Deuxième partie

Programmes modulaire

Chapitre 7

Procédures

Dans certains programmes, il arrive que des ensembles d'instructions se répètent. Considérons l'exemple suivant :

```
1  Tableau Entiers iTab Taille 10
2  Variable Indice iI
3
4  Début du Traitement
5     /* Remplissage du tableau avec les nombres de 1 à 10*/
6     Init
7     Compter avec iI de 0 à 9
8        iTab[iI] <- iI + 1
9     Fin de Compter
10
11    /* Affichage du tableau */
12    Init
13    Compter avec iI de 0 à 9
14       Afficher iTab[iI] Tab
15    Fin de Compter
16    Fin de ligne
17
18    /* Remplacement des nombres par leur carré */
19    Init
20    Compter avec iI de 0 à 9
21       iTab[iI] <- (iTab[iI]) * (iTab[iI])
22    Fin de Compter
23
24    /* Affichage du tableau */
25    Init
26    Compter avec iI de 0 à 9
27       Afficher iTab[iI] Tab
28    Fin de Compter
29    Fin de ligne
30
31 Fin du traitement
```

D'une manière évidente, les instructions 25 à 29 constituent une répétition des lignes 12 à 16. Tel quel, le programme présente les inconvénients suivants :

- ces instructions placées deux fois constituent une répétition fastidieuse ;
- les cinq lignes utilisées pour afficher le tableau détournent l'attention des tâches principales (à savoir le remplissage du tableau avec des valeurs) ;

- rien ne dit (à part le commentaire) ce que sont censées faire ces trois lignes ;
- l'affichage du tableau utilise et modifie la valeur de la variable `iI`, avec le risque d'interférer avec d'autres tâches en cours qui n'ont rien à voir avec l'affichage du tableau.
- il n'est pas possible de faire réaliser la tâche auxiliaire par un autre programmeur puisque son code est intimement mélangé au code du programme principal.

Une solution élégante pour éviter tous ces inconvénients consiste à extraire le code permettant l'affichage du tableau et à en faire une sous-routine. Une sous-routine qui réalise une tâche s'appelle une *procédure*.

7.1 Définition d'une procédure

Une procédure est une sorte de petit programme qui réalise une tâche précise et jouit d'une relative autonomie. Notamment, la manière exacte de réaliser la tâche importe peu. L'essentiel est de parvenir au résultat escompté. Une procédure pourra donc être réalisée par un programmeur différent de celui qui l'utilisera dans un programme.

Comme un programme normal, une procédure doit disposer de variables pour effectuer ses traitements. Nous verrons qu'une procédure dispose de trois types de variables :

- ses variables propres s'appellent des *variables locales*, elles ne sont reconnues qu'à l'intérieur de la procédure et ne conservent pas leur valeur après la fin de l'exécution de la procédure.
- les variables définies au niveau du programme sont aussi appelées *variables globales*. Elles reçoivent souvent des valeurs par l'action des instructions du programme principal. Les instructions de la procédure peuvent les manipuler également. Nous verrons que des modifications d'une variable globale par une sous-routine ne constituent pas souvent une bonne habitude de programmation, mais elles sont néanmoins possibles.
- les paramètres passés par le programme principal au moment de l'appel de la procédure seront examinés plus loin.

La structure générale d'une procédure sera donc la suivante :

```
Code procédure prNomProc(Paramètres éventuels)
Définitions Locales
/* déclarations des variables locales */
...
Fin des locales
/* Instructions */
...
Fin de procédure
```

Par convention, nous ferons commencer nos noms de procédure par les lettres minuscules `pr`. Le nom des procédures sera évidemment choisi pour en décrire la finalité. Nous pouvons ainsi proposer une version de notre affichage de tableau :

```
Code procédure prAfficherTableau()
   Définitions Locales
      Variable Indice iT
   Fin des locales
   Init
   Compter avec iT de 0 à 9
```

```
      Afficher iTab[iT] Tab
   Fin de Compter
   Fin de ligne
Fin de procédure
```

Nous voyons que cette procédure utilise deux variables : la variable globale `iTableau` et la variable locale `iT`, dont j'ai choisi le nom arbitrairement et différent de la variable utilisée dans le programme principal, par souci de clarté, mais ce n'était pas une obligation. La définition d'une procédure ne provoque aucune exécution. Une procédure ne s'exécute que si elle est *appelée* par le programme principal ou par une autre sous-routine.

7.2 Déclaration et appel d'une procédure

Le programme présenté au début de ce chapitre va donc utiliser la procédure `prAfficherTableau`. La plupart des langages exigent qu'une procédure soit définie pour pouvoir l'appeler. Le langage Pascal résout cela en faisant figurer le programme principal à la fin du code, le langage C préfère mettre le programme principal en tête du listing et prévoir des déclarations de procédures avant d'en détailler le code. Nous suivrons cette pratique.

Notre programme commencera donc par la déclaration du tableau et la déclaration de la procédure (son *prototype*). Quant à la variable `iI` utilisée pour parcourir le tableau, nous la déclarerons localement dans la routine principale, ce que nous appelons le *traitement*. Nous prendrons l'habitude de déclarer en tête du code les seules variables globales.

Dans le programme principal, il nous suffira de citer le nom de la procédure, `prAfficherTableau()` pour exécuter le code qu'elle contient.

```
Tableau Entiers iTab Taille 10
Prototype Procédure prAfficherTableau()

Début du Traitement
   Définitions Locales
      Variable Indice iI
   Fin des locales

   /* Remplissage du tableau */
   Init
   Compter avec iI de 0 à 9
      iTab[iI] <- iI + 1
   Fin de Compter

   /* Appel de la procédure d'affichage */
   prAfficherTableau()

   /* Modification du tableau */
   Init
   Compter avec iI de 0 à 9
      iTab[iI] <- iTab[iI] * iTab[iI]
   Fin de Compter

   /* Appel de la procédure d'affichage */
   prAfficherTableau()

Fin du traitement
```

```
Code procédure prAfficherTableau()
   Définitions Locales
      Variable Indice iT
   Fin des locales
   Init
   Compter avec iT de 0 à 9
      Afficher iTab[iT] Tab
   Fin de Compter
   Fin de ligne
Fin de procédure
```

7.3 Paramétrage d'une procédure

La procédure `prAfficherTableau()` ne permet que d'afficher un tableau d'entiers qui s'appelle `iTableau`. Nous pourrions avoir un programme qui manipule différents tableaux d'entiers, éventuellement de tailles différentes. Nous voudrions alors disposer d'une procédure plus souple pour laquelle le nom du tableau et sa taille seraient à préciser lors de son appel.

Il suffit pour cela de préciser les *paramètres* de la procédure, c'est à dire des variables formelles dont le nom ou la valeur sera suppléé réellement au moment de l'appel. Un paramètre qui figure dans une définition de procédure porte le nom de paramètre formel (ou paramètre au sens strict), un paramètre qui figure dans l'appel d'une procédure est dit paramètre réel ou *argument*. Ces paramètres sont toujours précisés entre les parenthèses qui suivent le nom de la procédure.

Le premier paramètre sera un tableau d'entiers. Notons que sa taille est ici sans importance, puisque il ne s'agit pas de réserver de l'espace pour ce tableau [1]. En effet, ce tableau aura été déclaré ailleurs et sa taille sera précisée dans sa déclaration. Le second paramètre est un entier. Nous indiquons `valeur` parce que cet argument est passé par valeur (nous verrons dans un prochain chapitre qu'il peut être passé autrement).

```
Tableau Entiers iTabTrois Taille 3
<* = {1,2,3} *>
Tableau Entiers iTabCinq Taille 5
<* = {5,10,15,20,25} *>
Tableau Entiers iTabDix Taille 10
<* = {10,20,30,40,50,60,70,80,90,100} *>
Prototype Procédure prAfficherTableau(
  Tableau Entiers iT, Valeur Entière iTaille)

Début du Traitement
```

1. Si nous voulons gérer un tableau à plusieurs dimensions, nous devrons spécifier la valeur de ces dimensions, sauf la première. On aura donc :

```
Prototype Procédure prTest(Tableau entiers iTab :5)
```

Il s'agit d'une contrainte imposée par le calcul des adresses en C. Nous n'entrerons pas dans les détails. Cela correspondrait à la déclaration C suivante :

```
void prTest(int iTab[][5]);
```

```
   prAfficherTableau(iTabTrois,3)
   prAfficherTableau(iTabCinq,5)
   prAfficherTableau(iTabDix,10)

Fin du traitement

Code Procédure prAfficherTableau(
   Tableau Entier iT, Valeur Entière iTaille)
   Définitions locales
      Variable Entière iI
   Fin des locales
   Init
   Compter avec iI de 0 à iTaille-1
      Afficher iT[iI] Tab
   Fin de Compter
   Fin de ligne
Fin de procédure
```

7.4 Avantages des procédures (et des fonctions)

La définition de procédures va nous offrir de nouvelles perspectives de programmation. Nous ajouterons quelques outils au cours des chapitres suivants et nous nous doterons notamment de fonctions, qui sont des procédures spéciales, capables de calculer une valeur. Ces sous-routines présentent les avantages suivants :

– débarrasser le programme principal d'une série de détails de mise en œuvre. Le programme principal se limite souvent à quelques appels de procédures. Voici à titre d'exemple le programme principal d'un jeu de Master Mind :

```
Début du traitement
   prAffichageRegles()
   Init
      prInitialisation()
   Tant que
      fnPartie() <> 4
   Répéter
      Moteur /* tout est dans fnPartie */
   Fin de boucle
   prAffichageFinal()
Fin de Traitement()
```

> Le texte de ce programme rejette évidemment tout le travail dans l'écriture de la fonction `fnPartie()` qui renverra la valeur 4 quand la partie sera terminée. Il y a fort à croire que le déroulement même de cette fonction consistera à appeler plusieurs sous-routines (lecture de la réponse du joueur, analyse, affichage du score...). L'idéal est de réussir à limiter chaque segment de programme à une vingtaine de lignes, qu'il est alors possible d'embrasser d'un seul coup d'œil [2].

2. Je dois concéder que, pour être convaincant, j'exagère un peu. La dernière procédure aura sans doute besoin de quelques informations pour réaliser son affichage final. Il faudra donc probablement l'une ou l'autre variables globales ou des paramètres pour faire transiter ces informations.

– permettre un morcellement du travail et une distribution au sein d'une équipe. Il convient alors de bien spécifier les conditions d'appel et le contrat réalisé par chaque sous-routine. C'est ici que prend tout son sens le conseil répété sans cesse pendant les exercices de laboratoires : respectez les directives.

Vocabulaire

ARGUMENT, FONCTION, PARAMÈTRES (RÉEL OU FORMEL), PROCÉDURE, PROTOTYPE, SOUS-ROUTINE

Exercices

VII-1. Écrire une procédure qui efface l'écran et met un titre en haut d'une page (par exemple le nom d'un programme, que vous inventerez).

VII-2. Écrire une procédure qui efface l'écran et y place un titre souligné par une ligne de 80 tirets.

VII-3. Définir uniquement les **prototypes** des procédures suivantes :

– afficher l'initiale du prénom et le nom d'une personne (deux données supposées connues du programme principal).
– afficher le nom, l'adresse et les coordonnées téléphoniques d'un client dont on connaît le numéro. Les données sont dans un fichier.
– ajouter un étudiant dont on connaît le nom, le prénom et la formation où il s'inscrit à la fin du fichier des inscriptions.
– afficher les correspondances entre deux tableaux de quatre nombres. Les détails exacts de l'affichage ne sont pas utiles dans cet exercice.

VII-4. Remplir un tableau de 10 éléments avec les multiples d'un nombre donné. Exemple, pour 7 : 7, 14, 21, 28, 35, 42, 47, 56, 63 et 70. Afficher ce tableau. Le programme serait parfait si vous disposiez de 2 procédures (`prRemplir()` et `prAfficher()`) pour réaliser le travail.

VII-5. Écrire les procédures `prAfficherEuro()` et `prAfficherFB()`, deux procédures qui prennent comme argument une somme en FB et l'affichent dans la monnaie voulue. Rappelons la parité de notre ancien franc par rapport à l'euro : 40.3399.

VII-6. Écrire une procédure `prFormater()` qui accepte deux arguments : un nombre entier à afficher et une largeur. Le nombre sera affiché avec des espaces initiaux pour remplir le champ. Si le champ fourni n'est pas assez large, la procédure affichera des '#', comme les tableurs. Exemple d'éxécution (la ligne manifeste la taille prévue) :

```
Nombre : 12345
--------
   12345
----
####
Nombre : 0
----
   0
```

Les exercices suivants constituent une série à faire dans l'ordre. Le principe est de gérer le contenu d'un tableau de manière dynamique. Un tableau `iTabDyn` de 20 éléments contiendra entre 0 et 20 valeurs entières intéressantes, classées en ordre croissant. On va lui associer une variable `iContenu` qui contiendra le nombre d'éléments à prendre en compte.

iTabDyn	5	10	15	20	25	13	14	20	8	9	10	11	12	13	14	0	0	0	0	0

iContenu	5

La situation initiale est illustrée par la figure ci-dessus. La variable `iContenu` nous signale qu'il faut tenir compte des 5 premiers nombres (5, 10, 15, 20 et 25). Les autres valeurs sont sans intérêt et ne doivent pas être initialisées.

VII-7. La première version du programme initialise le tableau pour qu'il soit conforme à la situation initiale et propose un menu avec deux options :

```
1. afficher les cases utiles du tableau[3]
9. quitter le programme.
```

VII-8. Suite de l'exercice précédent. Ajouter l'option suivante :

```
2. rechercher une valeur dans le tableau.
```

Elle doit permettre de rechercher une valeur dans le tableau et d'en indiquer la position. On affichera un message particulier si la valeur n'a pas été trouvée.

VII-9. Suite de l'exercice précédent. Ajouter l'option suivante :

```
3. ajouter une valeur dans le tableau
```

La valeur sera fournie par l'utilisateur. On veillera à respecter les règles suivantes :

- le tableau doit rester trié ;
- si on propose une valeur déjà présente dans le tableau, on ne l'ajoute pas (signaler l'erreur par un message) ;
- si le tableau est complètement rempli (vingt nombres déjà mémorisés), on n'ajoute plus rien et on signale l'erreur par un message.

VII-10. Suite de l'exercice précédent. Ajouter l'option suivante :

```
4. supprimer un nombre par sa position
```

L'utilisateur choisira la position du nombre qu'il veut supprimer. On veillera à vérifier que la position proposée corresponde à un élément existant (message d'erreur si nécessaire).

VII-11. Suite de l'exercice précédent. Ajouter l'option suivante :

```
5. supprimer un nombre par sa valeur
```

L'utilisateur fournira le nombre à supprimer. Si la valeur ne figure pas dans le tableau, un message d'erreur sera fourni.

3. En pseudo-code, vous pouvez utiliser l'instruction `Tab` pour espacer agréablement les nombres.