Chapitre 8

Procédures

1. Introduction

Comme les fonctions, les procédures sont des blocs d'instructions référencés par un nom et qui répondent à une spécification précise. Cependant elles ont des rôles différents :

- Une **fonction** renvoie une valeur qu'elle a calculé à partir des valeurs reçues en paramètres. L'appel d'une fonction se fait dans le même contexte qu'une **expression** au sein d'une instruction.
- Une **procédure** exécute des instructions qui utilisent des valeurs reçues en paramètres, mais qui peuvent aussi modifier l'environnement mémoire de l'unité qui l'appelle. L'appel d'une procédure se fait dans le même contexte qu'une **instruction**.
- Fonction et procédures sont désignées par le nom générique de **sous-programmes**.

2. Exemples

2.1. Procédure avec paramètres d'entrée

Supposons que l'on veuille afficher une suite de lignes comme le montre le schéma suivant :

```
*
++++
###
ooooooo
```

Au lieu de programmer chaque ligne l'une après l 'autre, on peut remarquer que chaque ligne est définie de manière non ambiguë dès que l'on connaît le caractère qui la compose et le nombre de caractères qu'elle comporte. En tenant compte de cette remarque, on peut définir une procédure dont le travail est d'afficher une ligne connaissant le caractère à afficher, ainsi que le nombre de caractères.

```
Algorithme dessin

Début
  ligne(1, '*')
  ligne(4, '+')
  ligne(3, '#')
  ligne(7, 'o')
Fin
```

```
procédure ligne
      (Entrée nb  : entier
       Entrée car : caractère)
Variables locales
  i : entier
Début
  Pour i de 1 à nb faire
    écrire(car)
  Finpour
  écrire(CRLF)    /* passage à la ligne */
Fin
```

2.2. Procédure avec paramètres d'entrée et paramètres de sortie

Cet algorithme demande deux angles exprimés en degrés et minutes d'angle, puis calcule la différence entre les deux. Avant de faire la différence, les deux angles sont convertis en minutes par la fonction `minute`. La fonction valAbs renvoie la valeur absolue du paramètre. Pour faire la conversion en sens inverse, on ne peut pas utiliser une fonction puisqu'il y a deux résultats (les degrés et les minutes). C'est pourquoi on utilise la procédure `convertir` qui a deux paramètres de sortie.

```
Algorithme rotation
Variables
  d1, m1, d2, m2  : entier
  mm1, mm2, dm    : entier
  deg, min : entier
Début
  lire(d1, m1)
  lire(d2, m2)
  mm1 ← minutes(d1, m1)
  mm2 ← minutes(d2, m2)
  dm ← valAbs(mm1 - mm2)
  convertir(dm, deg, min)
  écrire(deg, min)
Fin
```

```
fonction minutes(d, m : entier) : entier
Début
  retourner(60 * d + m)
Fin
```

```
fonction valAbs(x : entier) : entier
Variables locales
  abs : entier
Début
  si x > 0 alors
    abs ← x
  sinon
    abs ← -x
  finsi
  retourner(abs)
Fin
```

```
procédure convertir(Entree mi : entier
                    Sortie dd : entier
                    Sortie mm : entier)
Début
  dd ← mi div 60
  mm ← mi mod 60
Fin
```

2.3. Procédure avec paramètres d'entrée / sortie

Pour classer par ordre croissant trois lettres rangées dans trois variables c1, c2, et c3, on procède comme suit :

Variables :	**c1**	**c2**	**c3**
on compare le contenu de c1 et de c2 : dans l'ordre	T	Z	A
on compare et on échange le contenu de c2 et de c3	**T**	**Z**	A
on compare et on échange le contenu de c1 et de c2	T	**A**	**Z**
	A	**T**	Z

Les procédures classer et échanger sont susceptibles de modifier les valeurs des varaibles passées en paramètres (paramètres d'entrée/sortie).

```
Algorithme tri_lettres

Début
  lire(c1, c2, c3)
  classer(c1, c2)
  classer(c2, c3)
  classer(c1, c2)
  écrire(c1, c2, c3)
Fin
```

```
procédure classer(Entrée/Sortie a, b : caractère)
Début
  si a > b alors
    échanger(a, b)
  finsi
Fin
```

```
procédure échanger(Entrée/Sortie x, y : caractère)
Variables locales
  aux : caractère
Début
  aux ← x   ;     x ← y ;     y ← aux
Fin
```

3. Vocabulaire et syntaxe

Une procédure est un bloc d'instructions

- qui porte un nom, son *identificateur*
- qui communique avec l'unité appelante par l'intermédiaire des *paramètres*
- qui exécute ses instructions conformément à une spécification et aux conditions de son appel

Comme pour les fonctions il faut distinguer

- la définition de la procédure avec les *paramètres formels*,
- l'appel de la procédure avec les *paramètres effectifs*.

3.1. Définition de la procédure

Une procédure peut être définie n'importe où, à l'extérieur d'un algorithme. La définition de la procédure se compose de :

- son **en-tête** qui comprend :
- son identificateur
- la liste de ses **paramètres formels** et de leur type et de leur statut
- des déclarations locales (constantes ou variables) : les constantes ou variables locales ne sont connues que le temps de l'exécution de la procédure.
- les instructions qui calculent son résultat
- au moins une instruction **retourner** qui renvoie la valeur résultat

```
procédure <ident-procédure>
                (<statut> <ident-paramètre> : <type> <rôle>
                     ...
                <statut> <ident-paramètre> : <type> <rôle>)
/ spécification et conditions d'appels /
Variables locales
 <ident-variable> : <type>    <rôle>
 ....
 <ident-variable> : <type>    <rôle>

Début
      <instructions>
Fin
```

NB : `<role>` correspond à du commentaire explicitant le rôle d'un paramètre, ou d'une variable.

3.2. Appel de la procédure

La procédure est appelée depuis un algorithme principal, ou depuis une autre procédure, là où les instructions qu'elle renferme doivent être exécutées. Elle est appelée par son identificateur suivi de la liste des **paramètres effectifs** placés entre parenthèses..

```
<ident-procédure>(<ident-paramètre>, ..., <ident-paramètre>)
```

4. Paramètres

4.1. Règles générales

- Il doit y avoir une correspondance biunivoque entre les *paramètres effectifs* et les *paramètres formels.* La correspondance est assurée par l'**ordre d'écriture** dans les deux listes.
- Les types des paramètres qui se correspondent doivent être compatibles.

4.2. Trois statuts de paramètres

Paramètres d'entrée

Le paramètre effectif peut prendre la forme de n'importe quelle expression (constante, variable ou expression) dont la valeur est copiée dans le paramètre formel pour être **consultée** ou **utilisée** dans la procédure.

paramètre effectif		**paramètre formel**	**procédure action(info)**
une expression	►	`info`	`écrire (info)` `X ✱ info`

Paramètres de sortie

Le paramètre effectif est nécessairement un identificateur de variable qui reçoit une valeur produite par la procédure par l'intermédiaire du paramètre formel.

paramètre effectif		**paramètre formel**	**procédure action(info)**
un identificateur	◄	`info`	`lire (X)` `info ✱ X`

Paramètres d'entrée-sortie

Le paramètre effectif est un identificateur de variable qui contient une valeur. Celle-ci est envoyée à la procédure via le paramètre formel. La procédure modifie la valeur et la renvoie au paramètre effectif.

paramètre effectif		**paramètre formel**	**procédure action(info)**
un identificateur	► ◄	`info`	`info ✱ f(info)`

5. Intérêt des procédures et règle de bon usage

Les procédures sont les outils mêmes de la programmation modulaire. Elle permettent de fractionner les algorithmes (et par suite les programmes) en unités d'actions autonomes, plus intelligibles qu'un bloc d'instructions perdu au milieu d'un algorithme. Elles permettent aussi de partager le développement entre différentes personnes.

Toutes les informations qui circulent entre l'unité appelante et la procédure appelée doivent le faire par l'intermédiaire des paramètres.