Les fonctions et procédures

- Rappels
- Règles de nommage
- Les différents types de variable
- Les paramètres
  - Les passage de paramètres
    - Le passage de paramètres en entrée
    - Le passage de paramètres en sortie
    - Le passage de paramètres en entrée/sortie
- Les fonctions
  - Exemple de déclaration de fonction
  - Exemple de déclaration de fonction dans un algorithme complet
- Les procédures
- Mise en pratique
  - Exercice 0
  - Exercice 1
- Fonctions/procédures récursives
  - Récursivité : Arité/bien fondé
  - Récursivité : Ordre de récursion
  - Mise en pratique
    - Exercice 0
    - Exercice 1

Rappels

La méthodologie de base de l'informatique est :

1. Abstraire : Retarder le plus longtemps possible l'instant du codage.
2. Décomposer : *"... diviser chacune des difficultés que j'examinerai en autant de parties qu'il se pourrait et qu'il serait requis pour les mieux résoudre."* [Descartes].
3. Combiner : Résoudre le problème par combinaison d'abstractions.

Par exemple, résoudre le problème suivant :

> *Écrire un programme qui affiche en ordre croissant les notes d'une promotion suivies de la note la plus faible, de la note la plus élevée et de la moyenne*

revient à résoudre les problèmes suivants :

- Remplir un tableau de naturels avec des notes saisies par l'utilisateur.
- Afficher un tableau de naturels.
- Trier un tableau de naturel en ordre croissant.
- Trouver le plus petit naturel d'un tableau.
- Trouver le plus grand naturel d'un tableau.
- Calculer la moyenne d'un tableau de naturels.

Chacun de ces sous-problèmes devient un nouveau problème à résoudre. Si on considère que l'on sait résoudre ces sous-problèmes, alors on sait "quasiment" résoudre le problème initial.
Donc écrire un programme algorithmique qui résout un problème revient toujours à écrire des sous-programmes qui résolvent des sous parties du problème initial.

En algorithmique il existe deux types de sous-programmes :

- les fonctions,
- les procédures.

Un sous-programme est obligatoirement caractérisé par un nom (un identifiant) unique.
Lorsqu'un sous-programme a été explicité (on a donné l'algorithme), son nom devient une nouvelle instruction, qui peut être utilisé dans d'autres (sous-)programmes.
Le (sous-)programme qui utilise un sous-programme est appelé (sous-)programme appelant.

Règles de nommage

Nous savons maintenant que les variables, les constantes, les types définis par l'utilisateur et que les sous-programmes possèdent un nom.
Ces noms doivent suivre certaines règles :

- ils doivent être explicites (à part quelques cas particuliers, comme par exemple les variables i et j pour les boucles).
- ils ne peuvent contenir que des lettres et des chiffres et le symbole "-".
- ils commencent obligatoirement par une lettre.
- lorsqu'ils sont composés de plusieurs mots, on utilise les majuscules pour séparer les mots (par exemple jourDeLaSemaine)

Les différents types de variable

La portée d'une variable est l'ensemble des sous-programmes où cette variable est connue (les instructions de ces sous-programmes peuvent utiliser cette variable).

Une variable définie au niveau du programme principal (celui qui résout le problème initial, le problème de plus haut niveau) est appelée variable globale.
Sa portée est totale : tout sous-programme du programme principal peut utiliser cette variable.

Une variable définie au sein d'un sous programme est appelée variable locale.
La portée d'un variable locale est uniquement le sous-programme qui la déclare.

Lorsque le nom d'une variable locale est identique à une variable globale, la variable globale est localement masquée. Dans ce sous-programme la variable globale devient inaccessible.

Les paramètres

Un paramètre d'un sous-programme est une variable locale particulière qui est associée à une variable ou constante (numérique ou définie par le programmeur) du (sous-)programme appelant.
Puisque qu'un paramètre est une variable locale, un paramètre admet un type.
Lorsque le (sous-)programme appelant appelle le sous-programme il doit indiquer la variable (ou la constante), de même type, qui est associée au paramètre.

Par exemple, si le sous-programme **sqr** permet de calculer la racine carrée d'un réel :

- ce sous-programme admet un seul paramètre de type réel positif
- le (sous-)programme qui utilise **sqr** doit donner le réel positif dont il veut calculer la racine carrée, cela peut être :
    - une variable, par exemple *a*
    - une constante, par exemple *5.25*

Les passage de paramètres

Il existe trois types d'association (que l'on nomme passage de paramètre) entre le paramètre et la variable (ou la constante) du (sous-)programme appelant :

- le passage de paramètre en entrée,
- le passage de paramètre en sortie,
- le passage de paramètre en entrée/sortie.

Le passage de paramètres en entrée

Les instructions du sous-programme ne peuvent pas modifier l'entité (variable ou constante) du (sous-)programme appelant. En fait c'est la valeur de l'entité du (sous-)programme appelant qui est copiée dans le paramètre (à part cette copie il n'y a pas de relation entre le paramètre et l'entité du (sous-)programme appelant).
C'est le seul passage de paramètre qui admet l'utilisation d'une constante.

Par exemple :

- le sous-programme **sqr** permettant de calculer la racine carrée d'un nombre admet un paramètre en entrée.
- le sous-programme **saisir** qui permet d'afficher des informations admet n paramètres en entrée.

Le passage de paramètres en sortie

Les instructions du sous-programme affectent obligatoirement une valeur à ce paramètre (valeur qui est donc aussi affectée à la variable associée du (sous-)programme appelant).
Il y a donc une liaison forte entre le paramètre et l'entité du (sous-)programme appelant.
C'est pour cela qu'on ne peut pas utiliser de constante pour ce type de paramètre.
La valeur que pouvait posséder la variable associée du (sous-)programme appelant n'est pas utilisée par le sous-programme.

Par exemple :

- le sous-programme **afficher** qui permet de mettre dans des variables des valeurs saisies par l'utilisateur admet n paramètres en sortie.

Le passage de paramètres en entrée/sortie

Passage de paramètre qui combine les deux précédentes.
A utiliser lorsque le sous-programme doit utiliser et/ou modifier la valeur de la variable du (sous-)programme appelant.
Comme pour le passage de paramètre en sortie, on ne peut pas utiliser de constante.

Les fonctions

Les fonctions sont des sous-programmes admettant des paramètres et retournant un seul résultat ; comme les fonctions mathématiques res - f(x,y,. . .).
Les paramètres sont en nombre fixe (≥ 0).
Une fonction possède un seul type, qui est le type de la valeur retournée.
Le passage de paramètre est uniquement en entrée: c'est pour cela qu'il n'est pas précisé.
Lors de l'appel, on peut donc utiliser comme paramètre des variables, des constantes mais aussi des résultats de fonction.
La valeur de retour est spécifiée par l'instruction **retourner**.
Généralement le nom d'une fonction est soit un nom (par exemple *minimum*), soit une question (par exemple *estVide*).

On déclare une fonction de la façon suivante :

fonction nomDeLaFonction (paramètre(s) de la fonction) : type de la valeur retournée
variables locale1 : type; . . .

début
instructions de la fonction avec au moins une fois l'instruction **retourner**
fin

On utilise une fonction en précisant son nom suivi des paramètres entre parenthèses. Les parenthèses sont toujours présentes même lorsqu'il n'y a pas de paramètre.

Exemple de déclaration de fonction

```
fonction abs(unEntier : entier) : entier
variables      tmp: entier
début
    si unEntier ≥ 0 alors
        tmp ← unEntier
    sinon
        tmp ← -unEntier
    finsi
    retourner tmp
fin
```

Exemple de déclaration de fonction dans un algorithme complet

```
Algorithme valeurAbsolue

variables      a, b : entiers
fonction abs(unEntier : entier) : entier

    variables      tmp: entier
    début
        si unEntier ≥ 0 alors
            tmp ← unEntier
```

```
            sinon
                tmp ← -unEntier
            finsi
            retourner tmp
        fin

début
        afficher(“Saisissez un entier :”)
        saisir(a)

        b ← abs(a)
        afficher(“La valeur absolue de ”, a, “ est “, b)
fin
```

Lors de l’exécution de la fonction abs, la variable a et le paramètre unEntier sont associés par un passage de paramètre en entrée : la valeur de a est copiée dans unEntier.

Les procédures

Les procédures sont des sous-programmes qui ne retournent aucun résultat.
Par contre, elles admettent des paramètres avec des passages :

- en entrée, préfixés par Entrée (ou E)
- en sortie, préfixés par Sortie (ou S)
- en entrée/sortie, préfixés par Entrée/Sortie (ou E/S)

Généralement le nom d'une procédure est un verbe.

On déclare une procédure de la façon suivante :

procédure nomDeLaProcedure(E paramètre(s) en entrée; S paramètre(s) en sortie; E/S paramètre(s) en entrée/sortie)
variable(s) locale(s)
début
    instructions de la procédure
fin

On appelle une procédure comme une fonction, en indiquant son nom suivi des paramètres entre parenthèses.

Mise en pratique

Exercice 0

Écrire un algorithme qui échange la valeur de deux variables en s'appuyant sur une fonction ou une procédure.
Exemple, si a ← 2 et b ← 5, le programme donnera a ← 5 et b ← 2.

Exercice 1

Écrire le sous-algorithme de la fonction "moyenne" qui renvoie la moyenne de deux entiers.
Écrire l'algorithme qui contient la déclaration de la fonction moyenne et des instructions qui appellent cette fonction.

Fonctions/procédures récursives

Une fonction ou une procédure récursive est une fonction qui s'appelle elle même.
Exemple :

```
fonction factorielle(n: naturel) : naturel
début
    si n = 0 alors
        retourner 1
    sinon
        retourner n*factorielle(n-1)
    finsi

fin
```

On peut caractériser un algorithme récursif par plusieurs propriétés:

- le mode d'appel : direct/indirect.
- le nombre de paramètres sur lesquels porte la récursion : arité.
- le nombre d'appels récursifs : ordre de récursion.
- le genre de retour : terminal/non terminal.

Une fonction récursive s'appellant elle-même a un mode d'appel direct (ex: factorielle).
Si la récursivité est effectuée à travers plusieurs appels de fonctions différentes le mode d'appel est indirect.

Récursivité : Arité/bien fondé

L'arité d'un algorithme est le nombre de paramètres d'entrée.
Récursivité bien fondé : une récursivité dans laquelle les paramètres de la fonction appelée sont «plus simple» que ceux de la fonction appelante.
Par exemple factorielle(n) appelle factorielle(n-1).

Récursivité : Ordre de récursion

L'ordre de récursion d'une fonction est le nombre d'appels récursifs lancés à chaque appel de fonction.
Par exemple, factorielle(n) ne nécessite qu'un seul appel à la fonction factorielle avec comme paramètre n − 1. C'est donc une fonction récursive d'ordre 1.
Par exemple, la fonction suivante basée sur la formule $C^{p}_{n} = C^{p}_{n-1} + C^{p-1}_{n-1}$, est d'ordre 2.

```
fonction comb(p, n : entiers) : entier
début
    si p = 0 ou n = p alors
        retourner 1
    sinon
        retourner comb(p, n-1) + comb(p-1, n-1)
    finsi
fin
```

Mise en pratique

Exercice 0

En utilisant une fonction récursive, écrire un algorithme qui détermine le terme U_n de la suite de Fibonacci définit comme suit :
$U_0 = 0$
$U_1 = 1$
$U_n = U_n\text{-}1 + U_n\text{-}2, n >= 2$

Exercice 1

En utilisant une fonction récursive, écrire un algorithme qui écrit la structure d'un tableau HTML (<table><tr><td></td></tr></table>) en permettant l'écriture de plusieurs lignes et plusieurs cellules si l'utilisateur indique qu'il souhaite poursuivre chaque étape.