Les bases : exercices corrigés en VBA
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   Attribute VB_Name = "exo1_second"
2   '*********************************************************************
3   '*  Auteur   : Claude Monteil <monteil@ensat.fr>
4   '*  Version  : 1.0
5   '*  Objectif : Resolution de l'equation du second degre.
6   '*             Approche classique
7   '*********************************************************************
8
9   Option Explicit 'pour forcer la declaration explicite des variables
10
11  Sub second_degre()
12  ' Demande à l'utilisateur les coefficients a, b et c de l'equation du
13  ' second degre  a*x^2 + b*x + c = 0 , calcule et affiche les solutions
14  ' reelles de cette equation.
15
16      EffacerEcran "Equation du 2nd degré"
17    '1.Declaration des variables
```

```
18      Dim a, b, c As Double ' coefficients de l'equation
19      Dim delta As Double   ' discriminant
20
21     '2.Poser le problème
22      Afficher "Resolution de l'equation du second degre :"
23      Afficher "a.x^2 + b.x + c = 0"
24
25     '3.Saisir la valeur des coefficients
26      Afficher "Donnez la valeur des coefficients :"
27      Afficher "coefficient a : "
28      Saisir a
29      Afficher "coefficient b : "
30      Saisir b
31      Afficher "coefficient c : "
32      Saisir c
33
34     '4.Resoudre l'equation et afficher les resultats
35      If (a = 0) Then ' resoudre l'equation du premier degre b*x + c = 0
36          If (b = 0) Then
37              If (c = 0) Then
38                  Afficher "Tout les reels sont solutions"
39              Else
40                  Afficher "Pas de solution"
41              End If
42          Else
43              Afficher "Une seule solution : " & -c / b
44          End If
45      Else ' cas general
46          delta = b * b - 4 * a * c ' le discriminant
47          If (delta > 0) Then ' deux solutions reelles
48              Afficher "Deux solutions : "
49              Afficher "solution 1 = " & (-b + Sqr(delta)) / 2 / a
50              Afficher "solution 2 = " & (-b - Sqr(delta)) / (2 * a)
51              'NOTA : 1 / 2 / a et 1 / (2 * a) sont 2 écritures équivalentes
52          ElseIf (delta = 0) Then ' une solution double
53              Afficher "Une solution double : " & (-b / 2 / a)
54          Else ' pas de solutions reelles
55              Afficher "Pas de solutions reelles"
56          End If
57      End If
58
59  End Sub
```

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 Attribute VB_Name = "exo2_second_ameliore"
2 '*****************************************************************
3 '*  Auteur   : Claude Monteil <monteil@ensat.fr>
4 '*  Version  : 1.0
5 '*  Objectif : Resolution de l'equation du second degre.
6 '*             Approche classique
7 '*****************************************************************
8
```

```
9  Option Explicit 'pour forcer la declaration explicite des variables
10
11 Sub second_degre_ameliore()
12 ' Demande à l'utilisateur les coefficients a, b et c de l'equation du
13 ' second degre  a*x^2 + b*x + c = 0 , calcule et affiche les solutions
14 ' reelles de cette equation.
15
16   EffacerEcran "Equation du 2nd degré (amélioré)"
17  '1.Declaration des variables
18   Dim a, b, c As Double ' coefficients de l'equation
19   Dim delta As Double   ' discriminant
20   Dim x1 As Double, x2 As Double ' solutions
21
22  '2.Poser le problème
23   Afficher "Resolution de l'equation du second degre :"
24   Afficher "a.x^2 + b.x + c = 0"
25
26  '3.Saisir la valeur des coefficients
27   Afficher "Donnez la valeur des coefficients :"
28   Afficher "coefficient a : "
29   Saisir a
30   Afficher "coefficient b : "
31   Saisir b
32   Afficher "coefficient c : "
33   Saisir c
34
35  '4.Resoudre l'equation et afficher les resultats
36   If (a = 0) Then ' resoudre l'equation du premier degre b*x + c = 0
37       If (b = 0) Then
38           If (c = 0) Then
39               Afficher "Tout les reels sont solutions"
40           Else
41               Afficher "Pas de solution"
42           End If
43       Else
44           Afficher "Une seule solution : " & -c / b
45       End If
46   Else ' cas general
47       delta = b * b - 4 * a * c ' le discriminant
48       If (delta > 0) Then ' deux solutions reelles
49
50           If (b < 0) Then ' suivant le signe de b le calcul de -b + Sqr(delta)
51                           ' est plus ou moins precis
52                x1 = (-b + Sqr(delta)) / 2 / a
53           Else
54                x1 = (-b - Sqr(delta)) / 2 / a
55           End If
56           x2 = c / a / x1
57
58           Afficher "Deux solutions :"
59           Afficher "solution 1 = " & x1
60           Afficher "solution 2 = " & x2
```

```
61          ElseIf (delta = 0) Then ' une solution double
62              Afficher "Une solution double : " & (-b / 2 / a)
63          Else ' pas de solutions reelles
64              Afficher "Pas de solutions reelles"
65          End If
66      End If
67
68  End Sub
```

Exercice 3 : Le nombre d'occurrences du maximun

Compléter le programme de l'exercice 2 (Exercices résolus en VBA, 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  Attribute VB_Name = "exo3_statistiques_clavier"
2  '*************************************************************************
3  '*  Auteur  :  Claude Monteil <monteil@ensat.fr>
4  '*  Version :  1.0
5  '*  Objectif :
6  '*     Afficher la moyenne, la plus grande et la plus petite valeur et le
7  '*     nombre d'occurences de la plus grande valeur d'une serie de valeurs
8  '*     reelles lues au clavier.
9  '*************************************************************************
10
11 Option Explicit
12
13 Sub statistiques_clavier_occ_max()
14
15     Dim x As Double       ' un reel lu au clavier
16     Dim nb As Integer     ' le nombre de valeurs lues de la serie
17     Dim somme As Double   ' la somme des valeurs lues de la serie
18     Dim min As Double     ' la plus petite des valeurs lues de la serie
19     Dim max As Double     ' la plus grande des valeurs lues de la serie
20     Dim nb_max As Double  ' le nombre d'occurrences du max
21
22     EffacerEcran "Nombre d'occurrences du maximum"
23    '1.afficher la consigne
24     Afficher "Donnez, en colonne, une serie d'entiers qui se termine par 0."
25
26    '2.lire le premier reel
27     Saisir x
28
29     If (x = 0) Then        ' Pas de valeur dans la serie
30         Afficher "La serie est vide.  "
31         Afficher "Les statistiques demandees n'ont pas de sens !"
32     Else
33        '3.initialiser les variables statistiques avec x
34         max = x
35         min = x
36         somme = x
37         nb = 1
38         nb_max = 1
```

```
39
40          '4.lire une nouvelle valeur x
41           Saisir x
42
43           Do While x <> 0
44              '5.mettre à jour les variables statistiques
45               nb = nb + 1
46               somme = somme + x
47               If (x > max) Then
48                   max = x
49                   nb_max = 1
50               ElseIf (x = max) Then
51                   nb_max = nb_max + 1
52               ElseIf (x < min) Then
53                   min = x
54               End If
55              '6.lire une nouvelle valeur x
56               Saisir x
57           Loop
58          '7.afficher les statistiques
59           Afficher "Moyenne = " & somme / nb
60           Afficher "Plus petite valeur = " & min
61           Afficher "Plus grande valeur = " & max
62           Afficher "Nb d'occurrences du max = " & nb_max
63      End If
64
65  End Sub
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
```

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

Variables
    n, m: Entier        -- pour représenter les couples possibles
    somme_n: Entier     -- somme des diviseurs de n
    somme_m: Entier     -- somme des diviseurs de m

Début
    Pour m <- 2 JusquÀ m = MAX Faire
        -- calculer la somme des divieurs de m
        -- Remarque : on peut déplacer cette étape à l'extérieur de la
        -- boucle car elle ne dépend pas de n (optimisation).
        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


        Pour n <- 2 JusquÀ n = m - 1 Faire
            -- calculer la somme des divieurs de 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

            -- déterminer si n et m sont amis
            Si (somme_n = m) Et (somme_m = n) Alors { n et m sont amis }
                Afficher le couple (n, m)
            FinSi
        FinPour
    FinPour

Fin.
```

```
Attribute VB_Name = "exo4_nb_amis"
'*********************************************************************
'*  Auteur   : Claude Monteil <monteil@ensat.fr>
'*  Version  : 1.1
'*  Objectif : Afficher les couples de nombres amis (N, M)
'*             avec 1 < N < M <= MAX.
'*             version simple
'*********************************************************************
```

```
9  Option Explicit
10
11 Sub nb_amis()
12     Dim nmax As Integer            ' borne superieure
13     Dim n As Integer, m As Integer ' pour representer les couples possibles
14     Dim somme_n As Integer         ' somme des diviseurs de n
15     Dim somme_m As Integer         ' somme des diviseurs de m
16     Dim racine As Integer          ' la racine carree d'un nombre
17     Dim i As Integer               ' variable de boucle
18
19     EffacerEcran "Recherche de nombres amis"
20     Afficher "valeur de Max"
21     Saisir nmax
22     For m = 2 To nmax 'tester si m peut etre ami d'un autre nombre
23         ' calculer la somme des diviseurs de m
24         ' Remarque : on a pu deplacer cette etape a l'exterieur de la boucle
25         ' ci-dessous concernant n car elle ne depend pas de n (optimisation).
26         somme_m = 1
27         racine = Round(Sqr(m))
28         For i = 2 To racine
29             If (m Mod i = 0) Then ' i diviseur de m
30                 somme_m = somme_m + i + (m \ i)  ' operateur \ : division entiere
31             End If
32         Next i
33         If (m = racine * racine) Then ' m est un carre parfait : racine compté 2 fois
34             somme_m = somme_m - racine
35         End If
36
37         For n = 2 To m 'tester si n peut être amis avec m
38             ' calculer la somme des divieurs de n
39             somme_n = 1
40             racine = Round(Sqr(n))
41             For i = 2 To racine
42                 If (n Mod i = 0) Then ' i diviseur de n
43                     somme_n = somme_n + i + (n \ i)  ' operateur \ : division entiere
44                 End If
45             Next i
46             If (n = racine * racine) Then ' n est un carre parfait : racine compté 2 f
47                 somme_n = somme_n - racine
48             End If
49
50             ' determiner si n et m sont amis
51             If (somme_n = m) And (somme_m = n) And (n <> m) Then Afficher n, m
52               ' n et m sont amis s'ils sont differents
53          Next n
54     Next m
55 End Sub
```

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
```

```
19          FinPour
20          Si m est un carré parfait Alors
21              somme_m <- somme_m + i
22          FinSi
23
24          { somme_m est la candidat pour n }
25
26          n <- somme_m
27          Si n < m Alors  { on s'intéresse au cas  n <= m }
28              -- Déterminer la somme des diviseurs de n (somme_n)
29              somme_n <- 1
30              Pour i <- 2 JusquÀ i = racine_carrée(n) - 1 Faire
31                  Si i diviseur de n Alors
32                      somme_n <- somme_n + i + (n Div i)
33                  FinSi
34              FinPour
35              Si n est un carré parfait Alors
36                  somme_n <- somme_n + i
37              FinSi
38
39
40              Si somme_n = m Alors        { somme_m et m amis }
41                  Afficher le couple (somme_m, m)
42              FinSi
43          FinPour
44
45      FinPour
46  Fin.
```

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   Attribute VB_Name = "exo5_algo_puissance"
2   '*****************************************************************
3   '*  Auteur   : Claude Monteil <monteil@ensat.fr>
4   '*  Version  : 1.0
5   '*  Objectif : Afficher la puissance entiere d'un reel
6   '*****************************************************************
7
8   Option Explicit
9
10  Sub x_puissance_n()
11      Dim x As Double          ' valeur reelle lue au clavier
12      Dim n As Integer         ' valeur entiere lue au clavier
13      Dim valide As Boolean    ' x^n peut-elle être calculee
14      Dim xn As Double         ' x a la puissance n
15      Dim facteur As Double    ' facteur multiplicatif pour
16                               ' obtenir les puissances successives
17      Dim puissance As Integer ' abs(n).  On a : facteur^puissance = x^n
18      Dim i As Integer         ' variable de boucle
19
20      EffacerEcran "Puissance␣entière␣d'un␣nombre"
21     '1.Saisir avec controle les valeurs de x et n
```

```
22      Do
23         'la.saisir la valeur de x et n
24          Afficher "x = "
25          Saisir x
26          Afficher "n = "
27          Saisir n
28         'lb.controler x et n
29          valide = True
30          If (x = 0) Then
31              If (n = 0) Then
32                  Afficher "x et n sont nuls. x^n est indeterminee."
33                  valide = False
34              ElseIf (n < 0) Then
35                  Afficher "x nul et n negatif. x^n n'a pas de sens."
36                  valide = False
37              End If
38           End If
39           If (Not valide) Then Afficher " Recommencez "
40      Loop Until valide
41
42     '2.Calculer x à la puisssance n
43      If (x = 0) Then ' cas trivial
44        xn = 0
45      Else ' Determiner le facteur multiplicatif et la puissance
46          If (n >= 0) Then
47              facteur = x
48              puissance = n 'puissance >= 0
49          Else
50              facteur = 1 / x
51              puissance = -n 'puissance > 0
52          End If
53
54          ' Calculer xn par iteration (accumulation)
55          xn = 1
56          For i = 1 To puissance
57              ' invariant : xn = facteur^i
58              xn = xn * facteur
59          Next i
60      End If
61
62     '3.Afficher le resultat
63      Afficher x & "^" & n & " = " & xn
64  End Sub
```

Exercice 6 : Amélioration du calcul de la puissance entière
Améliorer l'algorithme de calcul de la puissance (exercice 5 du Exercices corrigés en VBA, 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  Attribute VB_Name = "exo6_algo_puissance_indienne"
2  '*********************************************************************
3  '*  Auteur   : Claude Monteil <monteil@ensat.fr>
4  '*  Version  : 1.0
5  '*  Objectif : Afficher la puissance entiere d'un reel
6  '*             Autre approche
7  '*********************************************************************
8
9  Option Explicit
10
11 Sub x_puissance_n_indienne()
12
13     Dim x As Double          ' valeur reelle lue au clavier
14     Dim n As Integer         ' valeur entiere lue au clavier
15     Dim valide As Boolean    ' x^n peut-elle être calculee
16     Dim xn As Double         ' x a la puissance n
17     Dim facteur As Double    ' facteur multiplicatif pour
18                              ' obtenir les puissances successives
19     Dim puissance As Integer ' abs(n).  On a : facteur^puissance = x^n
20
21     EffacerEcran "Puissance entière d'un nombre (exponentiation indienne)"
22    '1.Saisir avec controle les valeurs de x et n
23     Do
24        '1a.saisir la valeur de x et n
25         Afficher "x = "
26         Saisir x
27         Afficher "n = "
28         Saisir n
29        '1b.controler x et n
30         valide = True
31         If (x = 0) Then
32             If (n = 0) Then
33                 Afficher "x et n sont nuls. x^n est indeterminee."
34                 valide = False
35             ElseIf (n < 0) Then
36                 Afficher "x nul et n negatif. x^n n'a pas de sens."
37                 valide = False
38             End If
39          End If
40          If (Not valide) Then Afficher " Recommencez "
41     Loop Until valide
```

```
42
43      '2.Calculer x à la puisssance n
44      If (x = 0) Then ' cas trivial
45        xn = 0
46      Else ' Determiner le facteur multiplicatif et la puissance
47          If (n >= 0) Then
48              facteur = x
49              puissance = n 'puissance >= 0
50          Else
51              facteur = 1 / x
52              puissance = -n 'puissance > 0
53          End If
54
55          ' Calculer xn par iteration (accumulation)
56          xn = 1
57          Do While (puissance > 0)
58              'Invariant : xn * facteur ^ puissance = x ^ n
59              'Variant : puissance
60              If (puissance Mod 2 = 0) Then
61                  facteur = facteur * facteur
62                  puissance = puissance / 2
63              Else
64                  xn = xn * facteur
65                  puissance = puissance - 1
66              End If
67          Loop
68      End If
69
70      '3.Afficher le resultat
71      Afficher x & "^" & n & " = " & xn
72  End Sub
```

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 VBA suivant.

```
Attribute VB_Name = "exo7_nb_amstrong_simple"
'*****************************************************************
'*  Auteur   : Claude Monteil <monteil@ensat.fr>
'*  Version  : 1.0
'*  Objectif : Afficher les nombres de Amstrong
'*****************************************************************

Option Explicit

Sub nb_amstrong_simple()
' Principe : on parcourt tous les trois chiffres du nombre  compris entre 100 et 499.
'            On sort de la boucle interne certains calculs (centaine3, dizaine3)

    Dim nb As Integer   ' le nombre considere
    Dim centaine As Integer, dizaine As Integer, unite As Integer
```

```
16                          ' les trois chiffres de nb
17      Dim cube As Integer ' le somme des cubes des trois chiffres
18
19      EffacerEcran "Nombres de Amstrong (simple)"
20      For centaine = 1 To 4
21          For dizaine = 0 To 9
22              For unite = 0 To 9
23                  '1.determiner la somme des cubes
24                   cube = centaine * centaine * centaine
25                   cube = cube + dizaine * dizaine * dizaine
26                   cube = cube + unite * unite * unite
27                  '2.determiner le nombre
28                   nb = centaine * 100 + dizaine * 10 + unite
29                  '3.tester
30                   If (nb = cube) Then Afficher nb ' c'est un nombre de Amstrong
31              Next unite
32          Next dizaine
33      Next centaine
34  End Sub
```

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.

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
```