Sous-programmes : fonctions et procédures

Rappels du semestre 1

L'indispensable ... en une diapo

Connaître et bien distinguer, comprendre le sens des mots

LA définition	vs.	LES appels
l'en-tête, la signature	vs.	le corps
les paramètres formels	vs.	les paramètres effectifs
les variables locales	vs.	les variables globales
fonction	vs.	procédure
le mode logique des paramètres		IN vs. OUT vs. INOUT
le mode physique/de passage de paramètre : par valeur	vs.	par adresse (ou référence)

Pour les variables, être sensible aux notions de :

portée, visibilité, localité, effet de bord

Fonction ou procédure

Fonction

une fonction produit une valeur
cette valeur est ensuite affectée dans une variable du contexte appelant
entre le début de l'appel et cette affectation (exclus) , l'état du contexte appelant ne doit / devrait pas être modifié (pas d'effet de bord)

Procédure

une procédure ne retourne rien
une procédure réalise un traitement qui modifie l'état du système
l'état du système est modifié par l'intermédiaire des arguments de la procédure

En-tête, appels, fonctions et procédures, paramètres formels, effectifs, leurs modes

Les en-têtes ...

```
fonction deuxfois(x : in float) retourne float
fonction PGCD(i : in entier , j : in entier) retourne entier
procedure permuter(c1 : inout caractere , c2 : inout caractere)
```

suffisent pour les appels :

```
eff_2015 = deuxfois(eff_2014)
val = PGCD(12, min(5,3))
permuter(k, MonNom[0]) // A noter : pas d'affectation
```

En-tête, corps, paramètre formel, leur mode variable locale

```
fonction deuxfois(x : in float) retourne float
debut
        retourne 2.0 * x
fin fonction
```

```
fonction permuter(c1 : inout caractère, c2 : inout caractère)
        c : caractère   // c est une variable locale de type caractère
debut
        c = c1
        c1 = c2
        c2 = c
fin fonction
```


Conseil : ne pas mélanger traitements et E/S

On veut lire deux caractères au clavier (entrée), les permuter (traitements) et les afficher à l'écran (sortie).

Moche : 1. définir

```
fonction moche( c1 : inout caractere,
                c2 : inout caractere)
    c : caractere
debut
    lire(c1); lire(c2)
    c = c1; c1= c2; c2 = c
    afficher(c1,c2)
fin fonction
```

2. et appeler :

```
declare
    deb, ffin : caractere
debut
    moche(deb, ffin)
fin
```

Pas moche : 1. utiliser

```
fonction permuter(c1 : inout caractere,
                  c2 : inout caractere)
```

2. comme ça :

```
declare
    deb, ffin : caractere
debut
    lire(deb)
    lire(ffin)
    permuter(deb, ffin)
    afficher(deb,ffin)
fin
```


Paramètres formels ou arguments formels

fonction vs. procédure

Fonction

- tous les paramètres d'entrée sont de mode in, ce mode peut être implicite
- le paramètre est vue comme une constante dans le corps du sous-prog
- une fonction retourne une valeur dont le type est défini dans l'en-tête
- le corps contient au moins un retourne
- une variable locale est (souvent) définie dans le corps pour stocker la valeur qui sera retournée

Procédure

- les paramètres formels sont de modes in, inout ou out
- aucun retourne dans le corps

Exercices

Ecrire de façon synthétique un algorithme qui produit l'affichage :

```
Perpignan est la préfecture du département Pyrénées-Orientales
Carcassonne est la préfecture du département Aude
Foix est la prefecture du département Ariege
Montpellier est la prefecture du departement Herault
Le departement Herault a Montpellier pour prefecture
Le departement Ariege a Foix pour prefecture
Le departement Aude a Carcassonne pour prefecture
Le departement Pyrénées-Orientales a Perpignan pour prefecture
```

Exercices, suite

Ecrire la fonction min(a,b) de la diapo 10

S'en servir pour écrire la fonction min(x, y, z)

Ecrire une fonction qui retourne la valeur minimale d'un tableau de n entiers

Ecrire une fonction F qui, après le traitement suivant :

```
suite1 = [a, b, c] ;          suite2 = [a, b, c, d, e, f]
modifiee1 = F(suite1);        modifiee2 = F(suite2)
afficher(modifiee1);          afficher(modifiee2)
```

affiche :

```
[b, c, a]     [b, c, d, e, f, a]
```


Exercices, suite

Ecrire une fonction et un programme qui l'appelle pour résoudre chacun des problèmes suivants.

Préalable : analyser le problème et identifier les traitements, les E/S, les fonctions, les paramètres, leurs modes, les cas exceptionnels ...

- je calcule la pente d'une droite définie par deux points dans le plan
- je calcule l'ordonnée d'un point situé sur une droite définie par sa pente et son ordonnée à l'origine
- je calcule l'ordonnée à l'origine d'une droite définie par deux points
- je vérifie si un point appartient ou non à une droite définie par deux points
- je calcule l'intersection de deux droites définies par deux paires de points en veillant à répondre dans tous les cas

Compléments

Portée et visibilité des variables entre programme appelant et sous-programme appelé

Portée, visibilité, localité, effet de bord

Quand puis-je utiliser une variable ?

Portée : zone où une variable est accessible, où elle "existe"

- commence à sa déclaration
- termine à la fin du contexte qui englobe sa déclaration
- contexte : algo principal vs. le corps d'une fonction
  (éventuellement selon les langages : une structure de contrôle)

Comment puis-je utiliser cette variable ?

Visibilité : zone où une variable est accessible par son identificateur

- inclus dans la zone de portée
- cesse si un même identificateur valide cache son identificateur
  - si l'identificateur est caché, le langage définit des solutions pour accéder au "bon identificateur" , par exemple notation préfixée par le contexte

Portée, visibilité, localité, effet de bord

Quid des variables locales des sous-programme ?

Localité des variables locales d'un sous-programme

- leur portée est limitée au corps de la fonction
- elle n'existe pas à l'extérieur du corps
- leur identifiant cache un identifiant extérieur identique :
  - la variable locale x de la fonction F cache la variable x de l'appelant
  - mais la variable locale x de la fonction F cesse d'exister au retour vers l'appelant

Pas d'effet de bord avec les fonctions

- les fonction sont des blocs étanches et autonomes
- seuls les paramètres permettent d'échanger avec l'extérieur *
- si le langage ne l'oblige pas, S'INTERDIRE d'accéder à une variable globale

Deux conseils :

- une bonne variable globale est une constante
- limiter les effets à l'extérieur de la fonction aux seuls paramètres des fonctions *

Bien distinguer **tmp**, **x** et tmp, x

Les variables locales n'existent que dans la zone bleue

```
declare
        i, j, tmp : entier
        nb, x : float
debut
        i = 1; j = 2; x = 4.0
        tmp = j
        permuter(i,j)
        nb = float(tmp)
        si (x==double(nb)) alors
                nb = x
        sinon
                nb = x - 1.0
        fin si
        afficher(i, j, nb)
fin
```

```
fonction permuter(i:entier, j: entier)
        tmp : entier
debut
        tmp = i
        i = j
        j = tmp
fin fonction

fonction double(float:x) retourne float
debut
        retourne 2.0*x
fin fonction
```


Aspects plus avancés

Le passage des paramètres

Passage de paramètres ?

Le corps de la fonction ne connait que les paramètres formels.
L'appel de la fonction précise les paramètres effectifs.
Comment le paramètre effectif devient-il connu par la fonction appelée ?
Qu'est-ce qui est connu de l'appelé ?

- *la valeur du paramètre effectif ?*
- *la variable du paramètre effectif ?*

Se sont les questions du passage de paramètres appelant-appelé

Passage des paramètres par valeur

Passage par valeur

la valeur du paramètre effectif est copiée dans le contexte de la fonction

Les paramètres sont passés par valeur quand

mode IN : le paramètre effectif n'est pas modifié dans la fonction

mode OUT : la valeur du paramètre de retour est attendu par l'appelant
le paramètre formel est une variable non intialisée : elle attend absoluement une affectation dans le corps de la fonction

Exemple : la fonction `doubler(float:x) retourne float`

On supposera ce mode de passage pour les types scalaires :
entier, float, caractères, booléens

Attention : la fonction peut avoir aucun effet ... était-ce voulu ?

Passage des paramètres par adresse

Les paramètres sont passés par adresse, par référence

l'adresse de la variable du contexte appelant est passée comme paramètre effectif

le sous-programme lit et écrit à l'adresse de la variable globale

Quand ?

Nécessaire pour les paramètres de mode inout

On supposera ce mode de passage pour les types composés (non scalaires) :
les tableaux, les chaînes de caractères, les enregistrements

Pourquoi ?
la re-copie "coûte plus cher" en espace mémoire et en temps

Exemple : `fonction min(t : tableau entiers, n : entier) retourne entier`

Passage des paramètres, compléments

Varie selon les langages de programmation

C : implicitement par valeur, explicitement par référence (&)

python : scalaires par copie, composés par référence (même l'affectation d'objet composés !)

ada : caché à l'utilisateur, le compilateur choisit selon le mode IN, INOUT, et OUT

java : scalaires par valeurs, composés par références

Commentaires

en pratique on trouve de tout : c'est la jungle !

aspect très important :
mais qu'est véritablement en train de modifier ma fonction ???

Synthèse de la séance

Ce qu'il faut retenir !

Sous-programmes

- définition vs. appels
- en-tête vs. corps
- paramètres formels vs. effectifs
- fonction vs. procedure
- portée vs. visibilité
- variables locales vs. globales
- modes logiques des paramètres IN vs. OUT vs. INOUT
- modes physique/de passage de paramètre : par valeur vs. par adresse (ou référence)
- séparer les traitements et les entrées-sorties
- limiter les effets de bords au strict minimum (= 0 :)
- les constantes peuvent être des variables globales
- un algorithme est écrit une fois, il est lu des centaines de fois.
  Morale ?