Objectifs

- ☞ Prise en main du langage Fortran.
- ☞ Vous utiliserez, dans la mesure du possible, des procédures externes.
- ☞ Merci d'accorder l'attention nécessaire à la qualité et l'organisation de votre code.

Exercice 1 [Entiers parfaits]

Soit N un entier non nul.

1. Écrire le pseudo-code d'un algorithme permettant de lister tous les nombres entiers inférieurs à N et égaux à la somme de leurs diviseurs (sauf eux-mêmes). de tels entiers sont dits parfaits. Analyser sa complexité.
2. Implémenter votre algorithme en Fortran. Les sorties de votre programme doivent ressembler à celles de la figure 1.

```
. Veuillez indiquer la limite N : 56

    ...       6 est somme de ses diviseurs
    ...      28 est somme de ses diviseurs

STOP Fin entiers parfaits ...
lama-495:Illustrations ouzia$
```

FIGURE 1 – Sortie du programme entiers parfaits.

Indication : Utilisez la fonction $mod(i, j)$ qui donne le reste de la division de i par j.

Exercice 2 [Nombres premiers]

Il sera question, dans cet exercice, des nombres premiers. Pour rappel, un entier naturel p est *premier* si 1 et p sont ses seuls diviseurs.

1. Donner le pseudo-code d'un algorithme `estPremier` permettant de tester si un entier lu à partir de l'entrée standard est premier ou pas. Vous pouvez utiliser la procédure *intrinsèque* `mod(a,b)` retournant le reste de la division de a par b (et non l'inverse !).
2. Quelle est la complexité de votre procédure ?
3. Implémenter votre programme en Fortran. Les sorties de ce dernier doivent ressembler à celles indiquées dans les deux figures 2.
4. Donner le pseudo-code d'un algorithme `EntiersPremiers` retournant tous les entiers premiers inférieurs ou égaux à l'entier `limite` lu à partir de l'entrée standard. Quelle en est sa complexité ?
5. Implémenter votre procédure en Fortran. Les sorties de votre programme doivent ressembler à celles de la figure 3.

A présent, nous nous intéresserons aux entiers de *Hamming*. Un entier naturel non nul h est dit de Hamming si ses diviseurs premiers appartiennent à $\{2, 3, 5\}$.

```
. Bienvenue dans est premier

  . Veuillez indiquer votre entier : 1237
  . [Est premier ?] : OUI

STOP Fin primes ...
lama-495:Primes ouzia$
```

FIGURE 2 – Sortie du programme est premier.

```
. Bienvenue dans lister premiers

  . Veuillez indiquer votre entier limite : 35
  . [La liste]:
        1           2
        2           3
        3           5
        4           7
        5          11
        6          13
        7          17
        8          19
        9          23
       10          29
       11          31

STOP Fin primes ...
lama-495:Primes ouzia$
```

FIGURE 3 – Sortie du programme lister premiers.

```
. Bienvenue dans est hamming

  . Veuillez indiquer votre entier : 678
  . [Est Hamming ?] : NON

STOP Fin Hamming ...
lama-495:Primes ouzia$
```

FIGURE 4 – Sortie du programme est Hamming.

```
. Bienvenue dans lister Hammings

  . Veuillez indiquer votre entier limite : 37
  . [La liste]:
        1           2
        2           3
        3           4
        4           5
        5           6
        6           8
        7           9
        8          10
        9          12
       10          15
       11          16
       12          18
       13          20
       14          24
       15          25
       16          27
       17          30
       18          32
       19          36

STOP Fin lsiter Hamming ...
lama-495:Primes ouzia$
```

FIGURE 5 – Sortie du programme lister Hamming.

6. Donner le pseudo-code d'un algorithme retournant oui ou non suivant que l'entier lu à partir de l'entrée standard est de Hamming ou pas.
7. Implémenter votre algorithme en `Fortran`. L'affichage (ou les sorties) de votre programme doit ressembler à celui de la figure 4.
8. Donner le pseudo-code d'un algorithme permettant de lister tous les entiers de Hamming inférieurs ou égaux à une valeur limite lue à partir de l'entrée standard. Quelle en est sa complexité ?
9. Implémenter votre algorithme en `Fortran`. Les sorties de votre algorithme doivent ressemble à celles indiquées dans la figure 5.

Exercice 3 [Ordre lexicographique]

Soient $a = (x_1, \ldots, x_n)$ et $b = (y_1, \ldots, y_n)$ deux uplets appartenant à $\mathbb{N}^n$.

Notons par $\prec$ l'ordre lexicographique stricte sur $\mathbb{N}^n$. Ainsi, $a \prec b$ s'il existe un entier $k \in \{1, \ldots, n\}$ tel que :

$$\begin{cases} x_i = y_i, & \forall i \leq k, \\ x_{k+1} < y_{k+1}. \end{cases}$$

1. Donner le pseudo-code d'un algorithme permettant de comparer deux vecteurs suivant l'ordre *lexicographique*
2. Implémenter votre algorithme en `Fortran`. Vos affichage doivent correspondre à celui demandé dans l figure 6.

```
. Bienvenue dans le comparateur lexinf


  Vecteur u :   3    6    7    5    7
  Vecteur v :   3    6    7    5    5

  [u lexinf v ?] : NON

STOP Fin lexorder
```

FIGURE 6 – Affichage demandé.

3. Analyser sa complexité.

Exercice 4 [Lire puis écrire une matrice]

L'objectif de cet exercice est d'écrire et de lire les entrées d'une matrice sur et via la sortie standard (c-à-d, le terminal).

1. Écrire un programme permettant de lire puis d'afficher une matrice $A \in \mathcal{M}_{2\times 2}(\mathbb{R})$. Attention votre affichage doit être soigné. Lors de la saisie votre affichage doit ressembler à celui de la figure 7. Quant à l'affichage final, il doit ressemble à celui de la figure 8.

```
Taille de la matrice :
   Nombre de lignes    :   2
   Nombre de colonnes :   3

 Merci de saisir (ligne par ligne) les entrees de la mat

   Ligne :  1
     col 1 : 6
     col 2 : 8
     col 3 : 9

   Ligne :  2
     col 1 : 7
     col 2 : 8
     col 3 :
```

FIGURE 7 – Affichage partiel.

```
Taille de la matrice :
   Nombre de lignes    :   2
   Nombre de colonnes :   3

 Merci de saisir (ligne par ligne) les entrees de la matrice :

   Ligne :  1
     col 1 : 6
     col 2 : 8
     col 3 : 9

   Ligne :  2
     col 1 : 7
     col 2 : 8
     col 3 : 9


 Voici les entrees de la matrice :

  6      8      9
  7      8      9

STOP write and read matrices ...
```

FIGURE 8 – L'affichage final.

Pour y parvenir sans séquelles psychologiques majeures, commencez par écrire un programme vous permettant d'afficher sur la sortie standard les entrées d'une matrice constante que vous aurez définie. Puis, une fois que cette étape passée, lancez-vous dans l'écriture de la partie du programme vous permettant de lire les entrées d'une matrice quelconque. Cette dernière partie est l'inverse de la première.

Exercice 5 [Récurrence non linéaire]

Soit $(u_n)_{n\in\mathbb{N}}$ la suite de nombres réels donnée par :

$$\begin{cases} u_{n+1} = \frac{1}{2}u_n + \sqrt{1+\cos(u_{n-1})} \\ u_0 = 3,\ u_1 = \sqrt{3} \end{cases}$$

```
 Veuillez indiquer le rang N : 5

     iteration   1 :    0.97
     iteration   2 :    1.40
     iteration   3 :    1.95
     iteration   4 :    2.06

  Le   5-eme terme vaut :    2.06

 STOP Fin recurrence ...
```

FIGURE 9 – Sorties demandées.

1. Donner le pseudo-code d'une procédure qui calculerait, étant donné un entier naturel n, la valeur du terme u_n.
2. Implémenter votre procédure en Fortran. Les sorties de votre programme doivent ressembler à ceux de la figure 9
3. Analyser sa complexité.

Exercice 6 [Anti-transposée]

Soit n un entier naturel non nul. Soit M une matrice appartenant à $\mathcal{M}_{n\times n}(\mathbb{R})$. *L'anti-transposée* de la matrice M est la matricée obtenu en permutant symétriquement ses éléments relativement à sa diagonale secondaire. Par exemple, l'anti-transposée de la matrice A ci-dessous est la matrice B.

$$A = \begin{bmatrix} 2 & 7 & 13 \\ 4 & 11 & 8 \\ 8 & 9 & 14 \end{bmatrix}, \; B = \begin{bmatrix} 14 & 8 & 13 \\ 9 & 11 & 7 \\ 8 & 4 & 2 \end{bmatrix}.$$

1. Donner le pseudo-code d'un algorithme déterminant l'anti-transposée d'une matrice carrée.
2. Analyser la complexité de votre algorithme
3. Implémenter votre algorithme en `Fortran`. Pour l'initialisation de la matrice vous utiliserez une matrice aléatoire d'entiers tirés uniformément dans l'intervalle $[1, 50]$. Pour cela, vous utiliserez les procédures `rand()` (retourne un nombre réel tiré uniformément dans l'intervalle $[0, 1]$.) et `floor()` (arrondi inférieur de son argument.). Enfin, votre affichage doit être identique à celui de la figure 10.

```
Bienvenue dans anti-transposeur version 1.0 :

  Ordre de la matrice (matrice carree) : 7


    . La matrice :

          1       7      38      23      27      11       3
         34      34      46      19      26      41       2
          3      26      33       1      19       4      21
         34      29      46      42      26       5      33
         21      35      45      38      13       3      37
         17      31      38      49      18      13      49
         36      37      32       4      31      44      14

    . Son anti-transposee :

         14      49      37      33      21       2       3
         44      13       3       5       4      41      11
         31      18      13      26      19      26      27
          4      49      38      42       1      19      23
         32      38      45      46      33      46      38
         37      31      35      29      26      34       7
         36      17      21      34       3      34       1

STOP Fin anti-transpose
```

FIGURE 10 – Anti-transposée d'une matrice d'ordre 7.