Elements d'algorithmique

TP 3 : procédures et fonctions

1 Motivation

Pour la réutilisabilité d'un programme, il est souvent utile de regrouper un morceau de code dans un sous-programme. On peut de cette manière réutiliser le morceau de programme en invoquant simplement le nom du sous-programme. De tels sous-programmes sont appelés **procédures**.

Exemple :

– Un programme simple sans procédure :

```
PROGRAM avec_procedure;
VAR
     i : INTEGER;
BEGIN
     FOR i := 1 TO 10 DO
         writeln(i);
     FOR i := 80 TO 85 DO
         writeln(i);
     FOR i := 100 TO 103 DO
         writeln(i);
     writeln('Fin');
     readln;
END.
```

– Un programme ayant la même exécution que le précédent mais avec une procédure :

```
PROGRAM avec_procedure;

PROCEDURE afficher_borne(inf : INTEGER; sup : INTEGER);
VAR
    i : INTEGER;
BEGIN
    FOR i := inf TO sup DO
       writeln(i);
END;

BEGIN
  afficher_borne(1,10);
  afficher_borne(80,85);
  afficher_borne(100,103);
  writeln('Fin');
  readln;
END.
```

2 Programmation modulaire

Maintenant, on distinguera d'une part les procédures déclarées grâce au mot-clé *PROCEDURE* et d'autre part, le programme principal dont les instructions sont comprises entre *BEGIN* et *END.* (notez le point à *END* au lieu d'un point-virgule). Lors que l'on exécute un programme, ce sont les instructions du programme principal qui sont successivement exécutées.

Le canevas général d'un programme Pascal est le suivant :

```
PROGRAM mon_program;

PROCEDURE procedure1(param1 : TYPE1; param2 : TYPE2);
CONST
    ... (* déclarations de constantes locales *)
VAR
    ... (* déclarations de variables locales *)
BEGIN
    ... ; (* instructions *)
    ... ;
END;

PROCEDURE procedure2;
VAR
    ... (* déclarations de variables locales *)
BEGIN
    ... ; (* instructions *)
    ... ;
END;

(* programme principal *)
CONST
    ... (* déclarations de constantes globales *)
VAR
    ... (* déclarations de variables globales *)
BEGIN
    ... ; (* instructions *)
    procedure2;
    ... ;
    readln;
END.
```

3 Les procédures

Le canevas général d'une procédure est le suivant :

```
PROCEDURE nom_procedure(param1 : TYPE1; param2 : TYPE2; ... ; paramN : TYPEN);
CONST
    ... (* déclarations de constantes locales *)
VAR
    ... (* déclarations de variables locales *)
BEGIN
```

```
    ... ; (* instructions *)
    ... ;
END;
```

On peut aussi écrire des procédures sans paramètre. L'en-tête de la procédure est alors la suivante :

```
PROCEDURE nom_procedure;
...
```

On peut appeler une procédure à partir du programme principal, ou même d'une autre procédure ou d'une autre fonction. En fait, à n'importe quel endroit d'un programme où on a besoin d'exécuter la procédure, on peut le faire. Pour appeler une procédure, il suffit d'écrire son nom et de fournir des paramètres. Bien entendu, le nombre et le type des variables ou des valeurs passées en paramètres doivent correspondre à ceux définis dans l'en-tête de la procédure.

3.1 Variables locales et variables globales

Pour réaliser une tâche, une procédure peut avoir besoin de variables. Les déclarations de telles variables se font à l'intérieur de la procédure en utilisant la syntaxe habituelle (mot-clé *VAR*). Ces variables sont dites **locales**. Elle n'existe que pendant l'exécution de la procédure (elles sont effacées de la mémoire à la fin de la procédure) et ne peuvent pas être utilisées dans le programme principal.

Par opposition, les variables du programme principal sont appelées **globales**. Elles peuvent être utilisées dans les différentes procédures.

3.2 Passage par valeur et par variable

Dans l'exemple de début, les variables *inf* et *sup* définies dans l'en-tête sont créés à l'appel de la procédure et détruites après son exécution.

Or il est quelque fois nécessaire d'appeler une procédure sans pour autant avoir de valeur à lui affecter mais on souhaiterait que ce soit celle-ci qui renvoie des valeurs. En d'autres termes, on peut être amené à faire en sorte que la procédure puisse modifier la valeur de la variable passée en paramètre par le programme appelant. Un exemple typique de ce genre de manipulation est la procédure *readln(variable)*. Ici, la variable passée en paramètre prendre une nouvelle valeur, celle saisie par l'utilisateur à 'appel de la procédure.

Une procédure supporte, en fait, deux types de paramètre : les paramètres fixes et les paramètres modifiables. Un paramètre est dit fixe lorsque la procédure ne peut pas modifier la valeur de la variable passée en paramètre par le programme qui l'appelle ; on parle alors de **passage par valeur**. Sinon la variable est modifiable et on parle de **passage par variable**.

Lors de la déclaration de variable au sein de la procédure, le mot-clé *VAR* (placée devant l'identificateur de la variable) permet de déclarer des paramètres modifiables.

Exemple :

```
PROGRAM procedure_IO_entier;

PROCEDURE entrer_entier(VAR n : INTEGER);
BEGIN
writeln('Entrez un entier:');
readln(n);
END;

PROCEDURE afficher_entier(n : INTEGER);
BEGIN
```

```
writeln('Vous avez entré:',n);
END;

VAR nombre : INTEGER;

BEGIN
entrer_entier(nombre);
afficher_entier(nombre);
END.
```

Ici, lors de l'appel à `entrer_entier`, la variable *nombre* du programme principal prend la valeur de la variable n de la procédure (qui prendra la valeur entrée par l'utilisateur). Si on avait enlevé le mot-clé *VAR* de la déclaration de n, dans l'en-tête de la procédure, la variable *nombre* ne prendrait aucune valeur de celle-ci.

4 Les fonctions

Un sous-programme qui retourne une valeur (comme *sqr* qui retourne le carré de l'entier en paramètre) est appelé **fonction**. On utilise alors le mot-clé *FUNCTION*. Le type du résultat retourné doit être défini dans l'en-tête de la fonction :

```
FUNCTION nom_fonction(param1 : TYPE1; ... ; paramN : TYPEN) : type_resultat;
CONST
    ... (* déclarations de constantes locales *)
VAR
    ... (* déclarations de variables locales *)
BEGIN
    ... ; (* instructions *)
    ... ;
    nom_fonction := resultat;
END;
```

La ligne `nom_fonction := resultat;` est primordiale. En effet, c'est elle qui renvoie le résultat de la fonction au programme qui appelle la fonction. Tout corps de fonction doit contenir une telle ligne.

Exemple de fonction :

```
FUNCTION puissance(x : REAL; exposant : INTEGER) : REAL;
VAR k : INTEGER;
    U : REAL;
BEGIN
    U := 1;
    FOR k := 1 TO exposant DO
    BEGIN
            U := U * x;
    END;
    puissance := U;
END;
```

5 Exercices

5.1 Exercice 1

Écrire un programme affichant les n premiers termes de la suite de Fibonacci. Dans le programme principal, on demandera à l'utilisateur de rentrer la valeur de n, une procédure s'occupera de l'affichage des termes de la suite.

5.2 Exercice 2

Écrire une procédure qui permet de permuter deux nombres entiers. La procédure devra prendre en paramètre deux variables modifiables.

5.3 Exercice 3

Écrire une procédure qui permet de calculer la valeur en n de la suite récurrente $u_i = i.u_{i-1}$. La procédure devra avoir deux paramètres : un paramètre fixe n et un paramètre modifiable u qui correspondra aux différentes valeurs de la suite. Que rend ce programme en résultat lorsque $u_0 = 1$?

5.4 Exercice 4

Écrire une fonction *factorielle* qui calcule $n!$. Dans le programme principal, l'utilisateur sera invité à entrer un entier n puis $n!$ sera affiché.

5.5 Exercice 5