Chapitre 7

Fonctions

1. Appel de fonctions prédéfinies

Les algorithmes font souvent appel à des fonctions prédéfinies :

- **fonctions mathématiques** : `racine`, fonctions trigonométriques, etc.
  exemple : pour calculer l'hypoténuse d'un triangle rectangle

```
lire(a, b)
c ← racine((a * a) + (b * b))
```

- **fonction sur les chaînes de caractères** : `longueur, sous_chaine`
  exemple : pour afficher les trois premiers caractères d'une chaîne

```
lire(chaine)
si nbcar >= 3 alors
  écrire ('Début : ', sous_chaine(chaine, 1, 3))
finsi
```

Dans un langage de programmation, les *fonctions prédéfinies* constituent une véritable boîte à outils, souvent organisée en *bibliothèques de fonctions* qui regroupent les fonctions par domaine (math, chaînes de caractères, système, etc.).

Si de nouvelles fonctions sont nécessaires, il est possible de les définir, c'est à dire de leur donner un identificateur et de programmer les calculs qui conduisent à ce qu'elles doivent renvoyer.

2. Rôle d'une fonction

Une fonction prend des données par l'intermédiaire de ses *paramètres*[1], et renvoie un résultat conforme à sa *spécification*, à condition que les conditions d'appels soient respectées. Par exemple un algorithme ne doit pas appeler une fonction `racine` avec un paramètre de valeur négative, si cela arrive, c'est l'algorithme qui est faux, pas la fonction.

3. Exemple

Les deux algorithmes `Maxquatre` calculent le maximum de quatre nombres entiers.

Dans le premier algorithme, la même instruction conditionnelle est répétée trois fois, aux noms des variables près

Dans le deuxième algorithme, les trois instructions conditionnelles analogues sont remplacées par des appels à une même fonction qui s'applique à des valeurs différentes. Ceci rend plus compréhensible ce que fait l'algorithme en évitant de dupliquer des instructions identiques.

[1] définis plus loin

```
Algorithme Maxquatre_1
/* demande deux nombre à l'utilisateur, calcule et affiche le plus
grand des deux. */

Variables
  a, b, c, d  : entier  /* nombres d'étude */
  mab, mcd    : entier  /* maxima intermédiaires */
  maxi        : entier  /* maximum des 4 nombres */
Début
  lire(a, b, c, d)
  si a > b alors
    mab ← a
  sinon
    mab ← b
  finsi
  si c > d alors
    mab ← c
  sinon
    mab ← d
  finsi
  si mab > mcd alors
    maxi ← mab
  sinon
    maxi ← mcdb
  finsi
  écrire(maxi)
Fin

Algorithme Maxquatre_2
/* idem avec appels de fonctions */
Variables
  idem ci-dessus
Début
  lire(a, b, c, d)
  mab ← maxdeux(a, b)
  mcd ← maxdeux(c, d)
  maxi ← maxdeux(mab, mcd)
  écrire(maxi)
Fin
```

Définition de la fonction maxdeux

```
fonction maxdeux(x, y : entier) : entier
/* renvoie le maximum des deux entiers passés en paramètres) */
Variables locales
  mxy   : entier    /* maximum de x et de y */
Début
  si x > y alors
    mxy ← x
  sinon
    mxy ← y
  finsi
  retourner(mxy)
Fin
```

4. Définition, syntaxe et mécanisme de passage de paramètres

Une fonction est un bloc d'instructions

- qui porte un nom, son *identificateur*
- qui prend des valeurs en entrée par l'intermédiaire de *paramètres*
- qui calcule et renvoie un résultat conforme à sa spécification et aux conditions de son appel

Le *profil* (ou *signature*) d'une fonction est constitué de la liste de types des valeurs entrées et du type de la valeur renvoyée en résultat

liste des types des valeurs d'entrée → type du résultat

exemples :

```
maxdeux : (entier, entier) → entier
sous-chaine : (chaine de caractères, entier, entier) → chaine de caractères
```

4.1. Définition de la fonction

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

- son **en-tête** qui comprend :
- son identificateur
- la liste de ses **paramètres formels** et de leur type
- le type de la valeur qu'elle renvoie
- des déclarations locales (constantes ou variables)
- les instructions qui calculent son résultat
- au moins une instruction **retourner** qui renvoie la valeur résultat

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

Début
 |     <instructions>
 |     retourner <expression>
Fin
```

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

Attention : Il est indispensable de spécifier chaque fonction, et de préciser les conditions dans lesquelles elle doit être appelée. En effet si le programmeur qui intègre une fonction à un de ses programmes ne respecte pas les conditions d'appels, il n'a aucune garantie sur ce que fera la fonction dans son programme.

4.2. Appel de la fonction

La fonction est appelée depuis un algorithme principal, ou depuis une autre fonction (ou procédure[2]), dans l'expression où le résultat qu'elle renvoie doit être utilisé.

[2] Voir plus loin.

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

4.3. Passage de paramètres

Dans la définition de la fonction : ***paramètres formels***

Dans l'appel de la fonction : ***paramètres effectifs*** (ou réel[3])

Il doit y avoir une correspondance **biunivoque** entre les *paramètres formels* et les *paramètres effectifs*, la correspondance étant régie par l'**ordre d'écriture**.

À l'appel de la fonction, le *premier paramètre effectif* passe sa valeur au *premier paramètre formel*, le *deuxième paramètre effectif* passe sa valeur au *deuxième paramètre formel*, et ainsi de suite. La fonction exécute son code puis renvoie son résultat.

Les paramètres qui se correspondent doivent avoir des types compatibles.

Appel	Définition

Début
...
  mab ← maxdeux (17, 10)
...
Fin
 	
17
	10
	17
	fonction maxdeux(x, y : entier) : entier
	variables
	  mx : entier
	Début
	  si x > y alors
	  mx ← x
	  sinon
	  mx ← y
	  finsi
	  retourner(mx)
	Fin

5. Intérêt des fonctions

- Ne pas dupliquer du code
- Offrir une meilleure lisibilité car le lecteur peut comprendre ce que fait une fonction, uniquement à la lecture de son nom, sans avoir à déchiffrer du code.
- Partager le développement entre différentes équipes qui se spécialisent.
- Construire des bibliothèques de fonction pour réutiliser ce qui a déjà été fait.

[3] Attention rien à voir avec le type des paramètres !