Les sous-programmes : exercices résolus en F
Corrigé

Objectifs

- Savoir écrire des sous-programmes ;
- Comprendre les modes de passage de paramètres ;
- Faire la différence entre fonction et procédure ;
- Continuer à appliquer les principes des semaines 1 & 2.

1 Édition simplifiée d'une facture

L'objectif de ces exercices est de réaliser l'édition d'une facture dans une version relativement simplifiée puisqu'il s'agit de pouvoir ajouter et/ou supprimer des lignes sur une facture et de l'éditer à l'écran.

Nous ne traiterons pas l'interface homme/machine (IHM) et nous nous contenterons donc de développer les types et sous-programmes qui constituent ce modèle.

Exercice 1 : Notion de produit

Commençons par définir le type Produit et les opérations associées.

1.1 Écrire le type Produit sachant qu'un produit est caractérisé par un code (sur 3 caractères), un libellé (9 caractères) et un prix hors taxe.

Solution : Le type Produit est un enregistrement composé de trois champs un code de type chaîne de 3 caractères, un libellé (chaîne de 9 caractères) et un prix hors taxe (**Réel**).

```
1 INTEGER,PARAMETER,PUBLIC::LG_CODE =  3        ! longueur maximale d'un code
2 INTEGER,PARAMETER,PUBLIC::LG_LIBELLE=9        ! longueur maximale d'un libelle
3 TYPE,PUBLIC::Produit
4     CHARACTER(LEN=LG_CODE)::code
5     CHARACTER(LEN=LG_LIBELLE)::libelle
6     REAL:: prix_ht
7 END TYPE Produit
```

1.2 Écrire un sous-programme pour initialiser un produit à partir de son code, son libellé et son prix hors taxe.

Solution : Le but du sous-programme est « initialiser un produit à partir de son code, son libellé, et son prix hors taxe ».

Les paramètres sont donc :

- le produit à initialiser : **out** ;
- le code du produit : **in** ;
- le libellé du produit : **in** ;
- le prix du produit : **in**.

On constate qu'il n'y a qu'un seul paramètre en sortie et tous les autres sont en entrée. Nous en faisons quand même une procédure car il s'agit d'un sous-programme d'initialisation. Il est préférable dans ce cas que la mémoire soit fournie par le programme appelant (ceci évite par exemple de faire des copies inutiles lors du retour de la fonction).

On peut donc en déduire la spécification.

```
Procédure initialiser_produit(p: out Produit) ;
            c: in Code ; l: in Libellé ; prix_ht: in Réel) Est
    -- Initialiser le produit p à partir de son code c, de son libellé l
    -- et de son prix hors taxe.
```

On peut maintenant en déduire le code.

```
    Début
        produit.code <- c
        produit.libellé <- l
        produit.prix_ht <- prix_ht
    Fin
```

```
SUBROUTINE initialiser_produit(p,c,l, prix_ht)
    TYPE(Produit),INTENT(OUT)::p
    CHARACTER(LEN=LG_CODE),INTENT(IN)::c
    CHARACTER(LEN=LG_LIBELLE),INTENT(IN)::l
    REAL,INTENT(IN):: prix_ht
    p%code=c
    p%libelle=l
    p%prix_ht = prix_ht
END SUBROUTINE  initialiser_produit
```

Exercice 2 : Notion de ligne de facture

Le type LigneFacture permet de représenter une ligne d'une facture. Il est caractérisé par le produit concerné, la quantité commandée et le prix total hors taxe de la ligne.

2.1 Définir le type LigneFacture.

Solution : Le type LigneFacture est également un enregistrement. Notons la redondance entre le prix hors taxe de la ligne et des informations prix hors taxe d'un produit et quantité commandée.

```
TYPE,PUBLIC:: LigneFacture
    TYPE(Produit)::produit
    INTEGER:: quantite        ! > 0
    REAL:: prix_ht        ! prix_ht = quantite * produit.prix_ht
END TYPE LigneFacture
```

2.2 Définir une opération qui permet d'initialiser une ligne de facture à partir du produit concerné et de la quantité commandée. Le prix total hors taxe de la ligne peut être déduit de ces deux informations puisqu'il correspond au prix hors taxe du produit multiplié par la quantité commandée.

Solution : L'initialisation d'une ligne de facture est réalisée sur le même principe que le produit.

```
! Initialiser une ligne de facture a partir du produit concerne et de
! la quantite commandee.
SUBROUTINE initialiser_ligne(ligne,p,quantite)
    TYPE(Produit),INTENT(IN)::p
    INTEGER,INTENT(IN):: quantite
    TYPE(LigneFacture),INTENT(OUT)::ligne
    ligne%produit = p
    ligne%quantite = quantite
    ligne%prix_ht = p%prix_ht * quantite
END SUBROUTINE initialiser_ligne
```

2.3 Écrire une opération qui affiche une ligne de facture. Les informations sont affichées de manière tabulée avec dans l'ordre le code du produit, son libellé, son prix hors taxe, la quantité commandé, le prix total hors taxe et le prix total TTC. On supposera le taux de TVA constant et indépendant des produits.

Voici un exemple de ligne affichée.

```
| 001 |       Truc |   99.00 | 10 |  990.00 | 1184.04 |
```

Solution : Ce sous-programme réalise un affichage sur l'écran. Il convient donc d'en faire une procédure. Notons qu'il n'a qu'un seul paramètre en entrée, la ligne de facture, ce qui nous aurait aussi conduit à en faire une procédure.

```
! Afficher une ligne de facture.
SUBROUTINE afficher_ligne(ligne)
    TYPE(LigneFacture),INTENT(IN)::ligne
    REAL:: prix_ttc

    ! calculer le prix ttc
    prix_ttc = ligne%prix_ht * (1+TVA)

!    PRINT*,"| %3s | %10s | %7.2f | %2d | %7.2f | %7.2f |\n",&
    PRINT"(A5,A19,F9.2,I5,f9.2,f9.2)",&
          ligne%produit%code, ligne%produit%libelle, ligne%produit%prix_ht,&
           ligne%quantite, ligne%prix_ht, prix_ttc
END SUBROUTINE afficher_ligne
```

Exercice 3 : Notion de facture

Définissons enfin le type Facture. Une facture contient plusieurs lignes (20 au maximum). Elle est caractérisée par un numéro de facture.

3.1 Définir le type Facture.

Solution : Le type Facture est un tableau de lignes de facture. On ne sait pas à priori combien il y aura de lignes mais on sait qu'il y en a moins de 20. On peut donc prendre un tableau de

capacité 20 et il faudra gérer la taille effective. Ces deux informations sont donc regroupées dans un enregistrement

```
INTEGER,PARAMETER,PUBLIC:: NB_LIGNES = 20     ! nombre maximal de lignes sur une factu
TYPE,PUBLIC:: Facture
    INTEGER:: numero
    TYPE(LigneFacture),DIMENSION(NB_LIGNES):: lignes
    INTEGER:: nb_lignes_eff        ! nombre effectif de lignes sur la facture
END TYPE Facture
```

3.2 Écrire une opération qui permet d'initialiser une facture à partir de son numéro. Une telle facture est bien entendu vierge et ne contient donc aucune ligne.
Solution : Ce sous-programme a un seul paramètre, la facture à initialiser. C'est un paramètre en sortie. On fait donc une procédure.

```
! Initialiser une facture.
!
! Assure
!        une_facture%nb_lignes == 0
SUBROUTINE initialiser_facture(une_facture,son_numero)
    INTEGER,INTENT(IN):: son_numero
    TYPE(Facture),INTENT(OUT)::une_facture
    une_facture%numero = son_numero
    une_facture%nb_lignes_eff = 0

!    assert(une_facture->nb_lignes == 0)
END SUBROUTINE initialiser_facture
```

3.3 Écrire un sous-programme qui permet de calculer le montant total hors taxe de la facture. Il s'agit de faire la somme des totaux hors taxe des lignes de la facture.
Solution : Ce sous-programme prend en paramètre une facture (entrée) et fournit le total hors taxe de la facture (sortie). Il y a donc un seul paramère en sortie, d'autres qui sont tous en entrée. On en fait donc une fonction.

```
Fonction montant_total_ht_facture(une_facture: Facture): Réel Est
    -- Le montant total hors taxe de la facture une_facture.
```

```
! Determiner le montant total HT d'une facture
FUNCTION montant_total_ht_facture(une_facture) RESULT(resultat)
    REAL::resultat
    TYPE(Facture),INTENT(IN)::une_facture
    INTEGER:: l        ! parcourir les lignes de la facture

    resultat = 0
    DO l = 1,une_facture%nb_lignes_eff
        resultat = resultat+une_facture%lignes(l)%prix_ht
    ENDDO
END FUNCTION montant_total_ht_facture
```

Notons qu'en F, on a défini une variable locale `resultat` qui sert à accumuler le total des lignes.

3.4 Écrire un sous-programme pour ajouter une ligne à une facture, c'est-à-dire un produit et la quantité commandée.

Solution : Ce sous-programme prend en paramètre la facture à modifier (**in out**), le produit à ajouter (**in**) et la quantité commandée (**in**). On fait donc une procédure.

Notons que pour l'implantation de cette procédure, on peut utiliser l'opération `initialiser_ligne` déjà définie.

```
! Ajouter une ligne a une facture.
! Necessite
!        une_facture.nb_lignes < NB_LIGNES
SUBROUTINE ajouter_ligne(une_facture, p, quantite)
    TYPE(Facture),INTENT(OUT)::une_facture
    TYPE(Produit),INTENT(IN)::p
    INTEGER,INTENT(IN):: quantite
!    assert(une_facture->nb_lignes < NB_LIGNES)
    une_facture%nb_lignes_eff=une_facture%nb_lignes_eff+1
    CALL initialiser_ligne(une_facture%lignes(une_facture%nb_lignes_eff),
            p, quantite)
END SUBROUTINE ajouter_ligne
```

3.5 Écrire un sous-programme qui permet d'éditer une facture. Il s'agit d'afficher un entête portant le numéro de la facture. Ensuite sont affichées toutes les lignes et enfin le total hors taxe et TTC de la facture.

Voici un exemple d'édition d'une facture.

```

Édition de la facture 1
-------------------------------------------------------
| 001 |       Truc |   99.00 | 10 |  990.00 | 1184.04 |
| 020 |      Chose |   15.00 |  2 |   30.00 |   35.88 |
| 163 |     Bidule |  250.00 |  1 |  250.00 |  299.00 |
-------------------------------------------------------
                                  | 1270.00 | 1518.92 |
```

Solution : Ce sous-programme prend un seul paramètre, la facture à éditer (**in**). Son objectif est de faire un affichage sur l'écran (une sortie sur l'écran). Nous en faisons donc une procédure.

```
! Éditer a l'ecran la facture.
SUBROUTINE editer_facture(une_facture)
    TYPE(Facture),INTENT(IN)::une_facture
    INTEGER:: l        ! parcourir les lignes de la facture
    REAL:: total_ht       ! total ht de la facture
    REAL:: total_ttc       ! total ttc de la facture

    ! Calculer les totaux
    total_ht = montant_total_ht_facture(une_facture)
    total_ttc = total_ht * (1+TVA)

    ! Afficher l'entete
    PRINT*
    PRINT*
    PRINT*,"Edition de la facture : ", une_facture%numero
```

```
16      PRINT*,"-------------------------------------------------------"
17
18      ! Afficher les lignes
19      DO l = 1,une_facture%nb_lignes_eff
20          CALL afficher_ligne(une_facture%lignes(l))
21
22      ENDDO
23
24      ! Afficher le total
25      PRINT*,"-------------------------------------------------------"
26  !    PRINT*,"%33s | %7.2f | %7.2f |\n", "", total_ht, total_ttc
27      PRINT"(T39,f9.2,f9.2)", total_ht, total_ttc
28  END SUBROUTINE editer_facture
```

3.6 Écrire un sous-programme pour supprimer une ligne d'une facture en fonction de son numéro. Bien sûr, la première ligne porte le numéro 1.

Solution : Ce sous-programme a pour paramètres la facture à modifier (**in out**) et le numéro de la ligne à supprimer (**in**). On fait donc une procédure.

```
1   ! Supprimer une ligne de la facture.  Le numero n_ligne correspond a
2   ! la ligne de la facture qu'il faut supprimer.
3   !
4   ! Necessite
5   !        1 <= n_ligne && n_ligne <= une_facture->nb_lignes
6   SUBROUTINE supprimer_ligne(une_facture, n_ligne)
7       TYPE(Facture),INTENT(OUT)::une_facture
8       INTEGER,INTENT(IN):: n_ligne
9       INTEGER:: l         ! parcourir les lignes a decaler
10
11  !    assert(1 <= n_ligne && n_ligne <= une_facture->nb_lignes)
12
13      ! Decaler les lignes
14      DO l = n_ligne,une_facture%nb_lignes_eff
15          une_facture%lignes(l) = une_facture%lignes(l+1)
16      ENDDO
17      une_facture%nb_lignes_eff=une_facture%nb_lignes_eff-1
18  END SUBROUTINE supprimer_ligne
```

Exercice 4 : Programme de test

Écrire un programme de test pour les types et opérations définis dans les exercices précédents.

Remarque : Même si c'est le dernier exercice, il est souhaitable (et nécessaire) de le faire en même temps que les autres, au fur et à mesure que les sous-programmes sont écrits. En effet, tester au fur et à mesure permet de tester de manière plus exhaustive (donc de détecter plus d'erreurs), de localiser plus vite la cause de l'erreur et donc de la corriger plus facilement.

Solution :

```
1   ! Programme principal, en fait programme de test !
2   ! Attention : ce programme de test n'est absolument pas complet !
3   PROGRAM main
4       USE SP_MOD
5       TYPE(Facture) ::f1
```

```
6       TYPE(Produit):: p1, p2, p3
7
8       CALL initialiser_produit(p1, "001", "Truc     ", 99.0)
9       CALL initialiser_produit(p2, "020", "Chose    ", 15.0)
10      CALL initialiser_produit(p3, "163", "Bidule   ", 250.0)
11
12      ! Creation de la facture en ajoutant les lignes
13      CALL initialiser_facture(f1, 1)
14      CALL editer_facture(f1)
15      CALL ajouter_ligne(f1, p1, 10)
16      CALL editer_facture(f1)
17      CALL ajouter_ligne(f1, p2, 2)
18      CALL editer_facture(f1)
19      CALL ajouter_ligne(f1, p3, 1)
20      CALL editer_facture(f1)
21
22      ! Suppression de la deuxieme ligne
23      CALL supprimer_ligne(f1, 2)
24      CALL editer_facture(f1)
25
26  END PROGRAM main
```