# DS n°2 : Corrigé

**Ex ① :** (2,75)

1)- Il s'agit d'un traitement récurrent d'ordre deux, car pour calculer un terme (0,25) de la suite on utilise deux termes précédents. (0,75) (0,5) ( $U_i = -U_{i-1} + 2 * U_{i-2}$ )

2)- Fonction suite ( n:entier ) : entier (0,25)

```
Début
    u0 ← 0 , u1 ← 3                                   (0,25)
    pour i de 2 à (n-1) faire                         (0,25)
        u ← - u1 + 2*u0                               (0,5)
        u0 ← u1                                       (0,25)
        u1 ← u
    Fin
    Retourne u                                        (0,25)
Fin
```

T.D.O

| O | T/N |
|---|---|
| u0 | entier |
| u1 | |
| u | entier |
| i | |

(0,25)

**Ex ② :** (2,75)

① $\arctan(x) = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \dots -$

si $x = 1 \Rightarrow \arctan(x) = \frac{\pi}{4}$ ⇒ Remplacer x par 1.

(0,75) $\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \dots -$

② Fonction James ( Epsilon : réel ) : réel (0,25)

```
Début
    S1 ← 1                                            (0,25)
    i ← 1
    signe ← -1.
    Répéter
        S2 ← S1                                       (0,25)
        i ← i + 2                                     (0,25)
        S1 ← S2 + signe * 1/i                         (0,25)
        signe ← - signe
    Jusqu'à ( ABS ( 4S1 - 4S2) < Epsilon)             (0,25)
    Retourner 4*S1                                    (0,25)
Fin
```

T.D.O

| O | T/N |
|---|---|
| i | entier |
| signe | entier |
| S1 | réel |
| S2 | |

(0,25)

Ex (3) : (5,5)

Procédure Remplir (@F: Fich ; e: enreg) (0,25)

Début

ouvrir ("Niven.dat", F, "wb") (0,25)

pour n de 100 à 1000 faire (0,25)

(0,5) | Si Niven(n) ET [ ] (EstNum(conv_DecHex(n))) Alors = Faux

| e.NV ← n

| e.NV_Hex ← conv_Dec_Hex(n) (0,5)

Finsi

Ecrire(F, e) (0,25)

Fin

Fermer(F) (0,25)

Fin.

T.D.N.T :

| TYPE | |
|---|---|
| Enreg = Enregistrement | |
| NV | : entier |
| NV_Hex | : chaîne |
| Fin | |
| Fich = fichier de Enreg | |

(0,25)

(0,25)

Algorithmes des modules utilisés :

(0,25) Fonction Niven (n: entier): Booléen

Début

(0,25) Retourner (n MOD somme(n)=0)

Fin

T.D.O

| O | T/N |
|---|---|
| somme | Fonction |

(0,25) Fonction somme (n: entier): entier

Début

(0,25) si n=0 Alors Retourner 0

(0,25) | sinon Retourner (n MOD 10) + somme (n DIV 10)

finsi

Fin

(0,5)

T.D.O.G :

| O | T/N |
|---|---|
| ~~e~~ | ~~Enreg~~ |
| n | entier |
| Niven | Fonction |
| conv_Dec_Hex | Fonction |

(0,25)

autrement (solution itérative)

T.D.O

| O | T/N |
|---|---|
| - | - |

T.D.O

| O | T/N |
|---|---|
| S | entier |

S ← 0
Répéter
| S ← S + n MOD 10
| N ← N DIV 10
Jusqu'à (N=0)
Retourner S.

(2)

```
Fonction Conv_Dec_Hex ( n: entier ) : chaîne
Début                                          (0,25)
    CH ← "0123456789ABCDEF"
    Hex ← ""                          (0,25)
    Tantque (N ≠ 0) faire
        Hex ← CH[n MOD 16] + Hex      (0,25)
        n ← n Div 16
    Fin
    Retourne Hex.                     (0,25)
Fin
```

T.D.O

| O | T/N |
|---|---|
| CH | constante |
| Hex | chaîne |

⇒ On accepte toute autre solⁿ correcte.

Barème : - Remplir le fichier : 2,25
- Fonction Vérifie un Nb Niven : 1,5
- Fonction Conversion Dec-Hex : 1
- T.D.N.T + T.D.O.G : 0,75

} 5.5 pts.

Rq : L'élève pensera à écrire la fonction Somme dans le corps de fonction qui vérifie si un nbre est de Niven ou non ⇒ Répartition de (1,5 pts) selon la solⁿ proposée.

Ex (4) : (9 pts)

(1) Algorithme du programme principal : (2)

Algorithme Cryptage
Début
| Saisir (Key, F1)
| Remplir (Key, F1, F2)
| Afficher (F2).
Fin

(1,5) (0,5)

T.D.O.G

| O | T/N |
|---|---|
| Key | chaîne |
| F1 | Texte |
| F2 | Texte |
| Saisir | Procédure |
| Remplir | |
| Afficher | |

(2) Algorithmes des modules envisagés : (7,5)

(1,75 pt) * Procédure Affichage :

Procédure Afficher (@F2 : Texte) (0,25)
Début
| Ouvrir ("F_res.txt", f2, "r") (0,25)
| Tantque Non (fin_fichier(f2)) faire (0,25)
| | Lire_ligne (F2, CH). (0,25)
| | Ecrire (CH). (0,25)
| Fin
| Fermer (F2). (0,25)
Fin

T.D.O

| O | T/N |
|---|---|
| CH | chaîne |

(0,25)

⟹ (0,25) par type d'erreur.

(2 pts) * Procédure Saisir (@Key : chaîne, @F1 : Texte) (0,25)
Début
| Ouvrir ("F_init.txt", F1, "r") (0,25)
| Lire_ligne (F1, CH). (0,25)
| K ← "" (0,25)
| pour i de 0 à long(CH)-1 faire (0,25)
| | K ← K + CHR(Alea(65,91)) (0,25)
| Fin
| Key ← Gen_Bin (K). (0,25)
| Fermer (F1). (0,25)
Fin

(3)

⇒ Pour saisir la clé de sécurité il faut lire la 1ère ligne du fichier initial, pour savoir la longueur de la clé à générer.

⇒ la Fonction Gen_Bin permet de générer une chaîne binaire à partir des code ASCII des caractères d'une chaîne (K ou après, ligne)

⇒ Key est une chaîne binaire à utiliser durant avec toutes les lignes du fichier lors du cryptage XOR.

T.D.O

| O | T / N |
|---|---|
| CH | chaîne |
| K | chaîne |
| i | entier |
| Gen_Bin | Fonction |

ligne du fichier F1. clé aléatoire contenant des caractères ["A"-"Z"]

0,25

0,75

Fonction Gen_Bin (CH:chaîne): chaîne
Début.
| Bin ← "" (vv)
| pour i de 0 à long(CH) faire
| | Bin ← Bin + Conv_Dec_Bin(ORD(CH[i]))
| Fin
| Retourner Bin.
Fin

T.D.O

| O | T / N |
|---|---|
| Bin | chaîne |
| i | entier |
| Conv_Dec_Bin | Fonction |

0,25

## *- Procédure Remplir : (7pt)

```
Procédure Remplir (Key: chaîne, @F1, F2 : Texte)        (0,25)
Début                                                    (0,25)
    ouvrir ("F_init.txt", F1, "r")                       (0,5)
    ouvrir ("F_res.txt", F2, "w")
    Tantque Non(Fin_Fichier(F1)) faire.                  (0,25)
        lire_ligne (F1, ligne).                          (0,25)
        CH ← Gen_Bin (ligne)                             (0,25)
        CH_C ← crypter (Key, CH)                         (0,25)
        Ecrire_nl (F2, CH_C)                             (0,25)
    Fintque
    Fermer (F1)                                          (0,25)
    Fermer (F2).
Fin
```

T.D.O

| ○ | T / N |
|---|---|
| ligne | chaine |
| CH | chaine (Binaire) |
| CH_C | chaine (Cryptée). |
| Gen_Bin | fonction |
| Crypter | fonction. |

(0,25)

2 chaine Binaires

Key → CH1, CH2 : chaine

```
Fonction Crypter. (CH1, CH2 : chaine) : chaine
Début                                                    (0,5)
    CH ← Ou_exclusif (CH1, CH2)
    X ← Alea (65, 91)
    CH_C ← codage (X, CH)
    Retourner CH_C
Fin                                                      (0,25)
```

T.D.O

| ○ | T / N |
|---|---|
| CH | chaine |
| X | entier |
| CH_C | chaine |
| Ou_exclusif | Fonction |
| Codage | Fonction |

Fonction Ou_exclusif (CH1, CH2 : chaîne) : chaîne
Début
    CH ← ""
    pour i de 0 à long(CH1) - 1 faire
        si CH1[i] ≠ CH2[i] Alors
            CH ← CH + "1"
        sinon
            CH ← CH + "0"
        Finsi
    Finpour
    Retourner CH.
Fin

0,75

0,25

T.D.O

| ø | T/N |
|---|---|
| CH | à entrée chaîne (Binaire) |

Fonction Codage (X : entier, CH : chaîne) : chaîne.
Début
    CH_C ← ""
    Tantque (CH ≠ "") faire
        D ← conv_Bin_Dec (sous_chaine (CH, 0, 8))
        CH_C ← CH_C + CHR (D+X)
        ~~CH~~ effacer (CH, 0, 8)
    Fintq.
    Retourner CH_C.
Fin

0,75

0,25

T.D.O

| ø | T/N |
|---|---|
| CH_C<br>D<br>Conv_Bin_Dec | chaîne<br>entier<br>Fonction |

```
Fonction Conv_Dec_Bin ( N : entier) : chaine.
Début.
    Bin ← "00000000"                    i ← 7
    Tantque (N ≠ 0) faire
        si (N MOD 2 ≠ 0) Alors
            Bin[i] ← "1"
        Finsi
        i ← i-1.
        N ← N DIV 2.
    Fin
    Retourner Bin.
Fin.
```

0,75

⇒ on accepte toute autre str correcte.
⇒ il faut vérifie que la chaine Bin est de 8 bits.

autrement :

```
Bin ← "
Répéter
    si (N MOD 2 = 0) Alors
        Bin ← "0" + Bin
    sinon
        Bin ← "1" + Bin
    Finsi
    N ← N DIV 2
Jusqu'à (N = 0).
Tantque (long(Bin) < 8) faire
    Bin ← "0" + Bin.
Fin.
```

0,25

T.D.O

| O | T/N : |
|---|---|
| Bin<br>i | chaine<br>entier. |

Fonction Conv_Bni_Dec ( CH: chaine ): entier
Début.
    D ← 0 , p ← 1.
    pour i de long (CH) à 1 à 0 faire ( pas = -1 )
        D ← D + p * val ( CH[i] )
        p ← 2 * p.
    Fpour.
    Retourner D.
Fin.

⇒ on accepte toute autre Sté correcte.

T.D.D

| 0 | T / N. |
|---|---|
| D, i, p | entier |

Barème Ex 4 :

1) Algorithme P.P – Décomposition + Cohérence + T.D.O.G : 0,75 + 0,5 + 0,25

2) Algorithme des modules :

* – Saisie de la clé de cryptage + Conv_Binaire.
  - Longueur de la clé
  - Lecture d'une ligne de F1.
  - saisi aléatoire des caractère majuscule
  - Concaténation.
  - conversion de la clé en Binaire.
  - Ajout des zero à gauche ⇒ 8 bits.

* Remplir le fichier F2 à partir de f1 et la clé :
  - ouverture en lecture de F1.) + fermeture.
  - ouverture en écriture de F2)
  - lecture ligne par ligne de F1.
  - conversion de ligne vers des chaine Binaire.
  - cryptage de la ligne (CH).
    - Application de Ou-exclusif
    - choix de la valeur X ∈ [65-90].
    - Détermination du code ASCII +
    - Concaténation de la chaine cryptée.
  - extraction de 8 bits Binaire.
  - reconvertir vers la Base Dec

T.D.O

* Afficher le contenu de F2 :
  - ouverture
  - lecture ligne
  - Affichage
  - fermeture F2.