Les bases : exercices corrigés en F
Corrigé

Consignes : Les exercices 2, 4, 6 et 7 sont facultatifs. Dans le cas de l'exercice 5, on pourra se limiter au cas des puissances positives (x^n avec $n \geq 0$).

Objectifs

- Raffiner des problèmes simples ;
- Écrire quelques algorithmes simples ;
- Savoir utiliser les types de base ;
- Savoir utiliser les instructions « élémentaires » : d'entrée/sortie, affichage, bloc...
- Manipuler les conditionnelles ;
- Manipuler les répétitions.

Exercice 1 : Résolution d'une équation du second degré

Soit l'équation du second degré $ax^2 + bx + c = 0$ où a, b et c sont des coefficients réels. Écrire un programme qui saisit les coefficients et affiche les solutions de l'équation.
Indication : Les solutions sont cherchées dans les réels. Ainsi, dans le cas général, en considérant le discriminant $\Delta = b^2 - 4ac$, l'équation admet comme solutions analytiques :

$$\begin{cases} \Delta < 0 & \text{pas de solutions réelles.} \\ \Delta = 0 & \text{une solution double : } \frac{-b}{2a} \\ \Delta > 0 & \text{deux solutions : } x_1 = \frac{-b-\sqrt{\Delta}}{2a} \text{ et } x_2 = \frac{-b+\sqrt{\Delta}}{2a} \end{cases}$$

Quelles sont les solutions de l'équation si le coefficient a est nul ?
Solution : Voici un raffinage possible :

```
1 R0 : Résoudre l'équation du second degré
2
3 R1 : Raffinage De « Résoudre l'équation du second degré »
4      |  Saisir les 3 coefficients              a, b, c: out RÉEL
5      |                          -- les coefficients de l' équation
```

```
6       |  Calculer et afficher les solutions
7
8   R2 : Raffinage De « Saisir les 3 coefficients »
9       |  Écrire("Entrer les valeurs de a, b et c : ")
10      |  Lire(a, b, c)
11
12  R2 : Raffinage De « Calculer et afficher les solutions »
13      |  Si équation du premier degré Alors
14      |      Résoudre l'équation du premier degré bx + c = 0
15      |  Sinon
16      |      Calculer le discriminant
17      |          --> delta: RÉEL          -- le discriminant de ax^2 + bx + c = 0
18      |      Si discriminant nul Alors            -- une solution double
19      |          Écrire("Une solution double : ", - B / (2 * A))
20      |      SinonSi discriminant positif Alors  -- deux solutions distinctes
21      |          Écrire("Deux solutions : ")
22      |          Écrire((-B + √Delta) / (2*A))
23      |          Écrire((-B - √Delta) / (2*A))
24      |      Sinon       { discriminant négatif }        -- pas de solution dans IR
25      |          Écrire("Pas de solution réelle")
26      |      FinSi
27      |  FinSi
28
29  R3 : Raffinage De « Résoudre l'équation du premier degré bx + c = 0 »
30      |  Si B = 0 Alors
31      |      Si C = 0 Alors
32      |          Écrire("Tout IR est solution")
33      |      Sinon
34      |          Écrire("Pas de solution")
35      |      FinSi
36      |  Sinon
37      |      Écrire("Une solution : ", - C / B)
38      |  FinSi
```

Remarque : Notons que nous n'avons pas un premier niveau de raffinage en trois étapes, *saisir*, *calculer*, *afficher* car il est difficile de dissocier les deux dernières étapes car la forme des racines de l'équation du second est très variable et ne peut pas être facilement capturée par un type (deux solution distinctes, une solution double, pas de solution, une infinité de solution). Aussi, nous avons regroupé calcul et affichage.

Et l'algorithme correspondant :

```
1   Algorithme second_degré
2
3       -- Résoudre l'équation du second degré
4
5   Variables
6       a, b, c: Réel       -- les coefficients de l'équation
7       delta: Réel         -- le discriminant
8
9   Début
```

```
10      -- Saisir les 3 coefficients
11      Écrire("Entrer les valeurs de a, b et c : ")
12      Lire(a, b, c)
13
14      -- Calculer et afficher les solutions
15      Si a = 0 Alors              -- équation du premier degré
16          -- Résoudre l'équation du premier degré bx + c = 0
17          Si B = 0 Alors          -- équation constante
18              Si C = 0 Alors
19                  Écrire("Tout IR est solution")
20              Sinon
21                  Écrire("Pas de solution")
22              FinSi
23          Sinon                   -- équation réellement du premier degré
24              Écrire("Une solution : ", - C / B)
25          FinSi
26
27      Sinon                       -- équation réellement du second degré
28          -- Calculer le discriminant delta
29          delta <- b*b - 4*a*c
30
31          -- Déterminer et afficher les solutions
32          Si delta = 0 Alors              -- une solution double
33              Écrire("Une solution double : ", - B / (2 * A))
34          SinonSi delta > 0 Alors -- deux solutions distinctes
35              Écrire("Deux solutions : ")
36              Écrire((-B + sqrt(delta)) / (2*A))
37              Écrire((-B - sqrt(delta)) / (2*A))
38          Sinon   { discriminant négatif }        -- pas de solution dans IR
39              Écrire("Pas de solution réelle")
40          FinSi
41      FinSi
42  Fin.
```

```
1   !*********************************************************************
2   !*  Auteur   : Denis Barreteau <Denis.Barreteau@ensiacet.fr>
3   !*  Version  : 1.3
4   !*  Revision : Xuan Meyer < XuanMi.Meyer@ensiacet.fr>
5   !*
6   !*  Objectif : Resolution de l'equation du second degre.
7   !*             Approche classique
8   !*********************************************************************
9
10  ! Demande à l'utilisateur les coefficients a, b et c de l'equation du
11  ! second degre  ax^2 + bx + c* , calcule et affiche les solutions
12  ! reelles de cette equation.
13
14  PROGRAM main
15
16      ! Declaration des variables (F oblige)
17      REAL:: a, b, c        ! coefficients de l'equation
```

```
18      REAL:: delta
19      ! Poser le problème
20      PRINT*,"Resolution␣de␣l'equation␣du␣second␣degre␣:"
21      PRINT*,"a.x2␣+␣b.x␣+␣c␣=␣0"
22
23      ! Saisir la valeur des coefficients
24      PRINT*,"Donnez␣la␣valeur␣des␣coefficients␣a,␣b␣et␣c␣:␣"
25      READ*, a,b,c
26
27      ! Resoudre l'equation et afficher les resultats
28      if (a == 0.0) THEN        !{ equation du premier degre }
29          ! resoudre l'equation du premier degre
30          if (b == 0.0) THEN
31              if (c == 0.0) THEN
32                  PRINT*,"Tout␣les␣reels␣sont␣solutions"
33              else
34                  PRINT*,"Pas␣de␣solution"
35              ENDIF
36          else
37              PRINT*,"Une␣seule␣solution␣:", -c/b
38          ENDIF
39      else
40          ! cas general
41          delta = b*b - 4.0*a*c              ! le discriminant
42          if (delta > 0)THEN
43              ! deux solutions reelles
44              PRINT*,"Deux␣solutions␣:␣"
45              PRINT*,"solution␣1␣=␣", (-b + sqrt(delta))/2.0/a
46              PRINT*,"solution␣2␣=␣", (-b - sqrt(delta))/2.0/a
47          else if (delta == 0.0) THEN
48              ! une solution double
49              PRINT*,"Une␣solution␣double␣:␣", (-b/2.0/a)
50          else
51              ! pas de solutions reelles
52              PRINT*,"Pas␣de␣solutions␣reelles"
53          ENDIF
54      ENDIF
55      PRINT*
56
57  END PROGRAM main
```

Exercice 2 : Résolution numériquement correcte d'une équation du second degré

Utiliser les formules analytiques de l'exercice 1 pour calculer les solutions réelles de l'équation du second degré n'est pas numériquement satisfaisant. En effet, si la valeur de b est positive et proche de celle de $\sqrt{\Delta}$, le calcul de x_2 comportera au numérateur la soustraction de deux nombres voisins ce qui augmente le risque d'erreur numérique.

Par exemple, considèrons l'équation $x^2 + 62,10x + 1 = 0$ dont les solutions sont approximativement :

$$x_1 = -62,08390$$
$$x_2 = -0,01610723$$

Dans cette équation, la valeur de b^2 est bien plus grande que le produit $4ac$ et le calcul de Δ conduit donc à un nombre très proche de b. Dans les calculs qui suivent, on ne tient compte que des 4 premiers chiffres significatifs.

$$\sqrt{\Delta} = \sqrt{(62,10)^2 - 4 \times 1 \times 1} = \sqrt{3856 - 4} = 62,06$$

On obtient alors :

$$x_2 = \frac{-62,10 + 62,06}{2} = -0,02$$

L'erreur relative sur x_2 induite est importante :

$$\frac{|-0,01611 + 0,02|}{|-0,01611|} = 0,24 \quad \text{(soit 24\% d'erreur).}$$

En revanche, pour le calcul de x_1 l'addition de deux nombres pratiquement égaux ne pose pas de problème et les calculs conduisent à un erreur relative faible $(3,210^{-4})$.

Pour diminuer l'erreur numérique sur x_2, il suffirait de réorganiser les calculs :

$$x_2 = \left(\frac{-b + \sqrt{\Delta}}{2a}\right)\left(\frac{-b - \sqrt{\Delta}}{-b - \sqrt{\Delta}}\right) = \frac{b^2 - \Delta}{2a(-b - \sqrt{\Delta})} = \frac{-2c}{b + \sqrt{\Delta}}$$

Mais il existe une solution plus simple qui consiste à calculer d'abord x_1 par la formule classique puis en déduire x_2 en considérant que le produit des racines est égal à c/a ($x_1 x_2 = c/a$).

Conclusion : En pratique, si b est négatif, on calcule d'abord la racine $\frac{-b+\sqrt{\Delta}}{2a}$, si b est positif, on calcule la racine $\frac{-b-\sqrt{\Delta}}{2a}$. L'autre racine est calculée à l'aide du produit c/a.

Écrire le programme qui calcule les racines réelles de l'équation du second degré en s'appuyant sur cette formule.

Solution :

```
1 !*****************************************************************
2 !*  Auteur   : Denis Barreteau <Denis.Barreteau@ensiacet.fr>
3 !*  Version  : 1.4
4 !*  Revision : Xuan Meyer < XuanMi.Meyer@ensiacet.fr>
5 !*
6 !*  Objectif : Resolution de l'equation du second degre.
7 !*             Approche numerique
8 !*****************************************************************
```

```
9
10 ! Demande à l'utilisateur les coefficients a, b et c de l'equation du
11 ! second degre  ax^2 + bx + c* , calcule et affiche les solutions
12 ! reelles de cette equation.
13
14 PROGRAM main
15
16     ! Declaration des variables (F oblige)
17     REAL:: a, b, c        ! coefficients de l'equation
18     REAL:: delta
19     ! Poser le problème
20     PRINT*,"Resolution de l'equation du second degre :"
21     PRINT*,"a.x2 + b.x + c = 0"
22
23     ! Saisir la valeur des coefficients
24     PRINT*,"Donnez la valeur des coefficients a, b et c : "
25     READ*, a,b,c
26
27     ! Resoudre l'equation et afficher les resultats
28     IF (a == 0.0) THEN        !{ equation du premier degre }
29         ! resoudre l'equation du premier degre
30         IF (b == 0.0) THEN
31             IF (c == 0.0) THEN
32                 PRINT*,"Tout les reels sont solutions"
33             ELSE
34                 PRINT*,"Pas de solution"
35             ENDIF
36         ELSE
37             PRINT*,"Une seule solution :", -c/b
38         ENDIF
39     ELSE
40         ! cas general
41         delta = b*b - 4.0*a*c              ! le discriminant
42         IF (delta > 0)THEN
43             ! deux solutions reelles
44             PRINT*,"Deux solutions : "
45             IF(b<0) THEN   ! suivant le signe de b le calcul de -b + sqrt(delta)
46                            !est plus ou moins precis
47                  PRINT*,"solution 1 = ", (-b + sqrt(delta))/2.0/a
48                  PRINT*,"solution 2 = ", c/a/((-b + sqrt(delta))/2.0/a)
49             ELSE
50                  PRINT*,"solution 1 = ", (-b - sqrt(delta))/2.0/a
51                  PRINT*,"solution 2 = ", c/a/((-b - sqrt(delta))/2.0/a)
52             ENDIF
53         ELSEIF (delta == 0.0) THEN
54             ! une solution reelle::
55             PRINT*,"Une solution double : ", (-b/2.0/a)
56         ELSE
57             ! pas de solutions reelles
58             PRINT*,"Pas de solutions reelles"
59         ENDIF
60     ENDIF
```

```
61       PRINT*
62
63   END PROGRAM main
```

Exercice 3 : Le nombre d'occurrences du maximun

Compléter le programme de l'exercice 2 (Exercices résolus en F, Semaine 1) qui calcule les statistiques sur une série de valeurs réelles pour qu'il affiche également le nombre d'occurrences de la plus grande valeur, c'est-à-dire le nombre de valeurs de la série qui correspondent à la valeur maxiamle.

Par exemple, pour la série 1 2 3 1 2 3 3 2 3 1 0, le max est 3 et il y a quatre occurrences de 3. Dans la série 1 2 3 3 3 3 1 2 4 1 2 3 0, il y a une seule occurrence du maximun qui est 4.

Solution : Le principe est d'ajouter une nouvelle variable d'accumulation, `nb_max`, qui comptabilise le nombre d'occurrences du max rencontrées. Elle est initialisée à 1 sur la première valeur. Elle est remise à 1 dès qu'un nouveau maximum est identifié. Elle est augmentée de 1 si la valeur lue est égale au maximum précédent.

```
1  !*****************************************************************************
2  !*  Auteur  :  Denis Barreteau <Denis.Barreteau@ensiacet.fr>
3  !*  Version :  1.1
4  !*  Revision : Xuan Meyer < XuanMi.Meyer@ensiacet.fr>
5  !*
6  !*  Objectif :
7  !*        Afficher la moyenne, la plus grande et la plus petite valeur et le
8  !*        nombre d'occurences de la plus grande valeur d'une serie de valeurs
9  !*        reelles lues au clavier.
10 !*****************************************************************************
11
12 PROGRAM main
13
14     REAL:: x         ! un reel lu au clavier
15     INTEGER:: nb     ! le nombre de valeurs lues de la serie
16     REAL:: somme     ! la somme des valeurs lues de la serie
17     REAL:: min       ! la plus petite des valeurs lues de la serie
18     REAL:: max       ! la plus grande des valeurs lues de la serie
19     REAL:: nb_max    ! le nombre d'occurrences du max
20
21     ! afficher la consigne
22     PRINT*,"Donnez,en␣colonne,␣une␣serie␣d'entiers␣qui␣se␣termine␣par␣0."
23
24     ! lire le premier reel
25     READ*,x
26
27     IF (x == 0) THEN        ! Pas de valeur dans la serie
28         PRINT*,"La␣serie␣est␣vide.␣␣"
29         PRINT*,"Les␣statistiques␣demandees␣n'ont␣pas␣de␣sens␣!"
30     ELSE
31         ! initialiser les variables statistiques avec x
32         max = x
33         min = x
34         somme = x
35         nb = 1
36         nb_max = 1
37
38         ! lire une nouvelle valeur x
```

```
39          READ*,x
40
41          DO
42              ! mettre à jour les variables statistiques
43              nb=nb+1
44              somme = somme + x
45              IF (x > max) THEN
46                  max = x
47                  nb_max = 1
48              ELSEIF (x == max) THEN
49                  nb_max=nb_max+1
50              ELSEIF (x < min) THEN
51                  min = x
52              ENDIF
53              ! lire une nouvelle valeur x
54              READ*, x
55              IF(x==0)EXIT
56          ENDDO
57          ! afficher les statistiques
58          PRINT*,"Moyenne = ", somme / nb
59          PRINT*,"Plus petite valeur = ", min
60          PRINT*,"Plus grande valeur = ", max
61          PRINT*,"Nb d'occurrences du max = ", nb_max
62      ENDIF
63
64  END PROGRAM main
65
66
67  !                          moyenne, min, max,         nb_max
68  !     1 2 3 0          -->         2,         1,         3         1
69  !     2 -2 0          -->         0,         -2,         2         1
70  !     -4 -2 0          -->         -3,         -4,         -2         1
71  !     13 0          -->         13,         13,         13         1
72  !     0                    -->         Non defini
73  !     1 3 1 3 0          -->         2         1         3         2
74  !     3 3 3 3 0          -->         3         3         3         4
```

Exercice 4 : Nombres amis

Deux nombres N et M sont amis si la somme des diviseurs de M (en excluant M lui-même) est égale à N et la somme des diviseurs de N (en excluant N lui-même) est égale à M.

Écrire un programme qui affiche tous les couples (N, M) de nombres amis tels que $0 < N < M \leq MAX$, MAX étant lu au clavier.

Indication : Les nombres amis compris entre 1 et 100000 sont (220, 284), (1184, 1210), (2620, 2924), (5020, 5564), (6232, 6368), (10744, 10856), (12285, 14595), (17296, 18416), (66928, 66992), (67095, 71145), (63020, 76084), (69615, 87633) et (79750, 88730).

Solution : L'idée est de faire un parcours de tous les couples possibles et de se demander s'ils forment un couple de nombres amis ou non. Pour un n et un m donnés, il s'agit alors de calculer la somme de leurs diviseurs. C'est en fait deux fois le même problème. Il s'agit de calculer la somme des diviseurs d'un entier p. Naïvement, on peut regarder si les entiers de 2 à $p-1$ sont des diviseurs de p.

Une première optimisation consiste à remarquer que p n'a pas de diviseurs au delà de $p/2$. En fait, il est plus intéressant de remarquer que si i est diviseur de p alors p/i est également un diviseur de p. Il suffit donc de ne considérer comme diviseurs potentiels que les entiers compris dans l'intervalle $2..\sqrt{p}$. Il faut toutefois faire attention à ne pas comptabiliser deux fois $\sqrt{p}$.

```
1  R0 : Afficher les couples de nombres amis (N, M) avec 1 < N < M <= Max
2
3  R1 : Raffinage De « R0 »
4       | Pour m <- 2 JusquÀ m = Max Faire
5       |    | Pour n <- 2 JusquÀ n = m - 1 Faire
6       |    |    | Si n et m amis Alors
7       |    |    |    | Afficher le couple (n, m)
8       |    |    | FinSi
9       |    | FinPour
10      | FinPour
11
12 R2 : Raffinage De « n et m amis »
13      | Résultat <- (somme des diviseurs de N) = M
14      |          Et (somme des diviseurs de M) = N
15
16 R3 : Raffinage De « somme des diviseurs de p »
17      | somme <- 1
18      | Pour i <- 2 JusquÀ i = racine_carrée(p) - 1 Faire
19      |    | Si i diviseur de p Alors
20      |    |    | somme <- somme + i + (p Div i)
21      |    | FinSi
22      | FinPour
23      | Si p est un carré parfait Alors
24      |    | somme <- somme + i
25      | FinSi
```

L'algorithme est alors le suivant. Notez quelques optimisations qui font que l'algorithme de respecte pas complètement le raffinage.

```
1  Algorithme nb_amis
2
```

```
3     -- Afficher les couples de nombres amis (N, M) avec 1 < N < M <= MAX
4
5   Variables
6       n, m: Entier        -- pour représenter les couples possibles
7       somme_n: Entier     -- somme des diviseurs de n
8       somme_m: Entier     -- somme des diviseurs de m
9
10  Début
11      Pour m <- 2 JusquÀ m = MAX Faire
12          -- calculer la somme des divieurs de m
13          -- Remarque : on peut déplacer cette étape à l'extérieur de la
14          -- boucle car elle ne dépend pas de n (optimisation).
15          somme_m <- 1
16          Pour i <- 2 JusquÀ i = racine_carrée(m) - 1 Faire
17              Si i diviseur de m Alors
18                  somme_m <- somme_m + i + (m Div i)
19              FinSi
20          FinPour
21          Si m est un carré parfait Alors
22              somme_m <- somme_m + i
23          FinSi
24
25
26          Pour n <- 2 JusquÀ n = m - 1 Faire
27              -- calculer la somme des divieurs de n
28              somme_n <- 1
29              Pour i <- 2 JusquÀ i = racine_carrée(n) - 1 Faire
30                  Si i diviseur de n Alors
31                      somme_n <- somme_n + i + (n Div i)
32                  FinSi
33              FinPour
34              Si n est un carré parfait Alors
35                  somme_n <- somme_n + i
36              FinSi
37
38              -- déterminer si n et m sont amis
39              Si (somme_n = m) Et (somme_m = n) Alors { n et m sont amis }
40                  Afficher le couple (n, m)
41              FinSi
42          FinPour
43      FinPour
44
45  Fin.
```

```
1   !*********************************************************************
2   !*  Auteur   : Denis Barreteau <Denis.Barreteau@ensiacet.fr>
3   !*  Version  : 1.1
4   !*  Revision : Xuan Meyer < XuanMi.Meyer@ensiacet.fr>
5   !*
6   !*  Objectif : Afficher les couples de nombres amis (N, M)
7   !*             avec 1 < N < M <= MAX.
8   !*             version simple
```

```
9  !*****************************************************************
10
11 PROGRAM main
12     INTEGER:: nmax           ! borne superieure
13     INTEGER:: n, m           ! pour representer les couples possibles
14     INTEGER:: somme_n        ! somme des diviseurs de n
15     INTEGER:: somme_m        ! somme des diviseurs de m
16     INTEGER:: racine         ! la racine carree d'un nombre
17     INTEGER:: i              ! variable de boucle
18
19     PRINT*, "valeur de Max"
20     READ*, nmax
21     DO m = 2,nmax
22         ! calculer la somme des divieurs de m
23         ! Remarque : on peut deplacer cette etape a l'exterieur de la boucle
24         !** car elle ne depend pas de n (optimisation).
25         somme_m = 1
26         racine = sqrt(REAL(m))
27         DO i = 2,racine
28             IF (m/i*i == m)  somme_m = somme_m + i + (m / i)
29             ! i diviseur de m
30         ENDDO
31         IF (m == racine * racine)  somme_m = somme_m - racine
32         ! m est un carre parfait
33
34         DO n = 2,m
35             ! calculer la somme des divieurs de n
36             somme_n = 1
37             racine = sqrt(REAL(n))
38             DO i = 2,racine
39                 IF (n / i*i == n)  somme_n = somme_n + i + (n / i)
40                 !  i diviseur de n
41             ENDDO
42             IF (n == racine * racine) somme_n = somme_n - racine
43             ! n est un carre parfait
44
45
46
47             ! determiner si n et m sont amis
48             IF (somme_n == m .AND. somme_m == n)PRINT*, n, m         ! n et m sont ami
49
50
51          ENDDO
52     ENDDO
53
54 END PROGRAM main
```

Une solution plus efficace consiste à constater que pour un entier M compris entre 2 et MAX, le seul nombre ami possible est la somme de ses diviseurs que l'on note `somme_m`. Il reste alors à vérifier si la somme des diviseurs de `somme_m` est égale à M pour savoir si `somme_m` et M sont amis. Le fait de devoir afficher les couples dans l'ordre croissant nous conduit à ne considérer

que les sommes de diviseurs inférieures à M.

À titre indicatif, pour Max = 1000000, ce deuxième algorithme termine en 1 minute 23 alors que dans le même temps, seuls les sept premiers résultats sont trouvés avec le premier algorithme. Les solutions suivantes sont trouvées au bout d'une minute 32 secondes, 2 minutes 46, 5 minutes 35, 20 minutes, etc.

```
R0 : Afficher les couples de nombres parfaits (N, M) avec 1 < N < M <= Max

R1 : Raffinage De « R0 »
       | Pour m <- 2 JusquÀ m = Max Faire
       |   | Déterminer la somme des diviseurs de m
       |   |                                        m: in ; somme_m: out Entier
       |   | n <- somme_m
       |   | Si n < m Alors       { n est un nb amis potentiel }
       |   |   | Déterminer la somme des diviseurs de n (somme_n)
       |   |   | Si somme_n = m Alors      { somme_m et m amis }
       |   |   |   | Afficher le couple (n, m)
       |   |   | FinSi
       |   | FinPour
       | FinPour

R2 : Raffinage De « somme des diviseurs de p »
       | somme <- 1
       | Pour i <- 2 JusquÀ i = racine_carrée(p) - 1 Faire
       |    | Si i diviseur de p Alors
       |    |    | somme <- somme + i + (p Div i)
       |    | FinSi
       | FinPour
       | Si p est un carré parfait Alors
       |    | somme <- somme + i
       | FinSi
```

Voici l'algorithme correspondant.

```
Algorithme nb_amis

   -- Afficher les couples de nombres amis (N, M) avec 1 < N < M <= MAX

Variables
    m: Entier           -- pour parcourir les entiers de 2 à MAX
    n: Entier           -- l'amis candidat
    somme_m: Entier     -- somme des diviseurs de m
    somme_n: Entier     -- somme des diviseurs de somme_m correspondant à n

Début
    Pour m <- 2 JusquÀ m = MAX Faire
        -- Déterminer la somme des divieurs de m
        somme_m <- 1
        Pour i <- 2 JusquÀ i = racine_carrée(m) - 1 Faire
            Si i diviseur de m Alors
                somme_m <- somme_m + i + (m Div i)
            FinSi
```

```
        FinPour
        Si m est un carré parfait Alors
            somme_m <- somme_m + i
        FinSi

        { somme_m est la candidat pour n }

        n <- somme_m
        Si n < m Alors  { on s'intéresse au cas  n <= m }
            -- Déterminer la somme des diviseurs de n (somme_n)
            somme_n <- 1
            Pour i <- 2 JusquÀ i = racine_carrée(n) - 1 Faire
                Si i diviseur de n Alors
                    somme_n <- somme_n + i + (n Div i)
                FinSi
            FinPour
            Si n est un carré parfait Alors
                somme_n <- somme_n + i
            FinSi


            Si somme_n = m Alors        { somme_m et m amis }
                Afficher le couple (somme_m, m)
            FinSi
        FinPour

    FinPour
Fin.
```

```
!*****************************************************************
!*  Auteur   : Denis Barreteau <Denis.Barreteau@ensiacet.fr>
!*  Version  : 1.1
!*  Revision : Xuan Meyer < XuanMi.Meyer@ensiacet.fr>
!*
!*  Objectif : Afficher les couples de nombres amis (N, M)
!*             avec 1 < N < M <= MAX.
!*             version plus reflechie
!*****************************************************************

PROGRAM main
    INTEGER:: nmax
    INTEGER:: m             ! parcourir les entiers de 2 a MAX
    INTEGER:: somme_m       ! somme des diviseurs de m
    INTEGER:: somme_n       ! somme des diviseurs de somme_m correspondant a n
    INTEGER:: racine        ! la racine carree d'un nombre
    INTEGER:: i             ! variable de boucle

    PRINT*, "valeur de Max"
    READ*, nmax
    DO m = 2,nmax
        ! Determiner la somme des divieurs de m
        somme_m = 1
```

```
24          racine =sqrt(REAL(m))
25          DO i = 2,racine
26              IF (m/i*i == m)somme_m = somme_m + i + (m / i)
27              ! i diviseur de m
28          ENDDO
29          IF (m == racine * racine)somme_m = somme_m - racine
30          ! m est un carre parfait
31          IF (somme_m <= m)THEN        ! on s'interesse au cas  n <= m
32              ! Determiner la somme des diviseurs de somme_m (somme_n)
33              somme_n = 1
34              racine = sqrt(REAL(somme_m))
35              DO i = 2,racine
36                  IF (somme_m/i*i == somme_m)somme_n = somme_n + i + (somme_m / i)
37                  !  i diviseur de somme_m
38              END DO
39              IF (somme_m == racine * racine)somme_n = somme_n - racine
40              ! carre parfait
41              ! determiner si n et m sont amis
42              IF (somme_n == m)PRINT*, somme_m, m ! n et m sont amis
43          ENDIF
44      ENDDO
45
46  END PROGRAM main
47
48  !  Remarque : pour la structuration du programme, il serait preferable
49  !  d'utiliser un sous-programme (une fonction) qui calcule la somme des
50  !  diviseurs d'un entier n.
```

Exercice 5 : Puissance

Calculer et afficher la puissance entière d'un réel.

Solution : On peut, dans un premier temps, se demander si la puissance entière n d'un réel x a toujours un sens. En fait, ceci n'a pas de sens si x est nul et si n est strictement négatif. D'autre part, le cas $x = 0$, $n = 0$ est indéterminé. On choisit de contrôler la saisie de manière à garantir que nous ne sommes pas dans l'un de ces cas. Nous souhaitons donner à l'utilisateur un message le plus clair possible lorsque la saisie est invalide.

Nous envisageons les jeux de tests suivants :

```
1     x    n  -->   x^n
2
3     2    3  -->     8      -- cas nominal
4     3    2  -->     9      -- cas nominal
5     1    1  -->     1      -- cas nominal
6     2   -3  --> 0.125      -- cas nominal (puissance négative)
7     0    2  -->     0      -- cas nominal (x est nul)
8     0   -2  --> ERREUR     -- division par zéro
9     0    0  --> Indéterminé
10  1.5    2  --> 2.25       -- x peut être réel
```

Voyons maintenant le principe de la solution. Par exemple, pour calculer 2^3, on peut faire $2 * 2 * 2$. Plus généralement, on multiplie n fois x par lui-même (donc une boucle **Pour**).

Si on essaie ce principe, on constate qu'il ne fonctionne pas dans le cas d'une puissance négative. Prenons le cas de 2^{-3}. On peut le réécrire $(1/2)^3$. On est donc ramené au cas précédent. Le facteur multiplicatif est dans ce cas $1/x$ et le nombre de fois est $-n$.

Nous introduisons donc deux variables intermédiaires qui sont :

```
facteur: Réel    -- facteur multiplicatif pour obtenir les puissances
                 -- successives de x
puissance: Réel -- abs(n).  On a : facteur^puissance = x^n
```

On peut maintenant formaliser notre raisonnement sous la forme du raffinage suivant.

```
1  R0 : Afficher la puissance entière d'un réel
2
3  R1 : Raffinage De « R0 »
4       | Saisir avec contrôle les valeurs de x et n      x: out Réel; n: out Entier
5       | { (x <> 0) Ou (n > 0) }
6       | Calculer x à la puisssance n                    x, n: in ; xn: out Réel
7       | Afficher le résultat                            xn: in Réel
8
9  R2 : Raffinage De « Saisir avec contrôle les valeurs de x et n »
10      | Répéter
11      |   | Saisir la valeur de x et n          x: out Réel; n: out Entier
12      |   | Contrôler x et n                    x, n: in ; valide: out Booléen
13      | JusquÀ valide                           valide: in
14
15 R1 : Raffinage De « Calculer x à la puisssance n »
16      | Si x = 0 Alors
17      |   | xn := 0
18      | Sinon
```

```
19      |   | Déterminer le facteur multiplicatif et la puissance
20      |   |                     x, n: in ;
21      |   |                     facteur out Réel ;
22      |   |                     puissance: out Entier
23      |   | Calculer xn par itération (accumulation)
24      |   |                     n, facteur, puissance: in ; xn : out
25      | FinSi
26
27  R3 : Raffinage De « Calculer xn par itération (accumulation) »
28      | xn <- 1;
29      | Pour i <- 1 JusquÀ i = puissance Faire
30      |   | { Invariant : xn = facteur^i }
31      |   | xn <- xn * facteur
32      | FinPour
```

On peut alors en déduire l'algorithme suivant :

```
1   Algorithme puissance
2
3       -- Afficher la puissance entière d'un réel
4
5   Variables
6       x: Réel             -- valeur réelle lue au clavier
7       n: Entier           -- valeur entière lue au clavier
8       valide: Booléen     -- x^n peut-elle être calculée
9       xn: Réel            -- x à la puissance n
10      facteur: Réel       -- facteur multiplicatif pour obtenir
11                          -- les puissances successives
12      puissance: Entier;  - abs(n).  On a facteur^puissance = x^n
13      i: Entier           -- variable de boucle
14
15  Début
16      -- Saisir avec contrôle les valeurs de x et n
17      Répéter
18          -- saisir la valeur de x et n
19          Écrire("x = ")
20          Lire(x)
21          Écrire("n = ")
22          Lire(n)
23
24          -- contrôler x et n
25          valide <- VRAI
26          Si x = 0 Alors
27              Si n = 0 Alors
28                  ÉcrireLn("x et n sont nuls. x^n est indéterminée.")
29                  valide <- FAUX
30              SinonSi n < 0 Alors
31                  ÉcrireLn("x nul et n négatif. x^n n'a pas de sens.")
32                  valide <- FAUX
33              FinSi
34          FinSi
35
```

```
36          Si Non valide Alors
37              ÉcrireLn("␣␣Recommencez␣!")
38          FinSi
39      JusquÀ valide
40
41      -- Calculer x à la puisssance n
42      Si x = 0 Alors      -- cas trivial
43          xn <- 0
44      Sinon
45          -- Déterminer le facteur multiplicatif et la puissance
46          Si n >= 0 Alors
47              facteur <- x
48              puissance <- n
49          Sinon
50              facteur <- 1/x
51              puissance <- -n
52          FinSi
53
54          -- Calculer xn par itération (accumulation)
55          xn <- 1;
56          Pour i <- 1 JusquÀ i = puissance Faire
57              { Invariant : xn = facteur^i}
58              xn <- xn * facteur
59          FinPour
60      FinSi
61
62      -- Afficher le résultat
63      ÉcrireLn(x, "^", n, "␣=␣", xn)
64  Fin.
```

```
1   !*********************************************************************
2   !*  Auteur   : Denis Barreteau <Denis.Barreteau@ensiacet.fr>
3   !*  Version  : 1.1
4   !*  Revision : Xuan Meyer < XuanMi.Meyer@ensiacet.fr>
5   !*  Objectif : Afficher la puissance entiere d'un reel
6   !*********************************************************************
7
8   PROGRAM main
9
10      REAL:: x            ! valeur reelle lue au clavier
11      INTEGER:: n         ! valeur entiere lue au clavier
12      LOGICAL:: valide    ! x^n peut-elle être calculee
13      REAL:: xn           ! x a la puissance n
14      REAL:: facteur      ! facteur multiplicatif pour
15                          ! obtenir les puissances successives
16
17      INTEGER:: puissance    ! abs(n).  On a : facteur^puissance == x^n
18      INTEGER:: i            ! variable de boucle
19
20      ! Saisir avec controle les valeurs de x et n
21      do
```

```
22        ! saisir la valeur de x et n
23        PRINT*,"x = "
24        READ*,x
25         PRINT*,x
26        PRINT*,"n = "
27        READ*,n
28
29        ! controler x et n
30        valide = .true.
31        IF (x == 0) THEN
32            IF (n == 0) THEN
33              PRINT*,"x et n sont nuls. x^n est indeterminee."
34              valide = .false.
35            ELSEIF (n < 0) THEN
36              PRINT*,"x nul et n negatif. x^n n'a pas de sens."
37              valide = .false.
38            ENDIF
39         ENDIF
40         IF (.NOT.valide) PRINT*,"  Recommencez "
41         IF(valide) EXIT
42      ENDDO
43      ! Calculer x à la puisssance n
44      IF (x == 0) THEN      ! cas trivial
45        xn = 0.0
46      ELSE
47        ! Determiner le facteur multiplicatif et la puissance
48        IF (n >= 0) THEN
49            facteur = x
50            puissance = n
51        ELSE
52            facteur = 1.0/x
53            puissance = -n
54        ENDIF
55
56        ! Calculer xn par iteration (accumulation)
57        xn = 1.0
58        DO i = 1,puissance
59            xn = xn * facteur
60        ENDDO
61      ENDIF
62
63      ! Afficher le resultat
64       PRINT*, x,"^",n," =",xn
65
66      END PROGRAM main
```

Exercice 6 : Amélioration du calcul de la puissance entière

Améliorer l'algorithme de calcul de la puissance (exercice 5 du Exercices corrigés en F, Semaine 1) en remarquant que

$$x^n = \begin{cases} (x^2)^p & \text{si } n = 2p \\ (x^2)^p \times x & \text{si } n = 2p+1 \end{cases}$$

Ainsi, pour calculer 3^5, on peut faire $3 * 9 * 9$ avec bien sûr $9 = 3^2$.

Solution : Nous nous appuyons sur les mêmes variables que pour le calcul « naturel » de la puissance (exercice 5). Nous allons continuer à calculer x^n par accumulation. Nous avons l'invariant suivant :

$$x^n = \text{xn} * \text{facteur}^{\text{puissance}}$$

Lors de l'initialisation, cet invariant est vrai (`xn` vaut 1 et facteur et puissance sont tels qu'ils valent x^n).

D'après la formule donnée dans l'énoncé, à chaque itération de la boucle, deux cas sont à envisager :

– soit `puissance` est paire. On peut alors l'écrire $2 * p$. On a alors :

$$\begin{aligned} x^n &= \text{xn} * \text{facteur}^{\text{puissance}} \\ &= \text{xn} * \text{facteur}^{(2\ *\ \text{p})} \\ &= \text{xn} * (\text{facteur}^2)^{\text{p}} \end{aligned}$$

On peut donc faire :

```
facteur <- facteur * facteur
puissance <- puissance Div 2    -- car p = puissance Div 2
```

On constate que l'invariant est préservé d'après les égalités ci-dessus et la puissance a strictement diminué (car divisée par 2).

– soit `puissance` est impaire. On peut alors l'écrire $2 * p + 1$. On a alors :

$$\begin{aligned} x^n &= \text{xn} * \text{facteur}^{\text{puissance}} \\ &= \text{xn} * \text{facteur}^{(2\ *\ \text{p}\ +\ 1)} \\ &= \text{xn} * (\text{facteur} * \text{facteur}^{(2*\text{p})}) \\ &= (\text{xn} * \text{facteur}) * \text{facteur}^{(2*\text{p})} \end{aligned}$$

On peut donc faire :

```
xn <- xn * facteur
puissance <- puissance - 1
```

On constate que l'invariant est préservé d'après les égalités ci-dessus et la puissance a strictement diminué (car diminuée de 1).

On peut alors en déduire le raffinage suivant :

```
1 R3 : Raffinage De « Calculer xn par itération (accumulation) »
2       | xn <- 1;
3       | TantQue puissance > 0 Faire
```

```
4       |   | { Variant : puissance }
5       |   | { Invariant : x^n = xn * facteur^puissance }
6       |   | Si puissance Div 2 = 0 Alors        { puissance = 2 * p }
7       |   |  | puissance <- puissance Div 2
8       |   |  | facteur <- facteur * facteur
9       |   | Sinon                               { puissance = 2 * p + 1 }
10      |   |  | puissance <- puissance - 1
11      |   |  | xn <- xn * facteur
12      |   | FinSi
13      | FinTQ
```

```
1  !**********************************************************************
2  !*  Auteur   : Denis Barreteau <Denis.Barreteau@ensiacet.fr>
3  !*  Version  : 1.1
4  !*  Revision : Xuan Meyer < XuanMi.Meyer@ensiacet.fr>
5  !*
6  !*  Objectif : Afficher la puissance entiere d'un reel
7  !*             Autre approche
8  !**********************************************************************
9
10 PROGRAM main
11
12     REAL::x              ! valeur reelle lue au clavier
13     INTEGER:: n          ! valeur entiere lue au clavier
14     LOGICAL:: valide     ! x^n peut-elle etre calculee
15     REAL::xn             ! x a la puissance n
16     REAL::facteur        ! facteur multiplicatIF pour
17                          ! obtenir les puissances successives
18
19     INTEGER:: puissance  ! abs(n).  On a : facteur^puissance == x^n
20
21     ! Saisir avec contrôle les valeurs de x et n
22     do
23         ! saisir la valeur de x et n
24         PRINT*,"x = "
25         READ*, x
26         PRINT*,"n = "
27         READ*,n
28
29         ! contrôler x et n
30         valide = .true.
31         IF (x == 0) THEN
32             IF (n == 0) THEN
33                 PRINT*,"x et n sont nuls. x^n est indeterminee."
34                 valide = .false.
35             ELSE IF (n < 0) THEN
36                 PRINT*,"x nul et n negatIF. x^n n'a pas de sens."
37                 valide = .false.
38             ENDIF
39         ENDIF
40         IF (.NOT.valide) PRINT*,"  Recommencez !\n"
41         IF(valide)EXIT
```

```
42      ENDDO
43      ! Calculer x à la puisssance n
44      IF (x == 0.0) THEN        ! cas trivial
45          xn = 0.0
46      ELSE
47          ! Determiner le facteur multiplicatif et la puissance
48          IF (n >= 0) THEN
49              facteur = x
50              puissance = n
51          ELSE
52              facteur = 1.0/x
53              puissance = -n
54          ENDIF
55
56          ! Calculer xn par iteration (accumulation)
57          xn = 1.0
58          DO
59              IF(puissance == 0) EXIT
60              !{ Invariant : xn * facteur ^ puissance == x ^ n }
61              !{ Variant : puissance }
62              IF (puissance / 2*2 == puissance) THEN
63                  facteur = facteur * facteur
64                  puissance = puissance / 2
65              ELSE
66                  xn = xn * facteur
67                  puissance = puissance - 1
68              ENDIF
69          ENDDO
70      ENDIF
71
72      ! Afficher le resultat
73      PRINT*, x,"^",n,"␣=",xn
74
75  END PROGRAM main
```

Exercice 7 : Nombres de Amstrong

Les *nombres de Amstrong* appelés parfois *nombres cubes* sont des nombres entiers qui ont la particularité d'être égaux à la somme des cubes de leurs chiffres. Par exemple, 153 est un nombre de Amstrong car on a :

$$153 = 1^3 + 5^3 + 3^3.$$

Afficher tous les nombres de Amstrong sachant qu'ils sont tous compris entre 100 et 499.

Indication : Les nombres de Amstrong sont : 153, 370, 371 et 407.

Solution :

```
R0 : Afficher les nombres de Amstrong compris entre 100 et 499.
```

On peut envisager (au moins) deux solutions pour résoudre ce problème.

Solution 1. La première solution consiste à essayer les combinaisons de trois chiffres qui vérifient la propriété. On prend alors trois compteurs : un pour les unités, un pour les dizaines et un pour les centaines. Les deux premiers varient de 0 à 9. Le dernier de 1 à 4. Ayant les trois chiffres, on peut calculer la somme de leurs cubes puis le nombre qu'ils forment et on regarde si les deux sont égaux.

Ceci se formalise dans le raffinage suivant :

```
R1 : Raffinage De « Afficher les nombres de Amstrong »
    Pour centaine <- 1 JusquÀ centaine = 4 Faire
        Pour dizaine <- 1 JusquÀ dizaine = 9 Faire
            Pour unité <- 1 JusquÀ unité = 9 Faire
                Déterminer le cube
                Déterminer le nombre
                Si nombre = cube Alors
                    Afficher nombre
                FinSi
            FinPour
        FinPour
    FinPour
```

On en déduit alors le programme Fortran suivant.

```
!*****************************************************************
!*  Auteur   : Denis Barreteau <Denis.Barreteau@ensiacet.fr>
!*  Version  : 1.1
!*  Revision : Xuan Meyer < XuanMi.Meyer@ensiacet.fr>
!*
!*  Objectif : Afficher les nombres de Amstrong
!*             Version la plus simple
!*
!*****************************************************************

PROGRAM main
    ! Principe : on parcourt tous les trois chiffres du nombre
    ! * compris entre 100 et 499.

    INTEGER:: nb               ! le nombre considere
```

```
16      INTEGER:: centaine, dizaine, unite        ! les trois chiffres de nb
17      INTEGER:: cube                ! le somme des cubes des trois chiffres
18
19      DO centaine = 1,4
20          DO dizaine = 0,9
21              DO unite = 0,9
22                  ! determiner la somme des cubes
23                  cube = centaine * centaine * centaine
24                  cube = cube + dizaine * dizaine * dizaine
25                  cube = cube  + unite * unite * unite
26
27                  ! determiner le nombre
28                  nb = centaine * 100 + dizaine * 10 + unite
29
30                  if (nb == cube)PRINT*,nb ! c'est un nombre de Amstrong
31              ENDDO
32          ENDDO
33      ENDDO
34
35  END PROGRAM main
```

On remarque que les opérations peuvent être réoganisées pour augmenter les performances en temps de calcul. En particulier, il est inutile de faire des calculs à l'intérieure d'une boucle s'ils ne dépendent pas de la boucle. Ainsi, le calcul du cube des centaines peut se faire dans la boucle la plus externe au lieu de le faire dans la plus interne.

On arrive alors à une nouvelle version du programme.

```
1   !**********************************************************************
2   !*  Auteur   : Denis Barreteau <Denis.Barreteau@ensiacet.fr>
3   !*  Version  : 1.1
4   !*  Revision : Xuan Meyer < XuanMi.Meyer@ensiacet.fr>
5   !*
6   !*  Objectif : Afficher les nombres de Amstrong
7   !*
8   !**********************************************************************
9
10  PROGRAM main
11      ! Principe : on parcourt tous les trois chiffres du nombre
12      ! * compris entre 100 et 499.
13      ! * On sort de la boucle interne certains calculs (centaine3,
14      ! * dizaine3)
15
16      INTEGER:: nb                ! le nombre considere
17      INTEGER:: centaine, dizaine, unite        ! les trois chiffres de nb
18      INTEGER:: centaine3, dizaine3, unite3       ! le cube des trois chiffres
19      INTEGER:: cube                ! le somme des cubes des trois chiffres
20
21      DO centaine = 1,4
22          centaine3 = centaine * centaine * centaine
23          DO dizaine = 0,9
24              dizaine3 = dizaine * dizaine * dizaine
25              DO unite = 0,9
```

```
26                 unite3 = unite * unite * unite
27
28                 ! determiner la somme des cubes
29                 cube = centaine3 + dizaine3 + unite3
30
31                 ! determiner le nombre
32                 nb = centaine * 100 + dizaine * 10 + unite
33
34                 if (nb == cube)PRINT*, nb ! c'est un nombre de Amstrong
35             ENDDO
36         ENDDO
37     ENDDO
38
39 END PROGRAM main
```

Solution 2. La deuxième solution consiste à parcourir tous les entiers compris entre 100 et 499 et à regarder s'ils sont égaux à la somme des cubes de leurs chiffres. La difficulté est alors d'extraite les chiffres. L'idée est d'utiliser la division entière par 10 et son reste.

On obtient le raffinage suivant.

```
1  R1 : Raffinage De « Afficher les nombres de Amstrong »
2      Pour nombre <- 100 JusquÀ centaine = 499 Faire
3          Déterminer les chiffres de nb
4          Déterminer le cube des chiffres
5          Si nombre = cube Alors
6              Afficher nombre
7          FinSi
8      FinPour
9
10 R2 : Raffinage De « Déterminer les chiffres de nb »
11     unité <- nombre Mod 10
12     dizaine <- (nombre Div 10) Mod 10
13     centaine <- nombre Div 100
```

On en déduit alors le programme Fortran suivant.

```
1  !*********************************************************************
2  !*  Auteur   : Denis Barreteau <Denis.Barreteau@ensiacet.fr>
3  !*  Version  : 1.1
4  !*  Revision : Xuan Meyer < XuanMi.Meyer@ensiacet.fr>
5  !*
6  !*  Objectif : Afficher les nombres de Amstrong
7  !*             autre approche
8  !*********************************************************************
9
10 PROGRAM main
11     ! Principe : on parcourt tous les entiers de 100 à 499 et
12     ! on verifie s'il s'agit d'un nombre de Amstrong.
13
14
15     INTEGER:: nb               ! le nombre considere
```

```
16      INTEGER:: centaine, dizaine, unite        ! les trois chiffres de nb
17      INTEGER:: cube                ! le somme des cubes des trois chiffres
18
19      DO nb = 100,499
20          ! determiner les chiffres de nb
21          unite = nb-nb/10*10
22          dizaine = (nb / 10)
23          dizaine = dizaine-dizaine/10*10
24          centaine = nb / 100
25
26          ! determiner le cube des chiffres
27          cube = centaine * centaine * centaine
28          cube = cube + dizaine * dizaine * dizaine
29          cube = cube + unite * unite * unite
30
31          if (nb == cube)PRINT*, nb ! c'est un nombre de Amstrong
32      END DO
33
34  END PROGRAM main
```