Les sous-programmes (F)
Corrigé

Résumé

Ce document décrit les sous-programmes, outil essentiel pour structurer un programme car il permet au programmeur de définir ses propres instructions et opérateurs.

Table des matières

1 Les sous-programmes en F **2**

1 Les sous-programmes en F

En F, les notions de procédures et fonctions sont a peu près équivalente à leur sens algorithmique.

Cependant tous les sous-programmes doivent être inclus dans un MODULE. Ceci couvre des notions plus complexes, mais pour l'instant, nous n'utiliserons que le nécessaire.

```
MODULE machin
    Définition des constantes globales
    Définition des types dérivés envisagés dans le programme
    Variables globales éventuelles !!!!
CONTAINS
    Ensemble des sous-programmes
END MODULE machin
```

La première instruction du programme sera (après **PROGRAM** nom_du_programme) : **USE** machin.

Ceci a pour conséquence de rendre accessible les définitions de type et les variables globales au programme La syntaxe générale d'une fonction est

```
FUNCTION nom_fonction(paramètres_formels) RESULT (identificateur_resultat)
    Déclaration du type de résultat
    Déclaration du type des paramètres formels
                    et nature (entrée sortie, entrée/sortie)
    Déclaration des variables locales
    Instructions exécutables
END FUNCTION nom_fonction
```

En F, le type du résultat est quelconque (y compris tableau et enregistrement).

Toutefois dans cette initiation, nous vous conseillons d'écrire une fonction quand le type de retour est un scalaire uniquement. Tous les paramètres sont de types IN ce qui est conforme à une bonne écriture.

Un **RETURN** peut être écrit avant le **END FUNCTION**, mais il est facultatif. Plusieurs **RETURN** peuvent être utilisé en F, mais d'un point de vue algorithmique, ce n'est pas très bon.

La syntaxe générale d'une procédure est :

```
SUBROUTINE nom_sub(paramètres_formels)
    Déclaration du type des paramètres formels
                et nature (entrée sortie,entrée/sortie)
    Déclaration des variables locales
    Instructions exécutables
END SUBROUTINE nom_sub
```

L'appel à une procédure est :

```
CALL nom_sub(paramètres effectifs)
```

Notion publique et privé. En F, une variable globale, une constante globale, un enregistrement, une procédure peut être disponible à l'extérieur du module ou bien seulement utilisée dans le module. On dira qu'elle est **PUBLIC** si elle est disponible à l'extérieur du module et **PRIVATE** si elle n'est disponible que dans le module. On donnera cet attribut dans la déclaration pour les constantes et variables globales et dans une instruction spécifique pour les sous programmes.

```
MODULE machin
    INTEGER, PARAMETER, PUBLIC :: n=10 ! une constante globale exportée
    TYPE, PUBLIC :: tt ! une définition de type dérivé exporté
            ..
    END TYPE tt
    PUBLIC :: procedure   ! une procédure exportée
    PRIVATE :: procedure_2 ! une procédure privée du module
    CONTAINS
    SUBROUTINE procedure
            ..
    END SUBROUTINE procedure
    SUBROUTINE procedure_2
            ..
    END SUBROUTINE procedure_2
END MODULE machin
```

Exemple de fonctions :

```
!*****************************************************************
!* Auteur   : Denis Barreteau <Denis.Barreteau@ensiacet.fr>
!* Version  : 1.1
!* Revision : Xuan Meyer < XuanMi.Meyer@ensiacet.fr>
!* Objectif : Test de max2
!*****************************************************************
MODULE MAXI_MOD
PUBLIC :: max2
PUBLIC :: max3
CONTAINS

! Le plus grand des deux entiers a et b.
FUNCTION  max2(a,b) RESULT(resultat)
    INTEGER:: resultat       ! le type de la fonction
    INTEGER,INTENT(IN)::a,b  !ce sont obligatoirement des parametres d'entree
    IF (a > b) THEN
        resultat = a
    ELSE
        resultat = b
    ENDIF
END FUNCTION max2


! Le plus grand des trois entiers n1, n2 et n3.
FUNCTION max3( n1,  n2, n3) RESULT(resultat)
    INTEGER::resultat
    INTEGER,INTENT(IN)::n1,n2,n3
    INTEGER:: max12        ! le plus grand de n1 et n2
    max12 = max2(n1, n2)
    resultat= max2(max12, n3)
! Remarque on aurait pu remplacer dans les expressions max12 par resultat
END FUNCTION max3

END MODULE MAXI_MOD
! Programme de test de la fonction max2()
```

```
36
37
38 PROGRAM main
39 USE MAXI_MOD
40     INTEGER:: a = 2
41     INTEGER:: b = 5
42     INTEGER:: i, j
43
44     PRINT*,"a = et b = ", a, b
45     i = max2(a, b)        ! les deux parametres sont des variables
46     PRINT*," max2(a, b) = ",i
47     j = max2(8, 2*i)        ! mais peuvent etre des expressions quelconques
48     PRINT*,"max2(8, 2*i) = ", j
49     PRINT*,"max3(9, i, j) = ", max3(9, i, j)
50 END PROGRAM main
```

Le seul mode de passage des arguments en F est le passage par adresse. A l'appel d'une fonction ou d'une procédure, les adresses mémoires des paramètres effectifs sont empilés. Ceci permet le traitement adéquat des paramètres IN/OUT et OUT, puisque la procédure modifie la valeur de la variable. Pour les paramètres IN, F vérifie à la compilation que ce paramètre n'apparaît ni à gauche du signe égal dans le sous-programme ni dans une instruction READ, ce qui signifie qu'on ne la modifie pas. Si des arguments d'un sous programme sont les paramètres d'appels d'un autre sous-programme, il doit y avoir compatibilité de la notion d'entrée ou de sortie.

Quand, comme dans l'exemple précédent, on écrit une expression (j = max2(8, 2*i)), F fait le calcul de 2*i, stocke le résultat dans un emplacement mémoire et transmet cette adresse par la pile à la fonction. Bien sûr, cela n'aurait aucun sens de faire ceci pour un paramètre OUT ou IN/OUT.

Notez que le compilateur émet un message d'avertissement quand vous n'affectez pas de valeur à un paramètre OUT ou IN/OUT. Exemple de procédure :

```
1  !*****************************************************************************
2  !* Auteur   : Denis Barreteau <Denis.Barreteau@ensiacet.fr>
3  !* Version  : 1.2
4  !* Revision : Xuan Meyer < XuanMi.Meyer@ensiacet.fr>
5  !*
6  !* Objectif : Permuter la valeur de deux caracteres.
7  !*
8  !*****************************************************************************
9  MODULE PERMUTER_MOD
10
11 PUBLIC :: permuter
12
13 CONTAINS
14 ! Permuter la valeur des deux caracteres c1 et c2
15 SUBROUTINE permuter(c1,c2)
16     CHARACTER(LEN=1),INTENT(INOUT)::c1,c2
17     CHARACTER(LEN=1):: tmp        ! pour conserver la valeur de *c1
18     tmp = c1
19     c1 = c2
```

```
20      c2 = tmp
21  END SUBROUTINE permuter
22
23  END MODULE PERMUTER_MOD
24
25
26  PROGRAM main
27          USE PERMUTER_MOD
28          CHARACTER(LEN=1):: lettre = "A"
29          CHARACTER(LEN=1):: chiffre = "5"
30
31          PRINT*,"lettre␣=␣et␣chiffre␣=␣", lettre, chiffre
32          CALL permuter(lettre,chiffre)
33          PRINT*,"lettre␣=␣␣et␣chiffre␣=␣", lettre, chiffre
34
35  END PROGRAM main
```

Il est évident que c'est bien l'adresse des variables qui sont transmises au vu du résultat.
Cas des tableaux :

```
1   !*****************************************************************************
2   !* Auteur   : Denis Barreteau <Denis.Barreteau@ensiacet.fr>
3   !* Version  : 1.2
4   !* Revision : Xuan Meyer < XuanMi.Meyer@ensiacet.fr>
5   !*
6   !* Objectif : Tri par insertion.
7   !*
8   !*****************************************************************************
9   MODULE TRI_MOD
10
11  PUBLIC :: trier
12  PUBLIC :: afficher_tableau
13
14  CONTAINS
15
16  ! sous-programme de tri par insertion d'un vecteur
17  SUBROUTINE trier(tab,nb)
18      INTEGER,INTENT(IN)::nb  ! le nombre d'elements
19      INTEGER,DIMENSION(:),INTENT(INOUT)::tab ! le tableau est trie sur place
20      INTEGER:: indice
21      INTEGER::position
22      INTEGER::i ! variable de boucle
23      INTEGER:: memoire
24      DO indice=1,nb
25          ! conserver la valeur de tab(indice)
26          memoire=tab(indice)
27          ! determiner la position theorique de tab(indice)
28          position = 1
29          DO
30              IF(position>=indice .OR. memoire< tab(position))EXIT
31              position=position+1
32          ENDDO
33          !decaler les elements compris entre position et indice -1
```

```
34          DO i=indice-1,position,-1
35              tab(i+1)=tab(i)
36          ENDDO
37          ! ranger l'element
38          tab(position)=memoire
39      ENDDO
40  END SUBROUTINE trier
41
42  SUBROUTINE afficher_tableau(tab,nb)
43      INTEGER,INTENT(IN)::nb  ! le nombre d'elements
44      INTEGER,DIMENSION(:),INTENT(INOUT)::tab ! le tableau
45      INTEGER::i ! variable de boucle
46      PRINT*,"[",(tab(i),";",i=1,nb-1),tab(nb),"]"
47  END SUBROUTINE afficher_tableau
48
49  END MODULE TRI_MOD
50
51  PROGRAM main
52      USE TRI_MOD
53      INTEGER,DIMENSION(10)::tab=(/9,8,7,6,5,4,3,2,1,0/)
54      CALL afficher_tableau(tab,10)
55      CALL trier(tab(1:1),5)
56      CALL trier(tab(6:6),5)
57      CALL afficher_tableau(tab,10)
58      CALL trier(tab,10)
59      CALL afficher_tableau(tab,10)
60  END PROGRAM main
```

```
1  Résultats
2  [ 9 ; 8 ; 7 ; 6 ; 5 ; 4 ; 3 ; 2 ; 1 ; 0 ]
3  [ 5 ; 6 ; 7 ; 8 ; 9 ; 0 ; 1 ; 2 ; 3 ; 4 ]
4  [ 0 ; 1 ; 2 ; 3 ; 4 ; 5 ; 6 ; 7 ; 8 ; 9 ]
```

Il y a un sous-programme de tri de tableau et un sous programme d'affichage. Le premier appel : `CALL trier(tab(1:1),5)` indique de trier à partir du premier élément sur 5 éléments. Nous sommes obligé de mettre tab(1 :1) qui est un tableau à une valeur plutôt que tab(1) qui est un scalaire car F vérifie le type des arguments. On voit que les cinq premières valeurs sont triées puis les cinq suivantes par `CALL trier(tab(6:6),5)`.

Il n'existe pas en F de pré-conditions et post-conditions. La seule solution est d'effectuer des tests sur les variables d'entrée pour les pré-conditions et de sortie pour les post-conditions.