carré est diagonal :
c.x = b.x - diagonale/2 (=a.y-b.y)
c.y = b.y - diagonale/2 (=b.x-a.x)
d.x = a.x - diagonale/2 (=a.y-b.y)
d.y = a.y - diagonale/2 (=b.x-a.x)
       ->          ->
```

Puisque `DE = 1/2 AC`, alors

```
e.x = d.x + (c.x-a.x)/2
e.y = d.y + (c.y-a.y)/2
```

3.2. Expliquer en français la méthode pour réaliser cet affichage de façon **récursive**.

Si le coté du carré est trop petit, on s'arrête, sinon on dessine le carré courant (a,b,c,d), on calcule les coordonnées des point c,d et e comme mentionné dans 3.1., et on appelle récursivement la fonction `Pythagore-tree` sur (d,e) puis (e,c), et ainsi de suite.

3.3. Ecrire en pseudo-langage la procédure `Pythagore-tree (E a,b : coord)`

On suppose que la procédure `dessine-carré(E a,b,c,d : coord)` et la fonction `distance(a,b : coord)` sont données.

```
Procedure Pythagore-tree (E a,b : coord)
Var c,d,e : coord
Début
  Si distance(a,b)>6
     Alors c.x ← b.x - (a.y-b.y)
           c.y ← b.y - (b.x-a.x)
           d.x ← a.x - (a.y-b.y)
           d.y ← a.y - (b.x-a.x)
           e.x ← d.x + (c.x-a.x)/2
           e.y ← d.y + (c.y-a.y)/2
          dessine-carré(a,b,c,d)
          Pythagore-tree(d,e)
         Pythagore-tree(e,c)
  Finsi
Fin
```

Exercice 4 : Chiffre de Vigenère (5 pts)

Le **chiffre de Vigenère** est un algorithme de codage de message qui consiste à changer une lettre par une autre, qui n'est pas toujours là même. Cela permet une plus grande sécurité. Cet algorithme utilise une clé sous la forme d'un mot et la **table de Vigenère (donnée ci-dessous)**. Plus la clé sera longue, plus le cryptage sera dur à deviner.

Principe de chiffrement
Pour chaque lettre du message à coder, on écrit chaque lettre de la clé et on répète le motif autant de fois que nécessaire. Les lettres du message correspondent aux colonnes de la table de Vigenère et celles de la clé aux lignes. A chaque couple (lettre message, lettre clé), correspond une seule lettre dans la table. Les lettres concaténées formeront le message codé.
Exemple :
Pour le message « bonjour chez vous » et la clé « prisonnier » :

```
BONJOUR CHEZ VOUS
PRISONN IERP RISO
QFVBCHE KLVO MWMG
```

Le message codé est « `QFVBCHE KLVO MWMG` »

4.1. Expliquer en français comment effectuer la traduction d'un message (message et message codé sont enregistrés dans 2 fichiers texte) de façon optimale. La clé et la table sont données dans deux fichiers textes séparés. Décrire les structures de données utilisées.
Tout d'abord, il faut charger la table de vigenère à partir du fichier texte et la stocker dans une matrice de caractères. Pour cela, on lit ligne par ligne le fichier (26 lignes) et pour chaque ligne lue dans une chaine de caractères, on copie chaque caractère dans la matrice.
Ensuite, pour traduire une chaine de caractères en une chaine codée, on applique le principe de chiffrement. Pour traduire tout un fichier texte, on lit ligne par ligne le fichier dans une chaine de caractères qu'on traduit au fur et à mesure et qu'on écrit dans un nouveau fichier texte.
4.2. Ecrire en pseudo-langage le programme qui réalise cette traduction (possibilité écrire plusieurs procédures ou fonctions).
Je suppose que la clé tient sur une ligne de fichier et que le message à coder peut en contenir plusieurs.

```
Programme Vigenere
Type mat-car = tableau[1..26] de car
Var tv : mat-car
    fm, ft, fc : FT
    cle, mot : chaine
```



```
Fonction charge-table (nomfich : chaine) : mat-car
Var tv : mat-car
    i,j : entier
    c : chaine
Début
f ← OuvrirEnLecture(nomfich)
Pour i←1 à 26 inc +1 faire
      c ← LireChaine(f)
      Pour j←1 à 26 inc +1 Faire
            tv[i,j] ← c[j]
      FinPour
FinPour
fermer(f)
retourner(tv)
Fin

Function char-to-int(c : car) : entier
début
  retourner(ord(c)-ord('A')+1)
Fin


Fonction traduit-mes (mot,cle : chaine, tv : mat-car) : chaine
Var trad : chaine
    i,j : entier
Début
trad←''
j←1
Pour i←1 à lg(mot) inc +1 faire
      Si ord(mot[i])>=ord('A') et ord(mot[i])<=ord('Z') {si c'est une lettre}
         Alors trad ← concaténer(trad, tv[char-to-int(cle[j]),char-to-int(mot[i])]
                  j←j+1
                  Si j>lg(clé)
                     Alors j←1
                  FinSi
            Sinon trad ← concaténer(trad, mot[i])
      FinSi
FinPour
retourner(trad)
Fin

Début
tv←charge-table('vigenere')
fm←ouvrirEnLecture('fich-message')
fc←ouvrirEnLecture('fich-cle')
ft←ouvrirEnEcriture('fich-mes-trad')
cle←lireChaine(fc)
fermer(fc)
TantQue not finFichier(fm) Faire
      mot←lireChaine(fm)
      écrireChaine(ft,traduit-mes(mot,cle,tv))
FinTantQue
fermer(fm)
fermer(ft)
Fin
```