Les types utilisateurs : exercices résolus en F
Corrigé

Objectifs

- Connaître les types utilisateurs : tableaux, enregistrements et type énumérés ;
- Connaître les algorithmes fondamentaux sur les tableaux ;
- Continuer à appliquer les principes de la semaine 1.

Exercice 1 : Occurrences des chiffres d'un entier 1
Exercice 2 : Modéliser un robot de type 1 3
Exercice 3 : Lecture d'un tableau 7
Exercice 4 : Tri par insertion séquentielle 8

Exercice 1 : Occurrences des chiffres d'un entier

Écrire un programme qui compte le nombre d'occurrences des 10 chiffres dans un entier naturel donné.

Par exemple, l'entier 4214 a une occurrence du chiffre 1, une de 2 et deux de 4.

Solution :

```
1  R0 : Compter le nombre d'occurrences des chiffres d'un nombre
```

Puisque l'objectif est de compter le nombre d'occurrences de chacun des chiffres, l'idée est donc de prendre autant de compteur que de chiffre. Dans ce cas, on prend un tableau de compteur indicé par les chiffres.

```
1  nb: Tableau [0..9] De Entier
2      -- nb[i] = nb d'occurrences de i dans le nombre.
```

On peut alors en déduire l'algorithme suivant :

```
1  R1 : Raffinage De « Compter le nombre d'occurrences des chiffres d'un nombre »
2     | Initialiser les compteurs à 0
3     | Répéter
4     |    | Comptabiliser l'unité de nombre
5     |    | Supprimer l'unité de nombre
6     | JusquÀ nombre = 0
7
8
9  R2 : Raffinage De « Initialiser les compteurs à 0 »
10    | Pour i <- 0 JusquÀ i = 9 Faire
11    |     nb[i] <- 0
12    | FinPour
13
14 R2 : Raffinage De « Comptabiliser l'unité de nombre »
```

```
15   | chiffre = nombre Mod 10
16   | nb[chiffre] <- nb[chiffre] + 1
17
18 R2 : Raffinage De « Supprimer l'unité de nombre »
19   | nombre <- nombre Div 10
```

```
1  !*******************************************************************
2  !* Auteur   : Denis Barreteau <Denis.Barreteau@ensiacet.fr>
3  !* Version  : 1.1
4  !* Revision : Xuan Meyer <XuanMi.Meyer@ensiacet.fr>
5  !* Objectif : Compter le nb d'occurrences des chiffres d'un nombre
6  !******************************************************************
7
8  PROGRAM main
9
10     INTEGER:: nombre            ! entier saisi par l'utilisateur
11     INTEGER,DIMENSION(0:9):: nb ! nb(i) : nb d'occurrences du chiffre i
12                                 ! dans nombre
13     INTEGER:: i                 ! pour parcourir les chiffres
14     INTEGER:: chiffre           ! chiffre unite de nombre
15
16     ! saisir le nombre
17     PRINT*,"Nombre : "
18     READ*,nombre
19
20     ! initialiser les compteurs
21     nb(:)=0
22
23     ! comptabiliser chaque chiffre de nombre
24     DO
25
26         ! comptabiliser l'unite de nombre
27         chiffre = nombre - nombre/10*10
28         nb(chiffre)=nb(chiffre) + 1
29
30         ! supprimer l'unite de nombre
31         nombre = nombre / 10
32
33         IF(nombre == 0)EXIT
34     ENDDO
35
36     ! afficher les occurrences
37     DO i = 0,9
38         PRINT*,"nb de ( ", i,")=", nb(i)
39     ENDDO
40
41 END PROGRAM main
```

Exercice 2 : Modéliser un robot de type 1

L'objectif de cet exercice est de modéliser un robot de type 1. Un tel robot se déplace dans un environnement qui peut être modélisé par un quadrillage dont chaque case correspond à une position possible du robot.

Un robot est donc caractérisé par sa position (son abscisse, x, et son ordonnée, y) et sa direction (nord, sud, est ou ouest). La figure 1 décrit un robot à la position $x = 4$ et $y = 2$, sa direction est « ouest ».

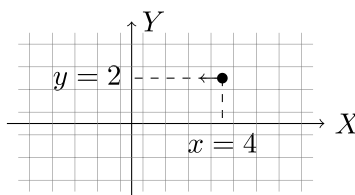

FIGURE 1 – Le robot dans un environnement illimité

Les robots de type 1 ont des possibilités réduites qui se limitent à pivoter de 90° vers la droite et à avancer d'une case suivant sa direction courante.

Pour simplifier, on considère que le robot évolue dans un environnement infini et sans obstacle.

2.1 Définir les types nécessaires pour modéliser un robot de type 1.

Solution : Un robot de type 1 est caractérisé par :

- une position, c'est-à-dire une abscisse (x) et une ordonnée. On peut donc définir un type Position comme un enregistrement.
- une direction. La direction peut prendre 4 valeurs que l'on peut par exemple appelées Nord, Sud, Est et Ouest. On aurait également pu les appeler Gauche, Droite, Haut et Bas. En conséquence, il s'agit d'un type énuméré.

Le type Robot1 est donc un enregistrement qui regoupe ces deux informations.

```
1  Type
2      Position =
3          Enregistrement
4              x: Entier   -- abscisse
5              y: Entier   -- ordonnée
6          FinEnregistrement
7
8      Direction = (NORD, SUD, EST, OUEST)
9
10     Robot1 =
11         Enregistrement
12             position: Position
13             direction: Direction
14         FinEnregistrement
```

2.2 L'environnement dans lequel évolue le robot est en fait fini et comporte des obstacles. On suppose qu'un obstacle occupe une case du quadrillage. La question est donc de savoir si la case est libre ou contient un obstacle.

Définir un type pour modéliser l'environnement.

Solution : L'environnement dans lequel évolue le robot est un quadrillage, donc il peut être modélisé par un tableau à deux dimensions. Nous utilisons deux constantes pour en préciser les dimensions (NB_COLONNES et NB_LIGNES).

Chaque case du tableau est soit une position qui peut être occupé par le robot, soit un mur, donc deux valeurs possibles. On peut donc représenter cette information par un type booléen (il faut choisir ce que signifie **VRAI** et **FAUX**) ou définir un type énuméré.

Nous choisissons ici cette deuxième solution :

```
1  Constante
2      NB_COLONNES = 20
3      NB_LIGNES = 30
4
5  Type
6      Case = (LIBRE, MUR)
7          -- indique si la case est LIBRE et peut donc être occupé par le robot
8          -- ou si elle contient un mur
9
10     Environnement = Tableau [1..NB_COLONNES, 1..NB_LIGNES] De Case
```

2.3 Écrire un programme qui fait avancer un robot toujours tout droit jusqu'à ce qu'il rencontre un obstacle ou qu'il arrive à la limite de son environnement. On supposera que le robot est initialement sur une position valide avec pour direction « est ».

Solution : On considère un robot particulier dans un environnement donné. Dans les programmes de test, il s'agira donc d'initialiser ces deux informations de manière à ce que l'on puisse tester notre algorithme dans des cas significatifs. Notons que la direction est fixé à l'EST.

```
1  R0 : Faire avancer le robot au maximum
2                  robot: in Robot1, environnement: in Environnement
3
4  tests :
5      - pas de mur => la limite est atteinte
6      - un mur => s'arrête devant le mur
7
8  R1 : Raffinage De « Faire avancer le robot au maximum »
9      TantQue (limite environnement non atteinte) et (Pas de mur) Faire
10         Faire avancer le robot d'une case (vers l'est)
11     FinTQ
12
13 R2 : Raffinage De « limite environnement non atteinte »
14     Résultat <- robot.position.x < NB_COLONNES
15
16 R2 : Raffinage De « Pas de mur »
17     Résultat <- environnement[robot.position.x + 1, robot.position.y] <> MUR
18
19 R2 : Raffinage De « Faire avancer le robot d'une case »
20     robot.position.x <- robot.position.x + 1
```

```
1  ! *******************************************************************
2  ! *  Auteur    : Denis Barreteau <Denis.Barreteau@ensiacet.fr>
3  ! *  Version   : 1.1
```

```
4  ! *  Objectif : Faire avancer un robot vers l'est au maximum
5  ! ****************************************************************
6
7  PROGRAM main
8
9  TYPE:: Posi
10     INTEGER:: x        ! abscisse
11     INTEGER:: y        ! ordonnée
12 END TYPE Posi
13
14 INTEGER,PARAMETER:: NORD=0, SUD=1, EST=2, OUEST=3
15 INTEGER,PARAMETER:: LIBRE=0, MUR=1
16
17 TYPE :: Robot1
18     TYPE(Posi):: position
19     INTEGER:: direction
20 END TYPE Robot1
21
22 INTEGER,PARAMETER::NB_COLONNES=20
23     INTEGER,PARAMETER::NB_LIGNES=30
24     INTEGER,DIMENSION(NB_COLONNES,NB_LIGNES)::environnement
25     TYPE(Robot1):: robot
26     INTEGER:: l, c ! parcourir les lignes et colonnes de l'environnement
27
28     ! Initialiser l'environnement
29     PRINT*,"Initialiser␣l'environnement..."
30         DO l = 1,NB_COLONNES
31             DO c = 1, NB_LIGNES
32                 environnement(l,c) = LIBRE
33             ENDDO
34         ENDDO
35         ! Mettre un mur
36         environnement(10,1:NB_LIGNES) = MUR
37
38     ! Initialiser le robot
39     PRINT*,"Initialiser␣le␣robot..."
40     robot%direction = EST
41     robot%position%x = 1
42     robot%position%y = 1
43
44
45     ! Faire avancer le robot
46     DO
47         IF(robot%position%x == NB_COLONNES   &
48          .OR.environnement(robot%position%x+1,robot%position%y) == MUR)EXIT
49
50
51         ! faire avancer le robot d'une case (vers l'est)
52         robot%position%x=robot%position%x+1
53
54         ! Afficher la position du robot
55         PRINT*,"Le␣robot␣est␣en␣:", robot%position%x, robot%position%y
```

```
56        ENDDO
57  END PROGRAM main
```

Exercice 3 : Lecture d'un tableau
Écrire un programme qui initialise un tableau d'entiers en lisant des valeurs au clavier. On considèrera que les valeurs sont saisies les unes après les autres et que la saisie s'arrête sur une valeur négative. La valeur négative ne doit pas être conservée dans le tableau.
Solution :

```
1  !*********************************************************************
2  !* Auteur   : Denis Barreteau <Denis.Barreteau@ensiacet.fr>
3  !* Version  : 1.1
4  !* Revision : Xuan Meyer <XuanMi.Meyer@ensiacet.fr>
5  !* Objectif :
6  !*       Saisir un tableau en s arretant sur une valeur negative.
7  !********************************************************************
8
9  PROGRAM main
10     INTEGER,PARAMETER::CAPACITE=10        ! capacite du tableau a lire
11     INTEGER,DIMENSION(CAPACITE):: tab   ! la tableau
12     INTEGER:: taille                  ! la taille effective du tableau
13     INTEGER:: valeur                  ! valeur lue au clavier
14     INTEGER:: i        ! pour parcourir les indices du tableau
15
16     ! Saisir le tableau
17     PRINT*, "Entrez␣les␣composantes␣du␣tableau"
18     PRINT*,"Entrez␣une␣valeur␣negative␣pour␣arreter."
19     taille = 0
20     READ*,valeur
21     DO
22         IF(valeur<0 .OR. taille>CAPACITE)EXIT
23         ! ranger la valeur
24         taille=taille+1
25         tab(taille) = valeur
26         ! Saisir une nouvelle valeur
27         READ*,valeur
28     ENDDO
29     IF (valeur >= 0)THEN
30         PRINT*,"Capactite␣du␣tableau␣atteinte.␣␣Arret␣de␣la␣saisie"
31     ENDIF
32
33
34     ! Afficher le tableau
35     PRINT*,"(/",(tab(i),",",i=1,taille-1),tab(taille),"/)"
36 END PROGRAM main
```

Exercice 4 : Tri par insertion séquentielle

Soit $A[1..N]$ un vecteur de N entiers relatifs quelconques, l'objectif est de trier le vecteur A. Le vecteur A est trié si $A[1] \leq A[2] \leq \cdots \leq A[N]$.

Le tri utilisé est le tri par insertion séquentielle. C'est un tri en $(N-1)$ étapes. L'étape i consiste à placer le $(i+1)^e$ élément du vecteur à sa place dans le sous-vecteur $A(1..i+1)$ sachant que $A(1..i)$ a été trié par les étapes précédentes. Dans le cas du tri par insertion séquentielle, la recherche de la position d'insertion, se fait séquentiellement en comparant successivement l'élément à insérer aux i premiers éléments du vecteur.

Exemple : Voici les différentes valeurs du vecteur 8 2 9 5 1 7 après chaque étape (la partie encadrée correspond à la partie du vecteur déjà traitée et donc triée) :

vecteur initial	:	[8] _2_ 9 5 1 7
après l'étape 1	:	[2 8] _9_ 5 1 7
après l'étape 2	:	[2 8 9] _5_ 1 7
après l'étape 3	:	[2 5 8 9] _1_ 7
après l'étape 4	:	[1 2 5 8 9] _7_
après l'étape 5	:	[1 2 5 7 8 9]

4.1 Écrire un programme qui lit une série de N ($N > 0$, lu au clavier) nombres relatifs, les range dans un vecteur A, les classe par ordre croissant en utilisant le tri par insertion séquentielle et, enfin, affiche la série ordonnée.

Solution :

```
1 R0 : Trier un tableau A de taille effective N (les indices sont 1..N).
```

On remarque qu'après chaque étape i du tri, les (i+1) premiers éléments sont à leur place. Plus précisément, le sous-tableau compris entre les indices 1 et i+1 est trié. Ainsi, au bout de la (N-1)^e étape l'ensemble du tableau est trié.

L'étape i consiste à trier le tableau 1..i+1 en sachant que le tableau 1..i est déjà trié.

```
1 R1 : Raffinage De << Trier un tableau A de taille effective N >>
2    | Pour indice := 2 JusquÀ indice = N Faire
3    |    Insérer A[indice] dans A(1..indice) sachant que A(1..indice) est trié
4    | FinPour
5
6 R2 : Raffinage De << Insérer A[indice] dans A(1..indice) sachant que A(1..indice) est t
7    | Déterminer la position théorique de A[indice] dans A(1..indice)
8    | Décaler les éléments compris entre la position théorique et indice-1
9    | Ranger l'élément A[indice]
```

Si on essaie d'exécuter mentalement ce raffinage [1], on constate que l'on a perdu la valeur A[indice] lorsque l'on veut la ranger. Elle a été écrasée lors du raffinage. Il est donc nécessaire de la conserver avant et donc de revoir notre raffinage.

```
1 R2 : Raffinage De << Insérer A[indice] dans A(1..indice) sachant que A(1..indice) est t
2    | Conserver la valeur de A[indice]                         memoire: out Entier
```

1. Ce qui doit toujours être fait, puisque ceci fait partie des choses à faire pour vérifier qu'un raffinage est correct.

```
3    | Déterminer la position pos de A[indice] dans A(1..indice)
4    | Décaler les éléments compris entre pos et indice-1
5    | Ranger l'élément mémorisé
6
7  R3 : Raffinage De << Déterminer la position théorique de A[indice] dans A(1..indice) >>
8    | pos <- 1
9    | TantQue (pos < indice ) Et (memoire >= A[pos]) Faire
10   |     pos <- pos + 1
11   | FinTQ
12
13 R3 : Raffinage De << Décaler les éléments compris entre la position théorique et indic
14   | Pour i <- indice - 1 Décrémenter JusquÀ i = pos Faire
15   |   A[i+1] <- A[i]
16   | FinPour
```

On peut en déduire l'algorithme.

```
1  Constante
2      CAPACITÉ = 100
3  Type
4      Vecteur = Tableau [1..CAPACITÉ] De Entier
5
6  Variable
7      A: Vecteur  -- le tableau à trier
8      N: Entier   -- le nombre d'éléments de A
9      indice: Entier      -- l'indice de l'élément à insérer
10     mémoire: Entier     -- mémoriser A[indice]
11     pos: Entier -- position de l'élément mémoire dans A(1..indice)
12     i: Integer; -- parcourir les éléments du tableau
13
14 Début
15     Pour indice := 2 JusquÀ indice = N Faire
16         -- Insérer A[indice] dans A(1..indice) sachant que A(1..indice) est trié
17         --   Conserver la valeur de A[indice]
18         mémoire <- A[indice]
19
20         --   Déterminer la position pos de A[indice] dans A(1..indice)
21         pos <- 1
22         TantQue (pos < indice ) Et (memoire >= A[pos]) Faire
23             pos <- pos + 1
24         FinTQ
25
26         --   Décaler les éléments compris entre pos  et indice-1
27         Pour i <- indice - 1 Décrémenter JusquÀ i = pos Faire
28             A[i+1] <- A[i]
29         FinPour
30
31         --   Ranger l'élément mémorisé
32         A[pos] <- mémoire
33     FinPour
34 Fin
```

On peut en écrire une version légèrement optimisée en réalisant les décalages en même temps que l'on cherche la position d'insertion.

```
Variable
    A: Vecteur  -- le tableau à trier
    N: Entier   -- le nombre d'éléments de A
    indice: Entier      -- l'indice de l'élément à insérer
    mémoire: Entier     -- mémoriser A[indice]
    pos: Entier -- pour chercher la position d'insertion dans A(1..indice)

Début
    Pour indice := 2 JusquÀ indice = N Faire
        -- Insérer A[indice] dans A(1..indice) sachant que A(1..indice) est trié
        --   Conserver la valeur de A[indice]
        mémoire <- A[indice]

        --   Faire de la place pour A[indice] dans A(1..indice)
        pos <- indice
        TantQue (pos > 1 ) Et (memoire < A[pos-1]) Faire
            A[pos] <- A[pos-1]
            pos <- pos - 1
        FinTQ

        --   Ranger l'élément mémorisé
        A[pos] <- mémoire
    FinPour
Fin
```

```
!*********************************************************************
!* Auteur   : Denis Barreteau <Denis.Barreteau@ensiacet.fr>
!* Version  : 1.1
!* Revision : Xuan Meyer <XuanMi.Meyer@ensiacet.fr>
!* Objectif : Trier un tableau par insertion sequentielle
!********************************************************************

PROGRAM main
    INTEGER,PARAMETER::MAX=20        ! Capacite maximale du tableau
    INTEGER,DIMENSION(MAX):: tab       ! la tableau a trier
    INTEGER:: nb                ! la taille effective du tableau a trier
    INTEGER:: valeur                ! valeur lue au clavier
    INTEGER:: indice                ! tab(0..indice) est trie
    INTEGER:: position        ! position d'insertion de tab(indice)
    INTEGER:: i                ! variable de boucle
    INTEGER:: memoire         ! conserver la valeur de tab(indice)

    ! Saisir le tableau
    PRINT*,"Entrez une valeur negative pour arreter."
    nb = 0
    READ*,valeur
    DO
        IF (valeur <0  .OR. nb == MAX)EXIT
        ! ranger la valeur
        nb=nb+1
```

```
26          tab(nb) = valeur
27
28          ! Saisir une nouvelle valeur
29          READ*,valeur
30      ENDDO
31      IF (valeur >= 0)THEN
32          PRINT*,"Capactite du tableau atteinte.  Arret de la saisie"
33      ENDIF
34
35      ! Trier le tableau
36      DO indice = 1,nb
37          ! Inserer tab(indice) dans tab(1..indice)
38
39          ! Conserver la valeur de tab(indice)
40          memoire = tab(indice)
41
42          ! determiner la position theorique de tab(indice)
43          position = 1
44          DO
45              IF (position >= indice .OR. memoire < tab(position))EXIT
46              position=position + 1
47          ENDDO
48
49          ! decaler les elements compris entre position et indice-1
50          DO i = indice-1,position,-1
51              tab(i+1) = tab(i)
52          ENDDO
53          ! ou bien sans boucle
54          !tab(position+1:indice)=tab(position:indice-1)
55
56          ! ranger l'element
57          tab(position) = memoire
58      ENDDO
59
60      ! Afficher le tableau
61      PRINT*,"(/",(tab(i),",",i=1,nb-1),tab(nb),"/)"
62  END PROGRAM main
```

4.2 En conservant le principe du tri par insertion, expliquer comment améliorer l'efficacité de cet algorithme.

Solution : Si on respecte le principe du tri par insertion, il faudra toujours réaliser les décalages. On ne peut donc gagner en performance qu'en optimisant la recherche (l'optimisation présentée précédemment n'est pas une optimisation réellement intéressante !). Comme on sait que le tableau A[1..indice-1] est trié, on peut utiliser une recherche par dichotomie pour rechercher la position théorique de l'élément dans le tableau.