Les bases : exercices résolus en F
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

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 raffinage 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) plustôt qu'une constante littéralle (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 seule 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 résulat
39     ÉcrireLn (lg_p, " p = ", lg_cm, " cm")
40 Fin
```

```
1  !*********************************************************************
2  !* Auteur : Denis Barreteau <Denis.Barreteau@ensiacet.fr>
3  !*  Version : 1.0
4  !* Revision : Xuan Meyer < XuanMi.Meyer@ensiacet.fr>
5  !*
6  !* Objectif : Conversion pouces/centimetres
7  !*
8  !* Remarque : Le sujet n'est pas entierement respecte car la partie
```

```
!* decimale est affichee même si elle est nulle !
!*****************************************************************

PROGRAM pouce_centimetre
REAL,PARAMETER::un_pouce=2.54
REAL:: valeur   ! valeur lue au clavier
CHARACTER(LEN=1):: unite  ! unite lue au clavier
! valeur et unité forment la longueur
REAL:: p  !longueur lue au clavier convertie en pouces
REAL:: cm  ! longueur lue au clavier convertie en centimètres
! Saisir la longueur
PRINT*,"Entrez une longueur (valeur et unite) :"
PRINT*,"unite : p (pouce) , m (metre) , c (centimetre)"
READ*, valeur, unite

! Calculer la longueur en pouces et en centimetres
SELECT CASE (unite)
        CASE("p","P")   ! longueur saisie en pouces
                p = valeur
                cm = valeur * un_pouce
        CASE("M","m")  ! longueur saisie en metres
                p = 100.0 * valeur / un_pouce
                cm = 100.0 * valeur

        CASE("c","C")  ! longueur saisie en centimetres
                p = valeur / un_pouce
                cm = valeur
        CASE DEFAULT
                p =0.0
                cm = 0.0
END SELECT
! Afficher les résultats
PRINT*, p,"pouces", cm,"cm"
!remarque
PRINT("(F7.2,A,F7.2,A)"), p," pouces", cm," cm"
END PROGRAM pouce_centimetre
```

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 :

```
Variables
    nb: Entier  -- le nombre de valeurs lues de la série
    somme: Réel -- la somme des valeurs lues de la série
```

Les premiers niveaux de raffinage peuvent être présentés ainsi :

```
R0 : Afficher la moyenne d'une série de valeurs réelles

R1 : Raffinage De « Afficher la moyenne d'une série de valeurs réelles »
      | Déterminer le nb d'éléments dans la série et leur somme
      | Afficher le résultat

R2 : Raffinage De « Déterminer le nb d'éléments dans la série et leur somme »
      | Initialiser : somme <- 0 et nb <- 0
      | Saisir le premier réel x
      | TantQue x <> 0 Faire
      |   | Mettre à jour les variables somme et nb en fonction de x
      |   | Saisir le réel suivant x
      | FinTQ

R2 : Raffinage De « Afficher le résultat »
      | Si nb > 0 Alors
      |   | Afficher la moyenne = somme / nb
      | Sinon
      |   | Signaler moyenne impossible
      | FinSi
```

Remarque : On pouvait pas utiliser un **Pour** car on ne connait 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 raffinage correspondant.

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

```
  | FinSi
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 :

```
Algorithme statistiques_simples_moyenne

    -- Afficher la moyenne d'une série de valeurs réelles lues au clavier.
    -- La série est terminée par zéro qui n'appartient pas à la série.

Variables
    x: Réel     -- un réel lu au clavier
    nb: Entier  -- le nombre de valeurs dans la série
    somme: Réel -- la somme des valeurs lues de la série

Début
    -- Déterminer le nb d'éléments dans la série et leur somme
    -- Initialiser les variables
    somme <- 0  -- pas encore d'éléments lus
    nb <- 0
    -- Saisir le premier réel
    Lire(x)
    -- Traiter les éléments de la série
    TantQue x <> 0 Faire
        -- Mettre à jour les variables somme et nb en fonction de x
        somme <- somme + x
        nb <- nb + 1

        -- Saisir le réel suivant
        Lire(x)
    FinTQ

    -- Afficher le résultat
    Si nb > 0 Alors                    { La moyenne a un sens }
        -- Afficher la moyenne
        ÉcrireLn("Moyenne = ", somme / nb)
    Sinon                              { La moyenne n'a PAS de sens }
        -- Signaler moyenne impossible
        ÉcrireLn("La série est vide.  La moyenne n'existe donc pas !")
    FinSi
Fin.
```

```
!************************************************************************
!*  Auteur  :  Denis Barreteau <Denis.Barreteau@ensiacet.fr>
!*  Version :  1.2
!*  Revision : Xuan Meyer < XuanMi.Meyer@ensiacet.fr>
!*
!*  Objectif : Afficher la moyenne d'une serie de valeurs reelles lues au
!*      clavier.
!************************************************************************
```

```
9
10 PROGRAM main
11
12     REAL :: x       ! un reel simple precision lu au clavier
13     INTEGER:: nb    ! le nombre de valeurs lues de la serie
14     REAL :: somme   ! la somme des valeurs lues de la serie
15
16     ! afficher la consigne
17     PRINT*,"Donnez, en colonne, une serie d'entiers qui se termine par 0."
18
19     ! determiner le nb d'elements dans la serie et leur somme
20     !    initialiser les variables
21     somme = 0               ! pas encore d'elements lus
22     nb = 0
23
24     !    lire le premier reel
25     READ*,x
26
27     !    traiter les elements de la serie
28     DO
29             IF(x == 0)EXIT
30             ! mettre a jour les variables somme et nb en fonction de x
31             somme = somme + x
32             nb=nb + 1
33
34             ! lire le reel suivant
35             READ*,x
36     ENDDO
37
38     ! afficher le resultat
39     IF (nb > 0)THEN          ! La moyenne a un sens
40         ! afficher la moyenne
41         PRINT*,"Moyenne = ", somme / nb
42     ELSE                        ! La moyenne n'a PAS de sens
43         ! signaler moyenne impossible
44         PRINT*,"La serie est vide.  La moyenne n'existe donc pas !"
45     ENDIF
46
47 END PROGRAM main
48
49
50 !
51 !    1 2 3 0        -->        2
52 !    2 -2 0        -->        0
53 !    -4 -2 0        -->        -3
54 !    13 0        -->        13
55 !    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.

```
Variables
    min: Réel   -- la plus petite des valeurs lues de la série
    max: Réel   -- la plus grande des valeurs lues de la série
```

Le problème est de savoir avec quelle valeur intialiser 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 raffinage sont alors :

```
R0 : Afficher des statistiques sur une série de valeurs réelles

R1 : Comment « Afficher des statistiques sur une série de valeurs réelles »
      | Saisir la première valeur x
      | Si x est nulle Alors
      |   | Indiquer que les statistiques demandées ne peuvent pas être faites
      | Sinon
      |   | Initialiser les variables statistiques avec x
      |   | Saisir une nouvelle valeur x
      |   | TantQue x est NON nulle Faire
      |   |   | Mettre à jour les variables statistiques
      |   |   | Saisir une nouvelle valeur x
      |   | FinTQ
      |   | Afficher les statistiques
      | FinSi
```

L'algorithme peut ensuite être le suivant :

```
Algorithme statistiques_simples_max

    -- Afficher la moyenne, la plus grande et la plus petite valeur
    -- d'une série de valeurs réelles lues au clavier.
    -- La série est terminée par zéro qui n'appartient pas à la série.

Variables
    x: Réel     -- un réel lu au clavier
    nb: Entier  -- le nombre de valeurs lues de la série
    somme: Réel -- la somme des valeurs lues de la série
    min: Réel   -- la plus petite des valeurs lues de la série
    max: Réel   -- la plus grande des valeurs lues de la série

Début
    -- Saisir le premier réel x
    Lire(x)

    Si x = 0 Alors
        -- Indiquer que les statistiques demandées ne peuvent être faites
        ÉcrireLn("La série est vide.");
        ÉcrireLn("Les statistiques demandées n'ont pas de sens !")
    Sinon
        -- Initialiser les variables statistiques avec x
        max <- x
        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.
```

```
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
8   !*        d'une serie de valeurs reelles lues au clavier.
9   !************************************************************************
10
11  PROGRAM main
12
13      REAL:: x        ! un reel lu au clavier
14      INTEGER:: nb    ! le nombre de valeurs lues de la serie
15      REAL:: somme    ! la somme des valeurs lues de la serie
16      REAL:: min      ! la plus petite des valeurs lues de la serie
17      REAL:: max      ! la plus grande des valeurs lues de la serie
18
19      ! afficher la consigne
20      PRINT*,"Donnez, en colonne, une serie d'entiers qui se termine par 0."
21
22      ! lire le premier reel
23      READ*,x
24
25      IF (x == 0) THEN        ! Pas de valeur dans la serie
```

```
26          PRINT*,"La serie est vide.  "
27          PRINT*,"Les statistiques demandees n'ont pas de sens !"
28      ELSE
29          ! initialiser les variables statistiques avec x
30          max = x
31          min = x
32          somme = x
33          nb = 1
34
35          ! lire une nouvelle valeur x
36          READ*,x
37
38          DO
39              ! mettre à jour les variables statistiques
40              nb=nb+1
41              somme = somme + x
42              IF (x > max) THEN
43                  max = x
44              ELSEIF (x < min) THEN
45                  min = x
46              ENDIF
47              ! lire une nouvelle valeur x
48              READ*, x
49              IF(x==0)EXIT
50          ENDDO
51          ! afficher les statistiques
52          PRINT*,"Moyenne = ", somme / nb
53          PRINT*,"Plus petite valeur = ", min
54          PRINT*,"Plus grande valeur = ", max
55
56      ENDIF
57
58  END PROGRAM main
59
60  !                          moyenne, min, max
61  !    1 2 3 0         -->         2,         1,        3
62  !    2 -2 0         -->         0,         -2,         2
63  !    -4 -2 0         -->         -3,         -4,         -2
64  !    13 0         -->         13,         13,         13
65  !    0                -->         Non defini
66  !    1 3 1 3 0         -->         2         1         3
67  !    3 3 3 3 0         -->         3         3         3
```

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**.

```
Algorithme div_mod

    -- Calculer le quotient (div) et le reste (mod) de la division entière de
    -- deux entiers lus au clavier

Variables
    dividende: Entier   -- dividende lu au clavier, positif
    diviseur: Entier    -- diviseur lu au clavier, strictement positif
    reste: Entier       -- reste de la division entière
    quotient: Entier    -- quotient de la division entière

Début
    -- saisir le dividende et le diviseur avec contrôle
    ...

    -- calculer le quotient et le reste
    reste <- dividende
    quotient <- 0
    TantQue reste >= diviseur Faire
        { Variant : reste }
        { Invariant : diviseur * quotient + reste = dividende }
        quotient <- quotient + 1
        reste <- reste - diviseur
    FinTQ

    -- afficher le résultat
    ÉcrireLn(dividende, " / ", diviseur, " = ",
            quotient, " * ", diviseur, " + ", reste)
Fin.

--      5       2       -->     2 * 2 + 1
--      10      2       -->     5 * 2 + 0
--      5       0       -->     diviseur nul !
```

```
!*********************************************************************
!*  Auteur   : Denis Barreteau <Denis.Barreteau@ensiacet.fr>
!*  Version  : 1.3
!*  Revision : Xuan Meyer < XuanMi.Meyer@ensiacet.fr>
!*
!*  Objectif : Quotient et reste de la division entiere
!*
! *********************************************************************

```

```
10 ! Calculer le quotient (div) et le reste (mod) de la division entiere
11 ! de deux entiers lus au clavier.
12
13 PROGRAM main
14     INTEGER:: dividende       ! dividende lu au clavier, positif
15     INTEGER:: diviseur        ! diviseur lu au clavier, strictement positif
16     INTEGER:: reste           ! reste de la division entiere
17     INTEGER:: quotient        ! quotient de la division entiere
18
19     ! saisir le dividende et le diviseur
20     DO
21         PRINT*,"Dividende␣:␣"
22         READ*,dividende
23         PRINT*,"Diviseur␣:␣"
24         READ*,diviseur
25
26         ! Verifier le dividende
27         IF (dividende < 0)PRINT*,"Le␣dividende␣doit␣etre␣strictement␣positif.␣␣"
28
29         ! Verifier le diviseur
30         IF (diviseur <= 0)PRINT*,"Le␣diviseur␣doit␣etre␣strictement␣positif.␣␣"
31
32         IF (dividende >= 0 .AND. diviseur > 0)EXIT
33     ENDDO
34
35     ! calculer le quotient et le reste
36     reste = dividende
37     quotient = 0
38     DO
39     IF (reste < diviseur)EXIT
40         ! Variant : reste
41         ! Invariant : diviseur * quotient + reste = dividende
42         quotient=quotient+1
43         reste = reste - diviseur
44     ENDDO
45
46     !{ (reste < diviseur)  & (diviseur * quotient + reste == dividende) }
47
48     ! afficher le resultat
49     PRINT*, dividende,"/", diviseur,"=",quotient,"*", diviseur,"+", reste
50
51 END PROGRAM main
52
53
54 !
55 !5        2        --> 2 * 2 + 1
56 !2        5        --> 0 * 5 + 2
57 !10        2        --> 5 * 2 + 0
58 !0        5        --> 0 * 5 + 0
59 !-2        -2        --> dividende non positif, diviseur non > 0
60 !-2        5        --> dividende non positif
61 !5        -2        --> diviseur non strictement positif
```

```
62 !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.

```
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 raffinage est classique. Il permet de clairement séparer la partie calcul de la partie interaction avec l'utilisateur.

```
R1 : Raffinage De « Afficher si un nombre saisi est premier ou non »
  | Saisir un nombre                          n: out Entier
  | Déterminer si le nombre est premier       n: in ; premier: out Booléen
  | 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**.

```
premier <- VRAI
Pour diviseur <- 1 JusquÀ diviseur = n-1 Faire
    Si diviseur est un diviseur de n Alors
        premier <- FAUX
    FinSi
FinPour
```

Ce qui s'écrit de manière équivalente :

```
premier <- VRAI
Pour diviseur <- 1 JusquÀ diviseur = n-1 Faire
    premier <- premier Et (diviseur est un diviseur de n)
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'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 raffinage correspondant.

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

```
  |     diviseur <- diviseur + 1
  | FinTQ

R3 : Raffinage De « diviseur est un diviseur de n »
  | Résultat <- n Mod diviseur = 0
```

```
!**********************************************************************
!*  Auteur   : Denis Barreteau <Denis.Barreteau@ensiacet.fr>
!*  Version  : 1.1
!*  Revision : Xuan Meyer < XuanMi.Meyer@ensiacet.fr>
!*
!*  Objectif : programme qui demande un entier et indique s'il est premier
!*             possibilite de repetition
! **********************************************************************

PROGRAM main
    CHARACTER(LEN=1)::rep   ! réponse utilisateur
    INTEGER:: nombre        ! nombre saisi au clavier
    LOGICAL::premier        ! est-ce que nombre est premier ?
    INTEGER:: diviseur      ! parcourir les diviseurs potentiels de nombre

    DO
        ! saisir le nombre
        PRINT*,"Entrez un nombre : "
        READ*,nombre

        ! déterminer si nombre est premier
        premier = ( nombre /= 1)
        diviseur = 2
        DO
            IF(.NOT.premier .OR. (diviseur>= nombre))EXIT
            premier = ((nombre/diviseur*diviseur) /= nombre)
            diviseur=diviseur+1

        ENDDO

        ! afficher le résultat
        IF (premier) THEN
            PRINT*,"premier"
        ELSE
            PRINT*,"NON premier"
        ENDIF
        PRINT*

        ! demander si l'utilisateur veut recommencez
        PRINT*,"Recommencer (o/n) ? "
        READ*,rep
        IF(rep == "n")EXIT
    ENDDO
END PROGRAM main
```

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 préférrable de réorganiser les calculs :

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

Une fois cette fransformation 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  !**********************************************************************
2  ! *  Auteur   : Denis Barreteau <Denis.Barreteau@ensiacet.fr>
3  ! *  Version  : Revision
4  ! *  Objectif : programme qui demande un entier et indique s'il est premier.
5  ! **********************************************************************
```

```
6  !*
7  PROGRAM main
8      CHARACTER(LEN=1)::rep                ! reponse utilisateur
9      INTEGER:: nombre                  ! nombre saisi au clavier
10     LOGICAL::premier          ! est-ce que nombre est premier ?
11     INTEGER:: diviseur          ! parcourir les diviseurs potentiels de nombre
12     REAL:: limite          ! le plus grand diviseur à essayer
13
14     DO
15         ! saisir le nombre
16         PRINT*,"Entrez␣un␣nombre␣:␣"
17         READ*,nombre
18
19         ! determiner si nombre est premier
20         premier = (nombre == 2) .OR. (nombre >= 3 .AND. (nombre / 2 *2 /= nombre))
21         diviseur = 3
22         limite = SQRT(REAL(nombre))+1.0
23         DO
24             IF(.NOT.premier .OR. diviseur>=limite)EXIT
25             premier = (nombre/diviseur*diviseur /= nombre)
26             diviseur = diviseur + 2
27         ENDDO
28
29         ! afficher le resultat
30         IF (premier)THEN
31             PRINT*,"premier"
32         ELSE
33             PRINT*,"NON␣premier"
34         ENDIF
35         PRINT*
36
37         ! demander si l'utilisateur veut recommencez
38         PRINT*,"Recommencer␣(o/n)␣?␣"
39         READ*,rep
40         IF(rep == "n")EXIT
41     ENDDO
42
43 END PROGRAM main
```

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

Solution : Le raffinage 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        | JusquÀ réponse = 'n'
```

On peut en déduire le raffinage suivant :

```
Algorithme nb_premier

    -- Indiquer si un nombre est premier où non
    -- Possibilité de recommencer

Variables
    n: Entier           -- parcourir les entiers de 1 à max
    i: Entier           -- parcourir les diviseurs potentiels de n
    premier: Booléen    -- n est-il premier
    réponse: Caractère  -- réponse de l'utilisateur (o/n)

Début
    Répéter             -- analyser un nombre de l'utilisateur
        -- saisir le nombre
        ÉcrireLn ("J'indique si un nombre est premier")
        Écrire ("Le nombre : ")
        Lire (n)

        -- Déterminer si n est premier
        Si n <= 3 Alors
            premier <- n > 0            -- 0 n'est pas premier
        Sinon
            premier <- ((n mod 2) <> 0) -- n Non divisible par 2
                    Et  ((n mod 3) <> 0)        -- n Non divisible par 3
            i <- 3
            TantQue premier Et (i < n Div i) Faire
                    -- n peut encore être premier
                    -- et il reste des diviseurs potentiels
                i <- i + 2
                premier <- (n mod i) <> 0       --  n Non divisible par i
            FinTQ
        FinSi

        -- afficher le résultat
        Si premier Alors
            ÉcrireLn ("OUI, ", n, " est premier")
        Sinon
            ÉcrireLn ("NON, ", n, " n'est pas premier")
        FinSi

        -- Demander à l'utilisateur si il veut continuer
        Écrire ("Encore un nombre (o/n) ? ")
        Lire (réponse)
    JusquÀ réponse = 'n'
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 :

```
R0 : Afficher les nombres parfaits compris entre 2 et Max, lu au clavier

R1 : Raffinage De « R0 »
      | Pour n <- 2 JusquÀ n = Max Faire
      |   | Si n est parfait Alors
      |   |   | Afficher n
      |   | FinSi
      | FinPour

R2 : Raffinage De « n est parfait »
      | Calculer la somme des diviseurs de n (autre que 1 et n)
      | Résultat <- n = somme des diviseurs

R3 : Raffinage De « Calculer la somme des diviseurs de n »
      | somme <- 1
      | Pour i <- 2 JusquÀ racine carrée de n Faire
      |   | Si i diviseur de n Alors
      |   |   | somme <- somme + i + (n Div i)
      |   | FinSi
      | FinPour
      | Si n est un carré parfait Alors
      |   | somme <- somme - racine carrée de n
      | FinSi
```

```
!*********************************************************************
!*  Auteur   : Denis Barreteau <Denis.Barreteau@ensiacet.fr>
!*  Version  : 1.1
!*  Revision : Xuan Meyer < XuanMi.Meyer@ensiacet.fr>
!*  Objectif : Trouver les nombres parfaits entre 2 et 1000000
!*********************************************************************

PROGRAM main
    INTEGER:: nombre
    INTEGER:: somme
    INTEGER:: borne
    INTEGER:: diviseur

    DO nombre = 2,1000000
            ! si NB est diviseur alors (nombre / NB) est egalement
            !  diviseur.  Pour trouver l'ensemble des diviseurs, il suffit
            !  donc d'aller jusqu'à sqrt(nombre).
        borne = SQRT(REAL(nombre))
        somme=1
        DO diviseur = 2, borne
            IF (nombre/diviseur*diviseur == nombre) THEN
```

```
22                  somme = somme + diviseur + nombre / diviseur
23              ENDIF
24          ENDDO
25
26          ! a-t-on compte deux fois le terme borne ?
27          IF (nombre == (borne * borne)) THEN        ! oui
28              somme = somme - borne
29          ENDIF
30
31          IF (nombre == somme) THEN
32              PRINT*,"*****␣", nombre
33          ENDIF
34      ENDDO
35
36  END PROGRAM main
37
38  ! Les premiers nombres parfaits sont :
39  ! 6 28 496 8128
```