Premiers pas en Fortran

Le Fortran est le langage évolué le plus ancien : il date de 1954. Mais il a connu plusieurs évolutions, dont une, en 2003, qui a ajouté l'approche à objets. La dernière version est Fortran 2008 (norme ISO 1539-1:2010). Il reste très utilisé aujourd'hui dans le monde scientifique parce qu'il offre des outils très efficaces pour traiter les tableaux. Environ la moitié des programmes scientifiques actuels sont écrits en Fortran (dynamique des flux, chimie numérique, météo, etc.).

Il en existe une vingtaine d'implantations. Les principaux compilateurs open source ou gratuits sont G95, GFortran, Open Watcom et Silverfrost. Celui qui est utilisé ici est GFortran.

Sous Unix et Linux, on peut vérifier que le langage est installé et reconnu par le système d'exploitation avec la commande *which nom-du-compilateur* et on peut savoir de quelle version il s'agit avec la commande *nom-du-compilateur -v* ou *nom-du-compilateur --version*.

Pour installer GFortran sur un Linux de la famille Debian, on peut aller dans l'interface de commande et taper :

```
sudo apt-get install gfortran
```

Pour l'installer sur Mac OS avec le gestionnaire de paquets Homebrew, taper :

```
brew install gcc
```

Pour Windows, c'est plus compliqué [1]. Voir *https://gcc.gnu.org/wiki/GFortranBinaries#Windows*.

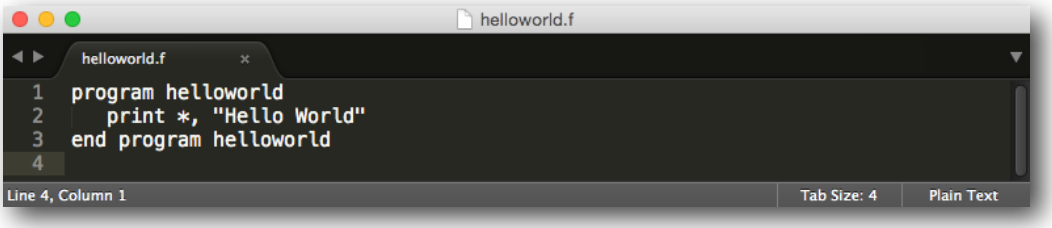
Premier programme

Comme avec la plupart des langages, un programme en Fortran s'écrit dans un fichier de texte au moyen d'un éditeur de texte. Il est utile d'employer un éditeur qui reconnaît le Fortran parce

[1] Le Fortran s'emploie peu sur les serveurs Windows. Les programmes écrits dans ce langage tournent en général sur de très gros serveurs Unix ou Linux.

que cela permet notamment d'avoir le surlignage du code et l'*autocompletion* (fonction de l'éditeur qui lui permet de compléter semi-automatiquement ce qu'on tape). Exemple : Photran (*http://www.eclipse.org/photran*)

Par tradition, le premier programme qu'on crée dans un langage est celui qui affiche à l'écran les mots *Hello world*. On lui donne le nom qu'on veut, ici *helloworld.f* :



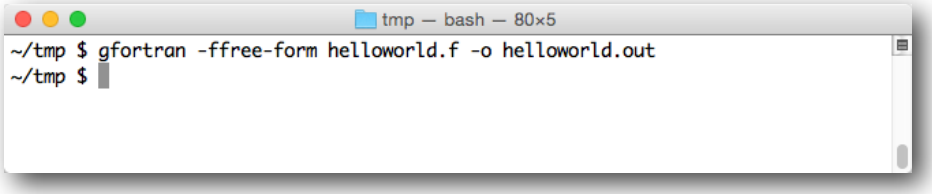
Cela peut se traduire ainsi en pseudo-code :

```
Début du programme helloworld
   Afficher « Hello World » ; l'étoile représente la sortie par défaut, en général l'écran
Fin du programme helloworld
```

Pour compiler ce code source, on utilise la commande suivante :

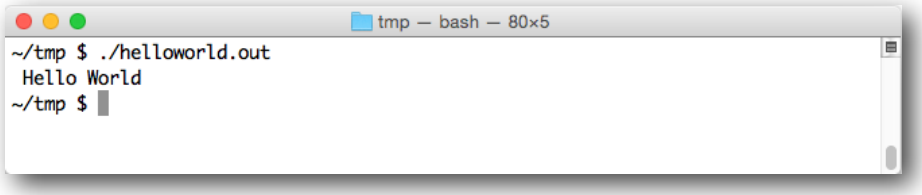
```
gfortran -ffree-form nom-du-code-source -o nom-du-programme-exécutable
```

Sous Windows, on donne l'extension *exe* pour l'exécutable. Sous Linux et Unix, on emploie généralement *out* (mais n'importe quel nom de fichier est possible, avec ou sans extension).



Jusqu'aux années 1980, la position du premier mot sur la ligne était gérée par le langage, mais ça n'a plus de sens aujourd'hui. Pour désactiver cette fonction, on emploie l'option *-ffree-form* avec GFortan (c'est spécifique à ce compilateur ; les autres utilisent d'autres méthodes).

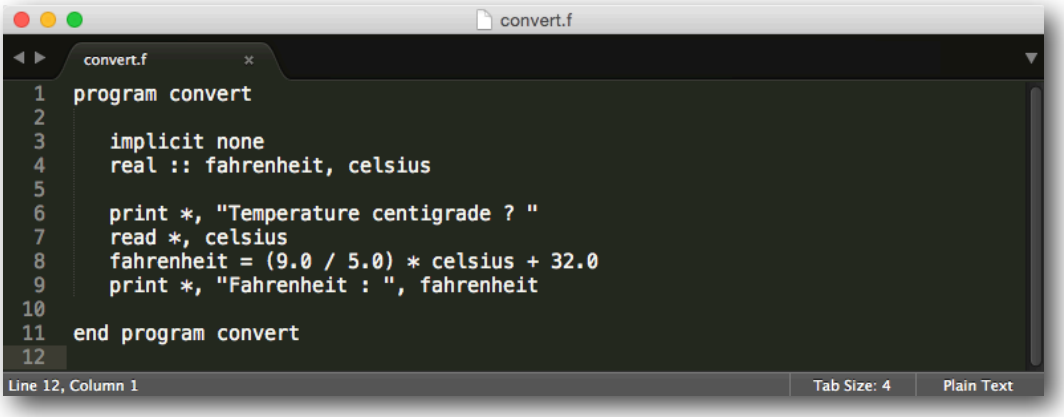
Pour exécuter le programme, on tape son nom :



Pour que le système d'exploitation trouve les fichiers sans difficulté, le plus simple est de tout faire dans un répertoire donné. Ici, c'est *~/tmp* (sous Unix et Linux, le tilde est une variable qui représente le répertoire personnel de l'utilisateur courant). Cela évite d'avoir à donner au système d'exploitation le chemin d'accès aux fichiers.

Conversion centigrade–fahrenheit

Voici un autre exemple : la conversion d'une température centigrade en température fahrenheit.



À la ligne 3, l'instruction *implicit none* spécifie que les variables doivent être déclarées avant d'être utilisées. En l'absence de cette instruction, il serait possible de créer une variable au moment où on lui assigne une valeur, ce qui peut être dangereux : si on fait une faute de frappe en tapant le nom d'une variable, le compilateur regarde cela comme la création d'une nouvelle variable et non comme une erreur.

Le code de ce programme peut se traduire ainsi :

```
Début du programme convert
   La déclaration des variable est obligatoire
   Déclaration des variables fahrenheit et celsius sous forme de nombres réels
   Afficher « Température centigrade ? » sur la sortie par défaut
   Lire la réponse de l'utilisateur et la mettre dans la variable celsius
   Faire le calcul de conversion et mettre le résultat dans la variable fahrenheit
   Afficher « Fahrenheit : » et la valeur de la variable fahrenheit
Fin du programme convert
```

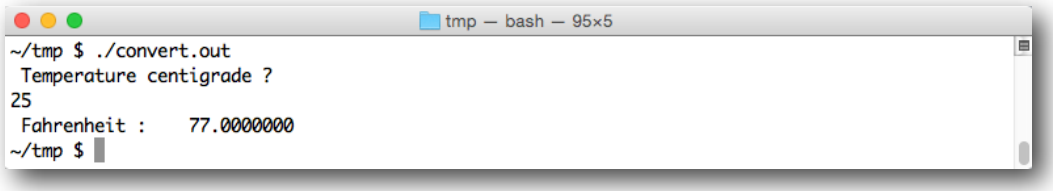
Dans une instruction d'entrée ou de sortie, on peut afficher plusieurs objets en les séparant par une virgule. C'est le cas à la ligne 9. L'instruction :

```
print *, "Fahrenheit : ", fahrenheit
```

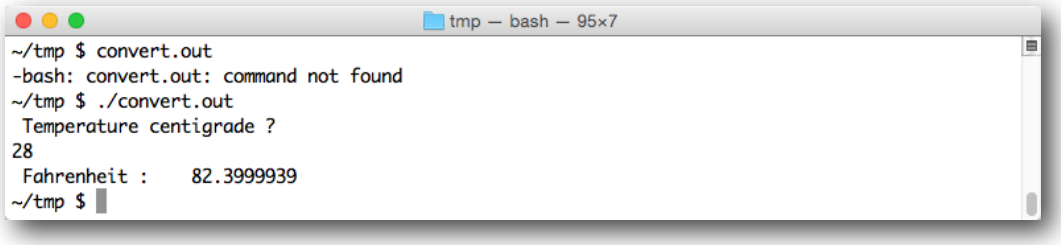
équivaut à :

```
print *, "Fahrenheit : "
print *, fahrenheit
```

On exécute le programme en tapant son nom :



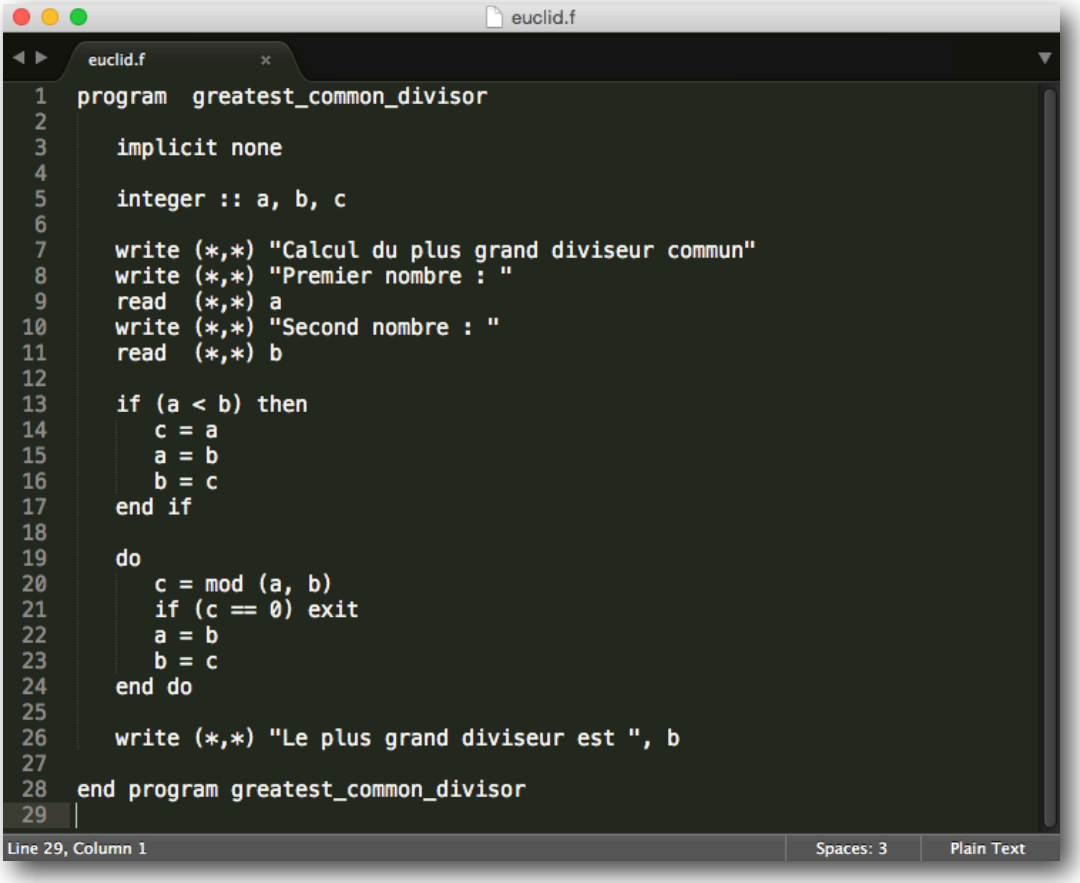
Remarque : sous Unix et Linux, on doit préfixer le nom d'un exécutable par un point et un slash (./) pour indiquer explicitement qu'il se trouve dans le répertoire courant. Sinon, le système d'exploitation ne le reconnaît pas :



C'est une mesure de sécurité : cela évite le risque de lancer par inadvertance un programme malveillant du même nom dissimulé quelque part ailleurs sur le système de fichiers.

Le plus grand diviseur commun

Un autre exemple est le calcul du plus grand diviseur commun de deux nombres.



Les lignes 13 à 17 servent à mettre la valeur la plus grande dans *a* et la plus petite dans *b*. Les lignes 19 à 24 effectuent le calcul en utilisant l'algorithme d'Euclide. On calcule d'abord le reste de la division de *a* par *b* au moyen de la fonction *mod*. Si le résultat est zéro, le calcul est

terminé : *b* est le plus grand diviseur commun ; et on sort de la boucle (*exit*). Sinon, *b* devient *a* et *c* devient *b*, et on refait le tour de la boucle (on remonte à la ligne 19).

Calcul matriciel en Fortran

Comme cela a déjà été dit, le Fortran convient très bien aux applications mathématiques et particulièrement à ce qui touche le calcul vectoriel et matriciel.

matrix.f

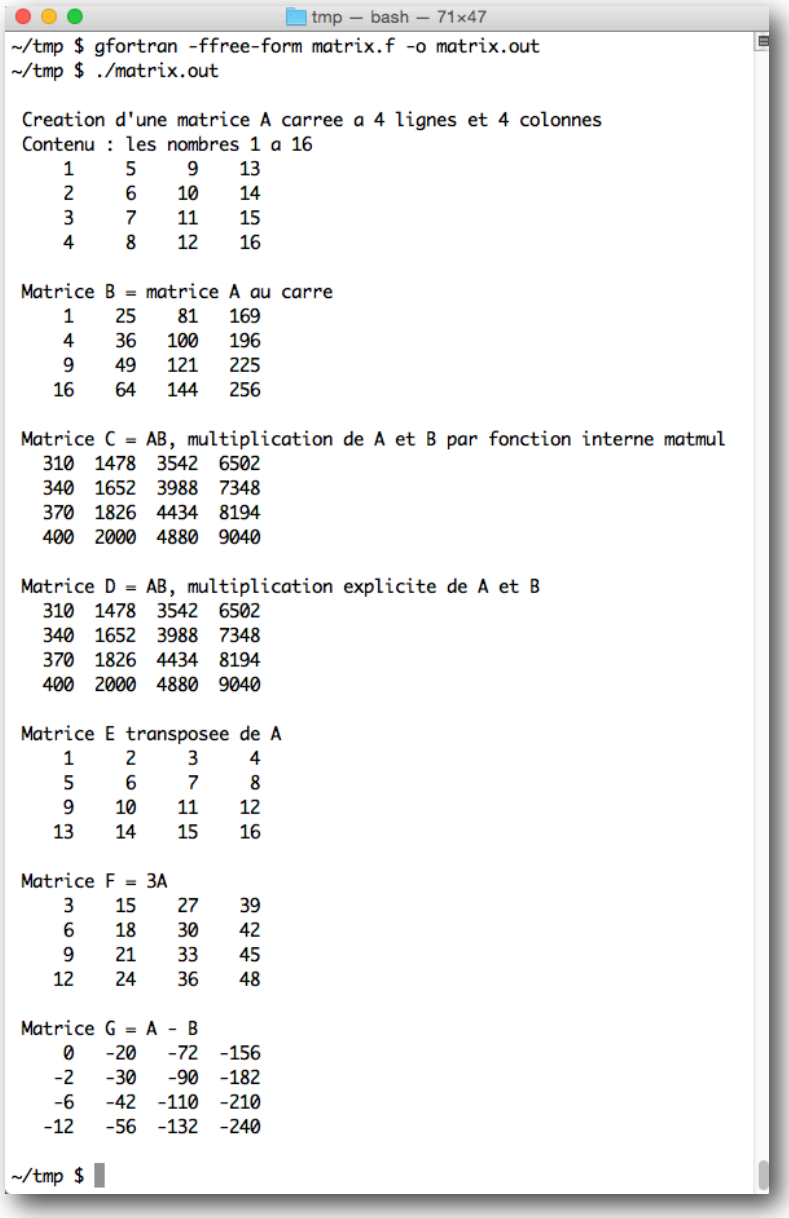
```
1  program matrices
2
3     implicit none
4
5     integer               :: i, j
6     integer, dimension (4,4) :: A, B, C, D, E, F, G
7
8     write (*,*)
9     print *, "Creation d'une matrice A carree a 4 lignes et 4 colonnes"
10    print *, "Contenu : les nombres 1 a 16"
11    A = reshape ( [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16], shape (A) )
12    call putmatrix (A,4)
13
14    print *, "Matrice B = matrice A au carre"
15    B = A * A
16    call putmatrix (B,4)
17
18    print *, "Matrice C = AB, multiplication de A et B par fonction interne matmul"
19    C = matmul (A, B)
20    call putmatrix (C,4)
21
22    print *, "Matrice D = AB, multiplication explicite de A et B"
23    do i = 1, 4
24       do j = 1, 4
25          D(i, j) = sum ( A(i,:)*B(:,j) )
26       end do
27    end do
28    call putmatrix (D,4)
29
30    print *, "Matrice E transposee de A"
31    E = transpose (A)
32    call putmatrix (E,4)
33
34    print *, "Matrice F = 3A"
35    F = 3 * A
36    call putmatrix (F,4)
37
38    print *, "Matrice G = A - B"
39    G = A - B
40    call putmatrix (G,4)
41
42    contains
43
44       subroutine putmatrix (M, n)
45
46          implicit none
47          integer                 :: n, i
48          integer, dimension (n, n) :: M
49          do i = 1, n
50             write (*," (4I6)") M(i,:)
51          end do
52          write (*,*)
53
54       end subroutine putmatrix
55
56 end program matrices
```

Line 57, Column 1 | Spaces: 3 | Plain Text

Le code de ce programme peut se traduire ainsi :

```
Début du programme matrices

   La déclaration des variable est obligatoire
   Déclarer les nombres entiers i et j
   Déclarer les matrices A, B, C, D, F, G (4 lignes, 4 colonnes ; éléments : nombres entiers)

   Afficher une ligne vide
   Afficher le message « Création d'une matrice A carrée à 4 lignes et 4 colonnes »
   Afficher le message « Contenu : les nombres 1 à 16 »
   Créer la matrice A avec les éléments 1 à 16
   Afficher le contenu de la matrice A

   Afficher « Matrice B = matrice A au carré »
   Mettre au carré la matrice A et stocker le résultat dans la matrice B
   Afficher le contenu de la matrice B

   Afficher « Matrice C = AB, multiplication de A et B par fonction interne matmul »
   Multiplier A et B et stocker le résultat dans C
   Afficher le contenu de la matrice C

   Afficher « Matrice D = AB, multiplication explicite de A et B »
   Multiplier A et B et stocker le résultat dans D
   Afficher le contenu de la matrice D

   Afficher « Matrice E, transposition de D par fonction interne transpose »
   Transposer D et stocker le résultat dans E
   Afficher le contenu de la matrice E

   Afficher « Matrice F = 3A »
   Multiplier A par 3 et stocker le résultat dans F
   Afficher le contenu de la matrice F

   Afficher « Matrice G = A – B »
   Soustraire A de B et stocker le résultat dans G
   Afficher le contenu de la matrice G

   Contains : début du bloc contenant les procédures (doit être placé à la fin du programme)

   Début de la procédure putmatrix

      La déclaration des variable est obligatoire
      Déclarer les variables n et i sous forme de nombres entiers
      Déclarer la matrice M (n lignes et colonnes ; éléments : nombres entiers)
      Boucle où i va de 1 à n
         Afficher le contenu de la matrice M (4I6 = 4 colonnes, entiers, 6 positions)
      Fin de la boucle
      Afficher une ligne vide

   Fin de la procédure putmatrix

Fin du programme matrices
```

Voici le résultat :



Dans les applications réelles, les éléments sont souvent des nombres réels. Dans ce cas, la ligne 6 devient :

```
real, dimension (4,4) :: A, B, C, D, E, F, G
```

la ligne 48 :

```
real, dimension (n, n) :: M
```

et la ligne 50 (par exemple) :

```
write (*," (4E15.6)") M(i,:)
```

Le formatage 4E15.6 signifie qu'on affiche sur 4 colonnes, que le nombre sera en forme exponentielle et que 15 positions seront utilisées, dont 6 pour le nombre, avec 3 positions vides en en-tête. Exemples :

```
1 = ␣0.100000E+01      256 = ␣0.256000E+03
```

Pour plus de détails, voir cette page du professeur Ching-Kuang Shene de la Michigan Tech : *http://www.cs.mtu.edu/~shene/COURSES/cs201/NOTES/format.html*.

Gestion d'une image

Ci-contre, voici un tableau de pixels. En le regardant de loin, on voit un émoticon très simple (une tête). Supposons qu'on veuille inverser cette image (la faire passer de blanc sur noir à noir sur blanc).

```
0000000000000
0000000000000
0001111111000
0111111111110
0110011100110
0111111111110
0110000000110
0011111111100
0000000000000
0000000000000
0000000000000
0000000000000
```

Pour cela, on peut utiliser les outils mis à disposition par le Fortran pour le calcul matriciel. Une seule instruction de dix caractères suffit pour effectuer l'opération. Dans le programme ci-dessous, c'est la ligne 22.

matrix_inverse.f

```
1  program matrix_inverse
2
3     implicit none
4
5     integer, dimension (13,13) :: A, B, C
6
7     A = reshape ( [0,0,0,0,0,0,0,0,0,0,0,0,0,  &
8                    0,0,0,0,0,0,0,0,0,0,0,0,0,  &
9                    0,0,0,0,0,0,0,0,0,0,0,0,0,  &
10                   0,0,0,1,1,1,1,1,1,1,0,0,0,  &
11                   0,1,1,1,1,1,1,1,1,1,1,1,0,  &
12                   0,1,1,0,0,1,1,1,0,0,1,1,0,  &
13                   0,1,1,1,1,1,1,1,1,1,1,1,0,  &
14                   0,1,1,0,0,0,0,0,0,0,1,1,0,  &
15                   0,0,1,1,1,1,1,1,1,1,1,0,0,  &
16                   0,0,0,0,0,0,0,0,0,0,0,0,0,  &
17                   0,0,0,0,0,0,0,0,0,0,0,0,0,  &
18                   0,0,0,0,0,0,0,0,0,0,0,0,0,  &
19                   0,0,0,0,0,0,0,0,0,0,0,0,0], shape (A) )
20
21    B = transpose (A)
22    C = abs (B - 1)
23
24    write (*,*)
25    write (*,*) "Dessin d'un emoticon tres simple"
26    write (*,*) "en blanc sur noir : "
27
28    call writematrix (B, 13)
29
30    write (*,*)
31    write (*,*) "Le meme dessin en noir sur blanc : "
32
33    call writematrix (C, 13)
34
35    contains
36
37       subroutine writematrix (M, n)
38
39          implicit none
40          integer                    :: n, i
41          integer, dimension (n, n) :: M
42          do i = 1, n
43             write (*," (13I2)") M(i,:)
44          end do
45          write (*,*)
46
47       end subroutine writematrix
48
49 end program matrix_inverse
```

Line 50, Column 1 | Tab Size: 4 | Plain Text

Le code de ce programme peut se traduire ainsi :

```
Début du programme matrix_inverse

   La déclaration des variable est obligatoire
   Déclarer les matrices A, B et C (13 lignes et colonnes ; éléments : nombres entiers)

   Créer la matrice A dans un format facile à lire

   Transposer la matrice A dans B
   Soustraire 1 de chaque élément de B puis prendre la valeur absolue de chacun d'eux

   Afficher une ligne vide puis un message (lignes 24 et 25)
   Afficher la matrice B

   Afficher une ligne vide puis un message (lignes 30 et 31)
   Afficher la matrice C

   Contains : début du bloc contenant les procédures (doit être placé à la fin du programme)

   Début de la procédure writematrix

      La déclaration des variable est obligatoire
      Déclarer les variables n et i sous forme de nombres entiers
      Déclarer la matrice M (n lignes et colonnes ; éléments : nombres entiers)
      Boucle où i va de 1 à n
         Afficher le contenu de la matrice M (13I2 = 13 colonnes, entiers, 2 positions)
      Fin de la boucle
      Afficher une ligne vide

   Fin de la procédure writematrix

Fin du programme matrix_inverse
```

Une fois le programme écrit, on le compile avec la commande qu'on a vue à la page 2 :

```
gfortran -ffree-form matrix_inverse.f -o matrix_inverse.out
```

en remplaçant *out* par *exe* si on est sous Windows.

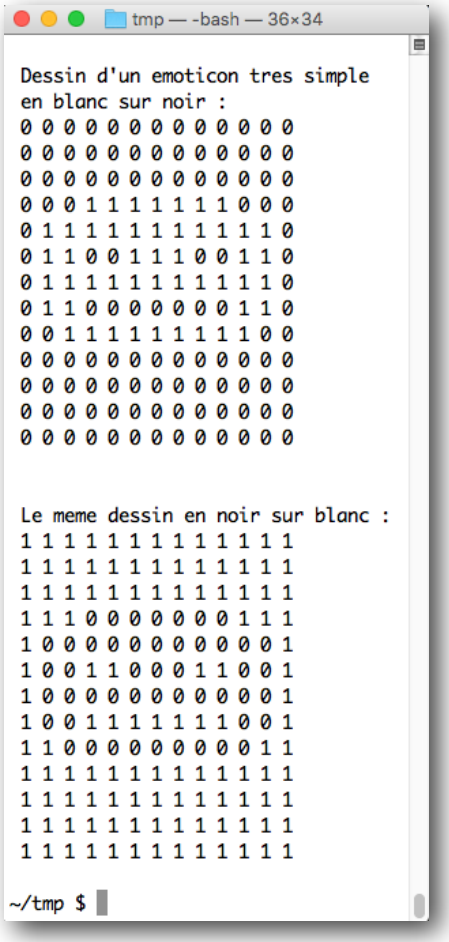
Puis on l'exécute sous Unix ou Linux en tapant :

```
./matrix_inverse.out
```

ou, sous Windows :

```
.\matrix_inverse.exe
```

Pour le résultat, voir l'illustration ci-contre.



On voit que des fonctions comme *transpose* et *matmul* simplifient considérablement les opérations sur les matrices. Il en existe beaucoup d'autres, qui sont notamment fournies dans des bibliothèques de tierces-parties. Exemple : LAPACK (*http://www.netlib.org/lapack*).

Si le Fortran est un langage efficace et populaire pour les applications scientifiques, c'est pour plusieurs raisons :

- Il existe un grand nombre de fonctions mathématiques écrites pour ce langage. Comme le dit une maxime connue, le meilleur code est le code qu'on n'écrit pas soi-même et qui a été testé dans de nombreux programmes.
- La syntaxe du Fortran est sobre (peu de caractères spéciaux, pas d'accolades, de points-virgules à chaque fin de lignes, etc.). Les matrices sont des objets complexes mais, en Fortran, le code nécessaire pour les manipuler est en général à la fois compact et simple. On le voit dans le programme de la page 5 si on compare la ligne 19 (typique du Fortran) avec les lignes 23 à 27 (typiques d'un autre langage).
- Grâce à cette simplicité, les erreurs de programmation tendent à être plus faciles à repérer et à corriger.
- Il faut beaucoup moins de temps pour maîtriser le Fortran que le C/C++, qui est l'autre langage populaire pour les grosses applications scientifiques.
- En règle générale, un mathématicien ou un physicien qui n'est pas informaticien peut écrire un programme fiable en Fortran, mais pas en C/C++ [2]. Il peut aussi comprendre un programme en Fortran plus facilement que le même programme écrit en C/C++.

[2] Le C/C++ est un langage très puissant mais difficile à maîtriser. Seule une minorité d'informaticiens a les compétences nécessaires pour écrire une application C/C++ qui soit à la fois fiable et efficace.

Annexe

En Fortran, les éléments d'une matrice se rangent colonne par colonne et non ligne par ligne :

$$\mathbf{M} = \begin{bmatrix} 1 & 5 & 9 & 13 \\ 2 & 6 & 10 & 14 \\ 3 & 7 & 11 & 15 \\ 4 & 8 & 12 & 16 \end{bmatrix}$$

Par contre, en mathématiques, les éléments sont notés ligne par ligne. Une matrice est symbolisée par une lettre majuscule en caractères gras et un élément qui se trouve à l'intersection entre la ligne *i* et la colonne *j* est noté A_{ij} :

$$\mathbf{A} = \begin{bmatrix} A_{11} & A_{12} & \dots & A_{1m} \\ A_{21} & A_{22} & \dots & A_{2m} \\ \dots & \dots & \dots & \dots \\ A_{n1} & A_{n2} & \dots & A_{nm} \end{bmatrix}$$

Un élément peut être de tout type numérique ($\mathbb{N}$, $\mathbb{Z}$, $\mathbb{R}$, etc.). On l'appelle un élément, un terme, un coefficient ou une composante.

Si $n = 1$, c'est une matrice ligne à m colonnes.

Si $m = 1$, c'est une matrice colonne à n lignes.

Si $n = m$, c'est une matrice carrée.

Si les éléments qui ne se trouvent pas sur la diagonale D_{11}, D_{22},... D_{nn} sont nuls, c'est une matrice diagonale :

$$\mathbf{D} = \begin{bmatrix} D_{11} & 0 & 0 \\ 0 & D_{22} & 0 \\ 0 & 0 & D_{33} \end{bmatrix}$$

Si les éléments de la diagonale I_{11}, I_{22}, I_{33},... ont la valeur 1 et les autres la valeur 0, c'est un cas particulier de matrice diagonale appelé matrice identité ou matrice unité d'ordre *n*. On peut indiquer l'ordre en indice (ici 4).

$$\mathbf{I_4} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Si les éléments du triangle inférieur gauche d'une matrice carrée sont nuls, c'est une matrice triangulaire supérieure. Exemple :

$$\mathbf{U} = \begin{bmatrix} U_{11} & U_{12} & U_{13} & U_{14} \\ 0 & U_{22} & U_{23} & U_{24} \\ 0 & 0 & U_{33} & U_{34} \\ 0 & 0 & 0 & U_{44} \end{bmatrix}$$

Si les éléments du triangle supérieur droit d'une matrice carrée sont nuls, c'est une matrice triangulaire inférieure. Exemple :

$$\mathbf{L} = \begin{bmatrix} L_{11} & 0 & 0 & 0 \\ L_{21} & L_{22} & 0 & 0 \\ L_{31} & L_{32} & L_{33} & 0 \\ L_{41} & L_{42} & L_{43} & L_{44} \end{bmatrix}$$