

Les bases : exercices résolus en VBA

Corrigé

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 : Conversion pouce/centimètre	1
Exercice 2 : Quelques statistiques	5
Exercice 3 : Division entière	10
Exercice 4 : Nombres premiers	12
Exercice 5 : Nombres parfaits	17

Exercice 1 : Conversion pouce/centimètre

Écrire un programme qui réalise la conversion pouce/centimètre d’une longueur saisie au clavier. Une longueur sera saisie comme un nombre réel suivi d’un caractère précisant l’unité. Les unités possibles sont le pouce (p), le centimètre (c) ou le mètre (m). Le programme affichera la longueur exprimée en pouce et en centimètre.

Voici des exemples d’exécution du programme :

Entrez une longueur : 1p
1 p = 2.54 cm

Entrez une longueur : 2m
78.7402 p = 200 cm

Entrez une longueur : 2km
0 p = 0 cm

Modifier le programme pour permettre la saisie de l’unité aussi bien en minuscules qu’en majuscules.

Solution :

R0 : Afficher une longueur saisie au clavier en pouces et en centimètres

Nous reprenons les jeux de test proposés dans le sujet. Nous ajoutons un jeu de test qui consiste à saisir une unité en centimètres. Par exemple, 5.08 cm qui doit donner 1 p = 5.08 p.

Lorsque l'on regarde les résultats d'exécution donnés dans le sujet (et qui servent donc de spécification pour le programme à écrire), on constate que, quelle que soit l'unité utilisée pour saisir la longueur, il faut afficher d'abord la longueur en pouces, puis son équivalent en centimètres. En conséquence, on en déduit qu'il faut calculer les deux longueurs (une dans chaque unité). Ainsi, après saisi de la longueur de l'utilisateur, nous allons la convertir dans les deux unités puis afficher la ligne d'équivalence.

```

1  R1 : Raffinage De « Afficher une longueur saisie au clavier... »
2  | Saisir la longueur          valeur: out Réel ; unité: out Caractère
3  | Calculer la longueur en pouces et en centimètres
4  |                             valeur, unité : in
5  |                             lg_p  : out Réel -- longueur en pouces
6  |                             lg_cm : out Réel -- longueur en centimètres
7  | Afficher le résultat          lg_p, lg_cm : in

```

La seule étape délicate est la deuxième. Calculer les longueurs en pouces et en centimètres dépend de l'unité saisie par l'utilisateur. Il s'agit donc d'utiliser une conditionnelle et plus précisément de faire un traitement par cas. Puisque l'unité est un caractère, donc un type scalaire, nous pouvons utiliser un **Selon**.

Remarque : Pour traiter le cas des majuscules, il suffit d'ajouter le cas majuscule à côté du cas minuscule correspondant.

On aurait également pu convertir la longueur saisie en minuscule avant de calculer les longueurs en pouces et centimètres.

Remarque : On aurait également pu utiliser des **Si** et **SinonSi**. Cependant, la structure du programme est plus claire en utilisant un **Selon**. Ceci est d'autant plus vrai si l'on traite également les majuscules.

Voici le raffinement correspondant.

```

1  R2 : Raffinage De « Calculer la longueur en pouces et en centimètres »
2  | Selon unité Dans
3  |   'p', 'P':          { la longueur a été saisie en pouces }
4  |   lg_p <- valeur
5  |   lg_cm <- lg_p * UN_POUCE
6  |
7  |   'c', 'C':          { la longueur a été saisie en centimètres }
8  |   lg_cm <- valeur
9  |   lg_p <- lg_cm / UN_POUCE
10 |
11 |   'm', 'M':          { la longueur a été saisie en mètres }
12 |   lg_cm <- valeur * 100
13 |   lg_p <- lg_cm / UN_POUCE
14 |
15 | Sinon                { Unité non reconnue }
16 |   lg_p <- 0
17 |   lg_cm <- 0
18 | FinSelon

```

Notons que nous avons utilisé une constante symbolique (UN_POUCE) plutôt qu'une constante littérale (2,54). L'intérêt est d'avoir un programme plus lisible et d'éviter la redondance (si on

se trompe sur la valeur d'un pouce en centimètres il suffit de faire un seul changement pour corriger le programme dans le cas de la constante symbolique contre trois sinon).

```

1  Algorithme pouce2cm
2
3      -- Afficher une longueur saisie au clavier en pouces et en centimètres.
4
5  Constantes
6      UN_POUCE = 2.54      -- valeur en centimètres d'un pouce
7  Variables
8      valeur: Réel        -- valeur de la longueur lue au clavier
9      unité: Caractère    -- unité de la longueur lue au clavier
10     lg_cm: Réel         -- longueur exprimée en centimètres
11     lg_p: Réel          -- longueur exprimée en pouces
12
13 Début
14     -- saisir la longueur (valeur + unité)
15     Écrire ("Entrer une longueur (valeur + unité) : ")
16     Lire (valeur)          -- saisir la valeur
17     Lire (unité)          -- saisir l'unité
18
19     -- calculer la longueur en pouces et en centimètres
20     Selon unité Dans
21         'p', 'P':          { la longueur a été saisie en pouces }
22             lg_p <- valeur
23             lg_cm <- lg_p * UN_POUCE
24
25         'c', 'C':          { la longueur a été saisie en centimètres }
26             lg_cm <- valeur
27             lg_p <- lg_cm / UN_POUCE
28
29         'm', 'M':          { la longueur a été saisie en mètres }
30             lg_cm <- valeur * 100
31             lg_p <- lg_cm / UN_POUCE
32
33     Sinon                  { Unité non reconnue }
34         lg_p <- 0
35         lg_cm <- 0
36     FinSelon
37
38     -- afficher le résultat
39     ÉcrireLn (lg_p, "_p=", lg_cm, "_cm")
40 Fin

1  Attribute VB_Name = "exo1_pouce2cm"
2  '*****
3  '* Auteur : Claude Monteil <monteil@ensat.fr>
4  '* Version : 1.0
5  '* Objectif : Conversion pouces/centimetres
6  '*
7  '* Remarque : Le sujet n'est pas entierement respecte car la partie
8  '* decimale est affichee même si elle est nulle '

```

```
9  '*****
10
11 Sub pouce_centimetre()
12     Const un_pouce As Double = 2.54
13     Dim valeur As Double ' valeur lue au clavier
14     Dim unite As String ' unite lue au clavier
15     ' valeur et unité forment la longueur
16     Dim p As Double ' longueur lue au clavier convertie en pouces
17     Dim cm As Double ' longueur lue au clavier convertie en centimètres
18
19     EffacerEcran "Conversion_en_pouces_et_cm"
20     '1.Saisir la longueur
21     Afficher "Entrez_la_valeur_de_la_longueur_:"
22     Saisir valeur
23     Afficher "Entre_l' unite_:_p_(pouce)_,_m_(metre)_,_c_(centimetre)"
24     Saisir unite
25
26     '2.Calculer la longueur en pouces et en centimetres
27     Select Case (unite)
28         Case "p", "P" ' longueur saisie en pouces
29             p = valeur
30             cm = valeur * un_pouce
31         Case "M", "m" ' longueur saisie en metres
32             p = 100 * valeur / un_pouce
33             cm = 100 * valeur
34
35         Case "c", "C" ' longueur saisie en centimetres
36             p = valeur / un_pouce
37             cm = valeur
38         Case Else
39             p = 0
40             cm = 0
41     End Select
42     '3.Afficher les résultats
43     Afficher p & "_pouces_et_" & cm & "_cm"
44     Afficher "Valeurs_arrondies_a_2_decimales_:"
45     Afficher Format(p, "0.00") & "_pouces_et_" & Format(cm, "0.00") & "_cm"
46 End Sub
```

Exercice 2 : Quelques statistiques

L'objectif de cet exercice est de calculer quelques statistiques sur une série de valeurs réelles. Cette série est lue au clavier. On considère qu'elle se termine par la valeur nulle (0) qui ne fait pas partie de la série.

2.1 Écrire un programme qui détermine la moyenne des valeurs de la série.

Solution : Pour calculer la moyenne, j'ai besoin de la somme des valeurs et du nombre de valeurs. On peut donc en déduire les deux variables d'accumulation suivantes :

```

1  Variables
2      nb: Entier  -- le nombre de valeurs lues de la série
3      somme: Réel -- la somme des valeurs lues de la série

```

Les premiers niveaux de raffinement peuvent être présentés ainsi :

```

1  R0 : Afficher la moyenne d'une série de valeurs réelles
2
3  R1 : Raffinage De « Afficher la moyenne d'une série de valeurs réelles »
4      | Déterminer le nb d'éléments dans la série et leur somme
5      | Afficher le résultat
6
7  R2 : Raffinage De « Déterminer le nb d'éléments dans la série et leur somme »
8      | Initialiser : somme <- 0 et nb <- 0
9      | Saisir le premier réel x
10     | TantQue x <> 0 Faire
11     |     | Mettre à jour les variables somme et nb en fonction de x
12     |     | Saisir le réel suivant x
13     | FinTQ
14
15  R2 : Raffinage De « Afficher le résultat »
16     | Si nb > 0 Alors
17     |     | Afficher la moyenne = somme / nb
18     | Sinon
19     |     | Signaler moyenne impossible
20     | FinSi

```

Remarque : On pouvait pas utiliser un **Pour** car on ne connaît pas à priori le nombre de valeurs dans la série. Ceci dépend de l'utilisateur. C'est lui qui décide de saisir 0. On doit donc utiliser soit un **TantQue**, soit un **Répéter**.

Nous avons utilisé un **TantQue** car nous avons considéré « traiter une valeur » qui peut ne pas être exécutée (cas où l'utilisateur saisit tout de suite 0 pour indiquer que la série est vide).

Nous aurions également pu utiliser un **Répéter** en nous intéressant à la saisie d'un réel. L'utilisateur doit nécessairement saisir au moins un réel (une valeur de la série ou zéro). Notons que dans ce cas, il faut ensuite faire un test pour savoir si le réel saisi fait partie de la série et, dans l'affirmative, le comptabiliser.

Voici le raffinement correspondant.

```

1  Initialiser les variables statistiques avec x
2  Répéter
3      | Saisir un réel                x: out
4      | Si x <> 0 Alors    { x est une valeur de la série }
5      |     | Mettre à jour les variables statistiques

```

```

6   | FinSi
7   Jusqu'à x = 0

```

On remarque que dans le cas du **TantQue** on duplique une instruction (la saisie d'un réel) et dans le cas du **Répéter** on duplique un condition.

L'algorithme peut ensuite être le suivant :

```

1  Algorithme statistiques_simples_moyenne
2
3      -- Afficher la moyenne d'une série de valeurs réelles lues au clavier.
4      -- La série est terminée par zéro qui n'appartient pas à la série.
5
6  Variables
7      x: Réel      -- un réel lu au clavier
8      nb: Entier  -- le nombre de valeurs dans la série
9      somme: Réel -- la somme des valeurs lues de la série
10
11 Début
12     -- Déterminer le nb d'éléments dans la série et leur somme
13     -- Initialiser les variables
14     somme <- 0 -- pas encore d'éléments lus
15     nb <- 0
16     -- Saisir le premier réel
17     Lire(x)
18     -- Traiter les éléments de la série
19     TantQue x <> 0 Faire
20         -- Mettre à jour les variables somme et nb en fonction de x
21         somme <- somme + x
22         nb <- nb + 1
23
24         -- Saisir le réel suivant
25         Lire(x)
26     FinTQ
27
28     -- Afficher le résultat
29     Si nb > 0 Alors                                { La moyenne a un sens }
30         -- Afficher la moyenne
31         ÉcrireLn("Moyenne_=", somme / nb)
32     Sinon                                           { La moyenne n'a PAS de sens }
33         -- Signaler moyenne impossible
34         ÉcrireLn("La_série_est_vide._La_moyenne_n'existe_donc_pas!")
35     FinSi
36 Fin.

1  Attribute VB_Name = "exo2_statistiques_clavier_moy"
2  '*****
3  '* Auteur : Claude Monteil <monteil@ensat.fr>
4  '* Version : 1.0
5  '* Objectif :
6  '*   Afficher la moyenne d'une série de valeurs réelles lues au clavier.
7  '*****
8

```

```

9 Sub statistiques_clavier()
10
11 Dim x As Double ' un reel lu au clavier
12 Dim nb As Integer ' le nombre de valeurs lues de la serie
13 Dim somme As Double ' la somme des valeurs lues de la serie
14
15 EffacerEcran "Moyenne_d'une_serie_de_valeurs"
16 '1.afficher la consigne
17 Afficher "Donnez,_en_colonne,_une_serie_d'entiers_qui_se_termine_par_0."
18
19 '2.lire le premier reel
20 Saisir x
21
22 If (x = 0) Then ' Pas de valeur dans la serie
23 Afficher "La_serie_est_vider"
24 Afficher "Les_statistiques_demandees_n'ont_pas_de_sens!"
25 Else
26 '3.initialiser les variables statistiques avec x
27 somme = x
28 nb = 1
29
30 '4.lire une nouvelle valeur x
31 Saisir x
32
33 Do While x <> 0
34 '5.mettre à jour les variables statistiques
35 nb = nb + 1
36 somme = somme + x
37 '6.lire une nouvelle valeur x
38 Saisir x
39 Loop
40 '7.afficher les statistiques
41 Afficher "Moyenne=_ " & somme / nb
42
43 End If
44
45 End Sub
46
47 '
48 ' 1 2 3 0 --> 2
49 ' 2 -2 0 --> 0
50 ' -4 -2 0 --> -3
51 ' 13 0 --> 13
52 ' 0 --> Non defini

```

2.2 En plus de la moyenne, on veut connaître la plus petite et la plus grande valeur de la série.

Solution : Pour calculer la plus grande et la plus petite valeur, je pourrais procéder comme pour la moyenne en identifiant les deux variables min et max.

```

1 Variables
2 min: Réel -- la plus petite des valeurs lues de la série
3 max: Réel -- la plus grande des valeurs lues de la série

```

Le problème est de savoir avec quelle valeur initialiser max et min. Particulariser une valeur est toujours dangereux. Le mieux est donc de les initialiser avec la première valeur de la série... à condition que cette valeur existe. Le test de la série vide n'est donc plus fait *a posteriori* mais *a priori*.

Les premiers niveaux de raffinement sont alors :

```

1  R0 : Afficher des statistiques sur une série de valeurs réelles
2
3  R1 : Comment « Afficher des statistiques sur une série de valeurs réelles »
4      | Saisir la première valeur x
5      | Si x est nulle Alors
6      | | Indiquer que les statistiques demandées ne peuvent pas être faites
7      | Sinon
8      | | Initialiser les variables statistiques avec x
9      | | Saisir une nouvelle valeur x
10     | | TantQue x est NON nulle Faire
11     | | | Mettre à jour les variables statistiques
12     | | | Saisir une nouvelle valeur x
13     | | FinTQ
14     | | Afficher les statistiques
15     | FinSi

```

L'algorithme peut ensuite être le suivant :

```

1  Algorithme statistiques_simples_max
2
3      -- Afficher la moyenne, la plus grande et la plus petite valeur
4      -- d'une série de valeurs réelles lues au clavier.
5      -- La série est terminée par zéro qui n'appartient pas à la série.
6
7  Variables
8      x: Réel      -- un réel lu au clavier
9      nb: Entier   -- le nombre de valeurs lues de la série
10     somme: Réel  -- la somme des valeurs lues de la série
11     min: Réel   -- la plus petite des valeurs lues de la série
12     max: Réel   -- la plus grande des valeurs lues de la série
13
14  Début
15     -- Saisir le premier réel x
16     Lire(x)
17
18     Si x = 0 Alors
19         -- Indiquer que les statistiques demandées ne peuvent être faites
20         ÉcrireLn("La_série_est_vide.");
21         ÉcrireLn("Les_statistiques_demandées_n'ont_pas_de_sens_!")
22     Sinon
23         -- Initialiser les variables statistiques avec x
24         max <- x
25         min <- x
26         somme <- x
27         nb <- 1
28

```



```
29      -- Saisir une nouvelle valeur x
30      Lire(x)
31
32      TantQue x <> 0 Faire
33          -- Mettre à jour les variables statistiques
34          nb <- nb + 1
35          somme <- somme + x
36          Si x > max Alors
37              max <- x
38          SinonSi x < min Alors
39              min <- x
40          FinSi
41
42          -- Saisir une nouvelle valeur x
43          Lire(x)
44      FinTQ
45
46      -- Afficher les statistiques
47      ÉcrireLn("Moyenne_=", somme / nb)
48      ÉcrireLn("Plus_petite_valeur_=", min)
49      ÉcrireLn("Plus_grande_valeur_=", max)
50  FinSi
51  Fin.
```

Exercice 3 : Division entière

Étant donnés deux entiers positifs lus au clavier, calculer le quotient et le reste de la division euclidienne du premier nombre par le deuxième. On utilisera uniquement l'addition et la soustraction sur les entiers.

Solution :

Remarque : En utilisant seulement l'addition et la soustraction, on ne peut pas utiliser une boucle Pour. Il faut donc choisir entre **TantQue** et **Répéter**.

```

1  Algorithme div_mod
2
3      -- Calculer le quotient (div) et le reste (mod) de la division entière de
4      -- deux entiers lus au clavier
5
6  Variables
7      dividende: Entier    -- dividende lu au clavier, positif
8      diviseur: Entier    -- diviseur lu au clavier, strictement positif
9      reste: Entier       -- reste de la division entière
10     quotient: Entier    -- quotient de la division entière
11
12  Début
13     -- saisir le dividende et le diviseur avec contrôle
14     ...
15
16     -- calculer le quotient et le reste
17     reste <- dividende
18     quotient <- 0
19     TantQue reste >= diviseur Faire
20         { Variant : reste }
21         { Invariant : diviseur * quotient + reste = dividende }
22         quotient <- quotient + 1
23         reste <- reste - diviseur
24     FinTQ
25
26     -- afficher le résultat
27     ÉcrireLn(dividende, "_/_", diviseur, "_=_",
28              quotient, "_*__", diviseur, "_+__", reste)
29  Fin.
30
31  --      5      2      -->    2 * 2 + 1
32  --     10     2      -->    5 * 2 + 0
33  --      5      0      -->    diviseur nul !

1  Attribute VB_Name = "exo3_div_mod"
2  '*****
3  '* Auteur   : Claude Monteil <monteil@ensat.fr>
4  '* Version  : 1.0
5  '* Objectif : Quotient et reste de la division entière
6  '*****
7
8  Sub div_mod()
9  ' Calcule le quotient (div) et le reste (mod) de la division entière

```

```

10 ' de deux entiers lus au clavier.
11   Dim dividende As Integer      ' dividende lu au clavier, positif
12   Dim diviseur As Integer      ' diviseur lu au clavier, strictement positif
13   Dim reste As Integer         ' reste de la division entiere
14   Dim quotient As Integer      ' quotient de la division entiere
15
16   EffacerEcran "Quotient_et_reste_d'une_division_entiere"
17   '1.saisir le dividende et le diviseur
18   Do
19       '1a.Saisir les valeurs
20       Afficher "Dividende:_:"
21       Saisir dividende
22       Afficher "Diviseur:_:"
23       Saisir diviseur
24       '1b.Verifier le dividende
25       If (dividende < 0) Then Afficher "Le_dividende_doit_etre_strictement_positif."
26       '1c.Verifier le diviseur
27       If (diviseur <= 0) Then Afficher "Le_diviseur_doit_etre_strictement_positif."
28   Loop Until (dividende >= 0) And (diviseur > 0)
29
30   '2.calculer le quotient et le reste
31   reste = dividende
32   quotient = 0
33   Do Until (reste < diviseur)
34       ' Variants : reste et quotient
35       ' Invariant : diviseur * quotient + reste = dividende
36       quotient = quotient + 1
37       reste = reste - diviseur
38   Loop
39   ' (reste < diviseur) et (diviseur * quotient + reste = dividende)
40
41   '3.afficher le resultat
42   Afficher dividende & "/" & diviseur & "=" & quotient & "*" & diviseur & "+" & reste
43 End Sub
44
45
46 '
47 '5      2      --> 2 * 2 + 1
48 '2      5      --> 0 * 5 + 2
49 '10     2      --> 5 * 2 + 0
50 '0      5      --> 0 * 5 + 0
51 '-2     -2     --> dividende non positif, diviseur non > 0
52 '-2     5      --> dividende non positif
53 '5      -2     --> diviseur non strictement positif
54 '5      0      --> Recommencez !

```

Exercice 4 : Nombres premiers

Écrire un programme qui permet à son utilisateur de saisir une valeur entière et qui, en retour lui indique si c'est un nombre premier ou pas.

Solution : Il s'agit d'afficher si un nombre saisi par l'utilisateur est premier ou non.

1 **R0** : Afficher si un nombre saisi est premier ou non

Pour ce qui concerne les jeux de test, on peut vérifier si le programme fonctionne sur les premiers nombres, par exemple de 1 à 20 ou de 1 à 100.

Le premier niveau de raffinement est classique. Il permet de clairement séparer la partie calcul de la partie interaction avec l'utilisateur.

```

1 R1 : Raffinage De « Afficher si un nombre saisi est premier ou non »
2   | Saisir un nombre                n: out Entier
3   | Déterminer si le nombre est premier  n: in ; premier: out Booléen
4   | Afficher le résultat            n, premier: in

```

Seule la deuxième étape mérite d'être détaillée. Un nombre premier est un nombre qui n'admet pas de diviseurs autres que 1 et lui-même. Nous allons en proposer plusieurs raffinements que nous allons comparer d'un point de vu performance (en nombre d'opérations arithmétiques).

Ainsi, étant donné un entier n , on peut regarder s'il est divisible par les entiers compris entre 2 et $n - 1$. L'idée est d'essayer chaque entier. Si on trouve un diviseur le n n'est pas premier. Si on ne trouve pas de diviseur, n est premier. On se sert donc de la variable booléenne que l'on initialise à *VRAI* et qui sera mise à faux dès que l'on trouve un diviseur. On pourrait donc l'écrire avec un **Pour**.

```

1 premier <- VRAI
2 Pour diviseur <- 1 JusquÀ diviseur = n-1 Faire
3   Si diviseur est un diviseur de n Alors
4     premier <- FAUX
5   FinSi
6 FinPour

```

Ce qui s'écrit de manière équivalente :

```

1 premier <- VRAI
2 Pour diviseur <- 1 JusquÀ diviseur = n-1 Faire
3   premier <- premier Et (diviseur est un diviseur de n)
4 FinPour

```

Si l'utilisation d'un **Pour** est possible, elle est cependant peu judicieuse. En effet, dès qu'on a trouvé un diviseur, on sait que le nombre n n'est pas premier et il est inutile d'essayer d'autres diviseurs. Nous devons donc utiliser un **TantQue** ou un **Répéter**. Nous optons pour un **TantQue**.

Remarquons que nous avons en fait initialisé la variable premier avec $n <> 1$ pour tenir compte du fait que 1 n'est pas un nombre premier.

Voici le raffinement correspondant.

```

1 R2 : Raffinage De « Déterminer si un nombre est premier »
2   | premier <- n > 1
3   | diviseur <- 2
4   | TantQue premier Et (diviseur < n - 1) Faire
5   |   premier <- Non diviseur est un diviseur de n

```

```

6   |     diviseur <- diviseur + 1
7   | FinTQ
8
9   R3 : Raffinage De « diviseur est un diviseur de n »
10  | Résultat <- n Mod diviseur = 0

1  Attribute VB_Name = "exo4_nb_premier"
2  '*****
3  '* Auteur   : Claude Monteil <monteil@ensat.fr>
4  '* Version  : 1.0
5  '* Objectif : programme qui demande un entier et indique s'il est premier
6  '*           (possibilité de répétition)
7  '*****
8
9  Sub nb_premier()
10     Dim rep As String      ' réponse utilisateur
11     Dim nombre As Integer ' nombre saisi au clavier
12     Dim premier As Boolean ' est-ce que nombre est premier ?
13     Dim diviseur As Integer ' parcourir les diviseurs potentiels de nombre
14
15     EffacerEcran "Test_de_nombres_premiers"
16     Do
17         '1.saisir le nombre
18         Afficher "Entrez_un_nombre:_:"
19         Saisir nombre
20
21         '2.déterminer si nombre est premier
22         premier = (nombre <> 1)
23         diviseur = 2
24         Do While (premier) And (diviseur < nombre)
25             premier = (nombre Mod diviseur <> 0)
26             diviseur = diviseur + 1
27         Loop
28
29         '3.afficher le résultat
30         If premier Then
31             Afficher "premier"
32         Else
33             Afficher "NON_premier"
34         End If
35
36         '4.demander si l'utilisateur veut recommencez
37         Afficher "Recommencer_(o/n)?_"
38         Saisir rep
39         Loop Until (rep = "n") Or (rep = "N")
40
41     End Sub

```

En fait, on sait qu'il n'est pas nécessaire de regarder les diviseurs au delà de $n/2$ car il ne peut pas y en avoir (trivial). On peut donc reformuler la condition du **TantQue**

| **TantQue** premier **Et** (diviseur <= n Div 2) **Faire**

Cette optimisation permet de réduire par 2 le nombre d'opérations. Cependant, on peut faire beaucoup mieux. En effet, si un nombre n'est pas premier il admet un diviseur et en fait au moins deux dont l'un au moins est inférieur ou égal à sa racine carrée.

```
| TantQue premier Et (diviseur <= racine carrée de n) Faire
```

Cette optimisation est bien meilleure que la précédente. En effet, si on considère un nombre premier supérieur à 10000, dans la version initiale, on doit considérer 10000 diviseurs, dans la première optimisation 5000 et seulement 100 dans la seconde.

Remarque : Pour éviter d'utiliser les réels et donc les problèmes d'arrondis (et éventuellement une perte de performance), on peut transformer l'expression

```
diviseur <= racine carrée de n
```

par (en élevant au carré)

```
diviseur * diviseur <= n
```

et, pour éviter les débordements (diviseur * diviseur peut dépasser la capacité des entiers), il est préférable de réorganiser les calculs :

```
diviseur <= n Div diviseur
```

Une fois cette transformation réalisée, on obtient donc :

```
| TantQue premier Et (diviseur <= n Div diviseur) Faire
```

On peut encore améliorer l'algorithme. En effet, si on sait que le nombre n'est pas divisible par 2, il est inutile d'essayer les diviseurs pairs (4, 6, 9, etc.). On peut se limiter aux diviseurs impairs (3, 5, 7, 9, 11...). Voici le raffinage correspondant.

```
1 R2 : Raffinage De « Déterminer si un nombre est premier »
2 | Si n = 2 Alors
3 |   premier <- VRAI
4 | Sinon
5 |   premier <- (n >= 2) Et Non (2 divise n)
6 |   diviseur <- 3
7 |   TantQue premier Et (diviseur <= n Div diviseur) Faire
8 |     premier <- Non diviseur est un diviseur de n
9 |     diviseur <- diviseur + 2
10 |   FinTQ
11 | FinSi
```

Remarque : On pourrait se passer du **Si** en initialisant premier de la manière suivante :

```
1 premier = (n == 2) Ou ((n >= 3) Et (n Mod 2 <> 0))
```

```
1 Attribute VB_Name = "exo4b_nb_premier_optimise"
2 '*****
3 '* Auteur : Claude Monteil <monteil@ensat.fr>
4 '* Version : 1.0
5 '* Objectif : programme qui demande un entier et indique s'il est premier
6 '* (possibilité de répétition)
7 ' *****
8
```

```

9 Sub nb_premier_optimise()
10 Dim rep As String ' réponse utilisateur
11 Dim nombre As Integer ' nombre saisi au clavier
12 Dim premier As Boolean ' est-ce que nombre est premier ?
13 Dim diviseur As Integer ' parcourir les diviseurs potentiels de nombre
14 Dim limite As Integer ' le plus grand diviseur à essayer
15
16 EffacerEcran "Test_(amélioré)_de_nombres_premiers"
17 Do
18 '1.saisir le nombre
19 Afficher "Entrez_un_nombre_:"
20 Saisir nombre
21
22 '2.déterminer si nombre est premier
23 premier = (nombre = 2) Or (nombre >= 3 And (nombre Mod 2 <> 0))
24 diviseur = 3
25 limite = Round(Sqr(nombre))
26 Do While (premier) And (diviseur <= limite)
27 premier = (nombre Mod diviseur <> 0)
28 diviseur = diviseur + 2
29 Loop
30
31 '3.afficher le résultat
32 If premier Then
33 Afficher "premier"
34 Else
35 Afficher "NON_premier"
36 End If
37
38 '4.demander si l'utilisateur veut recommencez
39 Afficher "Recommencer_(o/n)_?"
40 Saisir rep
41 Loop Until (rep = "n") Or (rep = "N")
42
43 End Sub

```

Modifier le programme pour qu'il propose à l'utilisateur d'entrer une autre valeur à traiter ou d'arrêter le programme.

Solution : Le raffinement est le suivant :

```

1 R0 : Indiquer si des entiers saisis au clavier sont premiers ou non
2
3 R1 : Raffinage De « R0 »
4   | Répéter
5   | | Saisir un entier
6   | | Indiquer le caractère premier de l'entier
7   | | Demander à l'utilisateur s'il veut continuer
8   | Jusqu'À réponse = 'n'

```

On peut en déduire le raffinement suivant :

```

1 Algorithme nb_premier
2

```

```

3      -- Indiquer si un nombre est premier où non
4      -- Possibilité de recommencer
5
6  Variables
7      n: Entier           -- parcourir les entiers de 1 à max
8      i: Entier           -- parcourir les diviseurs potentiels de n
9      premier: Booléen    -- n est-il premier
10     réponse: Caractère  -- réponse de l'utilisateur (o/n)
11
12  Début
13     Répéter              -- analyser un nombre de l'utilisateur
14         -- saisir le nombre
15         ÉcrireLn ("J'indique_si_un_nombre_est_premier")
16         Écrire ("Le_nombre:_")
17         Lire (n)
18
19         -- Déterminer si n est premier
20         Si n <= 3 Alors
21             premier <- n > 0           -- 0 n'est pas premier
22         Sinon
23             premier <- ((n mod 2) <> 0) -- n Non divisible par 2
24                 Et ((n mod 3) <> 0)   -- n Non divisible par 3
25             i <- 3
26             TantQue premier Et (i < n Div i) Faire
27                 -- n peut encore être premier
28                 -- et il reste des diviseurs potentiels
29                 i <- i + 2
30                 premier <- (n mod i) <> 0   -- n Non divisible par i
31             FinTQ
32         FinSi
33
34         -- afficher le résultat
35         Si premier Alors
36             ÉcrireLn ("OUI,_" & n & "_est_premier")
37         Sinon
38             ÉcrireLn ("NON,_" & n & "_n'est_pas_premier")
39         FinSi
40
41         -- Demander à l'utilisateur si il veut continuer
42         Écrire ("Encore_un_nombre_(o/n)?_")
43         Lire (réponse)
44     JusquÀ réponse = 'n'
45 Fin.

```


Exercice 5 : Nombres parfaits

Écrire un programme qui affiche tous les nombres parfaits compris entre 2 et 1000.

Un nombre parfait est un entier égal à la somme de ses diviseurs, lui exclu. Par exemple, 28 est un nombre parfait ($28 = 1 + 2 + 4 + 7 + 14$).

Solution :

```

1  R0 : Afficher les nombres parfaits compris entre 2 et Max, lu au clavier
2
3  R1 : Raffinage De « R0 »
4      | Pour n <- 2 JusquÀ n = Max Faire
5      |     | Si n est parfait Alors
6      |     |     | Afficher n
7      |     |     FinSi
8      |     FinPour
9
10 R2 : Raffinage De « n est parfait »
11     | Calculer la somme des diviseurs de n (autre que 1 et n)
12     | Résultat <- n = somme des diviseurs
13
14 R3 : Raffinage De « Calculer la somme des diviseurs de n »
15     | somme <- 1
16     | Pour i <- 2 JusquÀ racine carrée de n Faire
17     |     | Si i diviseur de n Alors
18     |     |     | somme <- somme + i + (n Div i)
19     |     |     FinSi
20     |     FinPour
21     | Si n est un carré parfait Alors
22     |     | somme <- somme - racine carrée de n
23     |     FinSi

1  Attribute VB_Name = "exo5_nb_parfait"
2  '*****
3  '* Auteur   : Claude Monteil <monteil@ensat.fr>
4  '* Version  : 1.0
5  '* Objectif : Trouver les nombres parfaits entre 2 et 10000
6  '*****
7
8  Sub nb_parfaits()
9      Dim nombre As Integer
10     Dim somme As Integer
11     Dim borne As Integer
12     Dim diviseur As Integer
13
14     EffacerEcran "Nombres_perfaits_inférieurs_à_10000"
15     For nombre = 2 To 10000 'tester si nombre est parfait
16         borne = Round(Sqr(nombre))
17         ' si NB est diviseur de nombre alors (nombre / NB) est également
18         ' diviseur. Pour trouver l'ensemble des diviseurs, il suffit
19         ' donc d'aller jusqu'à Sqr(nombre).
20         somme = 1
21         For diviseur = 2 To borne 'tester si diviseur divise nombre

```

```
22         If (nombre Mod diviseur = 0) Then
23             somme = somme + diviseur + nombre / diviseur
24         End If
25     Next diviseur
26
27     ' a-t-on compte deux fois le terme borne ?
28     If (nombre = (borne * borne)) Then ' oui
29         somme = somme - borne
30     End If
31
32     If (nombre = somme) Then
33         Afficher "*****_" & nombre
34     End If
35 Next nombre
36 End Sub
37
38 ' Les premiers nombres parfaits sont :
39 ' 6 28 496 8128
```