

Plan	Sous-programmes	Globales/Locales	Les paramètres	Exercices
	○○○ ○○○	○ ○ ○ ○	○ ○ ○○○	

## Le langage Pascal(3)

Nour-Eddine Oussous

S3H – Programmation

28 septembre 2009

Plan	Sous-programmes	Globales/Locales	Les paramètres	Exercices
	○○○ ○○○	○ ○ ○ ○	○ ○ ○○○	

## Les sous-programmes

- ▶ Le PASCAL, permet de découper un programme en plusieurs parties : modules
- ▶ La modularité permet :
  - ▶ d'éviter des séquences d'instructions répétitives : *factorisation du code*
  - ▶ le partage d'outils communs qu'on écrit une seule fois
  - ▶ la lisibilité des programmes
- ▶ Le découpage d'un programme en modules peut être répété sur les modules eux-mêmes
- ▶ Un module est un sous-programme qui peut être une **fonction** ou une **procédure**

Plan	Sous-programmes	Globales/Locales	Les paramètres	Exercices
	○○○ ○○○	○ ○ ○ ○	○ ○ ○○○	

### Les sous-programmes

Les fonctions

Les procédures

### Les variables locales/globales

La notion de bloc

Les variables globales

Les variables locales

Exemple

### Les paramètres

Définitions

Mode de passage des paramètres

Exemple

### Exercices

Plan	Sous-programmes	Globales/Locales	Les paramètres	Exercices
	●○○ ○○○	○ ○ ○ ○	○ ○ ○○○	

Les fonctions

## Les fonctions

- ▶ une *fonction* permet d'*abstraire* et de *nommer* un calcul ou une expression
- ▶ une fois définie, une fonction peut être utilisée comme une nouvelle expression du langage

### Syntaxe

```

1 function nom_fonction(liste_param) : type_result ;
2   { partie declarations }
3 begin
4   { partie instructions }
5 end ; {function}

```

Plan	Sous-programmes	Globales/Locales	Les paramètres	Exercices
	○●○ ○○○	○ ○ ○ ○	○ ○ ○○○	
Les fonctions				
<h1>Les fonctions</h1>				

### Remarques

- ▶ La première ligne s'appelle l'*entête* de la fonction
- ▶ Dans le corps de la fonction, on doit avoir au moins une instruction `nom_fonction:=expression`
- ▶ La déclaration d'une fonction se fait dans la partie *déclarations* d'un programme ou d'un sous-programme
- ▶ L'appel d'une fonction c'est son utilisation

Plan	Sous-programmes	Globales/Locales	Les paramètres	Exercices
	○○○ ○○○	○ ○ ○	○ ○ ○○○	
Les fonctions				
<h1>Les fonctions</h1>				

### Exemple

Une fonction qui calcule la factorielle d'un entier  $n \geq 0$ .

$$0! = 1 \quad \text{et} \quad n! = 1 \times 2 \times \dots \times n$$

```

1 function fact(const n: Cardinal) : Cardinal ;
2   var f,i: Cardinal ;
3   begin
4     f := 1 ;
5     for i := 1 to n do
6       begin
7         f := f * i
8       end; {for}
9     fact := f ;
10  end; {fact}

```

Plan	Sous-programmes	Globales/Locales	Les paramètres	Exercices
	○○○ ●○○	○ ○ ○	○ ○ ○○○	
Les procédures				
<h1>Les procédures</h1>				

- ▶ une *procédure* permet de créer une nouvelle action, ou encore d'*abstraire* et *nommer* une suite d'instructions
- ▶ une fois définie, une procédure peut être utilisée comme une nouvelle instruction du langage

### Syntaxe

```

1 procedure nom_procedure(liste_param) ;
2   { partie declarations }
3 begin
4   { partie instructions }
5 end; {proc}

```

Plan	Sous-programmes	Globales/Locales	Les paramètres	Exercices
	○○○ ○○○	○ ○ ○	○ ○ ○○○	
Les procédures				
<h1>Les procédures</h1>				

### Remarques

- ▶ La première ligne s'appelle l'*entête* de la procédure
- ▶ La déclaration d'une procédure se fait dans la partie *déclarations* d'un programme ou d'un sous-programme
- ▶ L'appel à une procédure est une instruction

Plan	Sous-programmes	Globales/Locales	Les paramètres	Exercices
	○○○ ○○●	○ ○ ○	○ ○ ○○○	
Les procédures				
<h2>Les procédures</h2>				

### Exemple

Une procédure qui permet d'afficher le contenu d'un tableau de type T\_Tableau.

```

1 procedure afficher(const T: T_Tableau) ;
2 var i: Cardinal ;
3 begin
4   writeln('Contenu du tableau T:');
5   for i := low(T) to high(T) do
6     begin
7       write(T[i], ' ')
8     end; {for}
9   writeln ;
10 end ; {afficher}

```

Plan	Sous-programmes	Globales/Locales	Les paramètres	Exercices
	○○○ ○○○	○ ● ○ ○	○ ○ ○○○	
Les variables globales				
<h2>Les variables globales</h2>				

- ▶ Les variables globales sont communes à plusieurs sous-programmes et leur permettent d'échanger des informations
- ▶ Cette notion est relative : une variable peut être globale pour certains blocs et locale à un bloc
- ▶ Une variable déclarée par exemple dans le programme principal avant les sous-programmes est globale : elle peut être utilisée dans tout ce qui suit sa déclaration
- ▶ Les variables globales servent à transmettre les valeurs aux sous-programmes

Plan	Sous-programmes	Globales/Locales	Les paramètres	Exercices
	○○○ ○○○	● ○ ○ ○	○ ○ ○○○	
La notion de bloc				
<h2>La notion de bloc</h2>				

- ▶ Un programme est un bloc formé d'une *partie déclarations* et une *partie instructions*.
- ▶ Dans la *partie déclarations*, on trouve en particulier les déclarations de sous-programmes (fonctions et/ou procédure)
- ▶ Les sous-programmes sont eux-mêmes des blocs structurés de la même façon

Plan	Sous-programmes	Globales/Locales	Les paramètres	Exercices
	○○○ ○○○	○ ○ ● ○	○ ○ ○○○	
Les variables locales				
<h2>Les variables locales</h2>				

- ▶ Une variable est dite locale à un sous-programme si elle est déclarée dans celui-ci
- ▶ Une variable locale n'est connue qu'à l'intérieur du bloc où elle est définie. Sa portée est donc limitée à ce bloc.
- ▶ Une variable locale a une durée de vie limitée à celle d'une exécution du bloc où elle figure
- ▶ Une variable locale cache la variable globale de même nom.

Plan	Sous-programmes ○○○ ○○○	Globales/Locales ○ ○ ○ ●	Les paramètres ○ ○ ○○○	Exercices
Exemple				
<h1>Exemple</h1>				

### Exemple

```

1 program exemple ;
2 var i: Cardinal ;
3   r: Real ;
4   //-----
5   procedure p1(const i: Integer) ;
6     var r: Real ;
7     //-----
8     procedure p2 ;
9       var r: Real ;
10      begin
11        ...
12      end; // p2 -----
13    begin
14      ...
15    end; // p1 -----
16  begin
17    ...
18  end.

```

Plan	Sous-programmes ○○○ ○○○	Globales/Locales ○ ○ ○	Les paramètres ● ○ ○○○	Exercices
Définitions				
<h1>Les paramètres</h1>				

- ▶ Les paramètres rendent plus souple l'utilisation des sous-programmes
- ▶ On distingue deux types de paramètres :
  - ▶ Les *paramètres formels* : permettent de décrire l'algorithme d'un sous-programme
  - ▶ Les *paramètres effectifs* : permettent de transmettre au sous-programme l'objet réel sur lequel on veut qu'il s'exécute

Plan	Sous-programmes ○○○ ○○○	Globales/Locales ○ ○ ○	Les paramètres ○ ● ○○○	Exercices
Mode de passage des paramètres				
<h1>Mode de passage</h1>				

- ▶ **entrée** le sous-programme ne fait que *consulter* la valeur du paramètre. On utilise le mot clef `const`
- ▶ **sortie** on souhaite que le sous-programme *initialise* le paramètre effectif lors d'un appel du sous-programme. On utilise le mot clef `out`
- ▶ **entré et sortie** on souhaite que le sous-programme *modifie* le paramètre effectif en fonction de sa valeur actuelle. On utilise le mot clef `var`
- ▶ Si l'on ne met pas de mot clef, le paramètre est en entrée et peut être modifié *localement*.

Plan	Sous-programmes ○○○ ○○○	Globales/Locales ○ ○ ○	Les paramètres ○ ○ ●○○	Exercices
Exemple				
<h1>Exemple</h1>				

### Exemple

```

1 function est_pair(const N: Integer) : Boolean ;
2 begin
3   est_pair := N mod 2 = 0 ;
4 end {est_pair}

1 procedure ecrire_si_pair(const N: Integer) ;
2 begin
3   if est_Pair(N) then
4     begin
5       writeln(N,' est pair') ;
6     end
7   else
8     begin
9       writeln(N,' est impair') ;
10    end; {if}
11 end; {ecrire_si_pair}

1 ecrire_si_pair(6) ;
2 n := 7 ;
3 ecrire_si_pair(3*n+1) ;

```

Plan	Sous-programmes	Globales/Locales	Les paramètres	Exercices
	○○○ ○○○	○ ○ ○	○ ○ ○○○	
Exemple				
<h2>Exemple</h2>				

### Exemple

```

1 procedure remettre_a_zero(out N: Integer) ;
2 begin
3   N := 0 ;
4 end ; {remettre_a_zero}

1 procedure incrementer(var N: Integer) ;
2 begin
3   N := N + 1 ;
4 end {incrementer}

1 procedure incrementer_de(var N: Integer;
2                           const combien: Integer) ;
3 begin
4   N := N + combien ;
5 end ; {incrementer_de}

```

Plan	Sous-programmes	Globales/Locales	Les paramètres	Exercices
	○○○ ○○○	○ ○ ○	○ ○ ○○●	
Exemple				
<h2>Exemple</h2>				

### Exemple

```

1 var X : Integer ;
2 begin
3   { X est indefini }
4   remettre_a_zero(X) ;
5   { X = 0 }
6   Incrementer(X) ;
7   { X = 1 }
8   incrementer_de(X, 13) ;
9   { X = 14 }
10  incrementer_de(X, X*X) ;
11  { X = 14 + 14*14 = 210 }
12  incrementer_de(X+X, 2) ; { engendre une erreur }
13 end.

```

Plan	Sous-programmes	Globales/Locales	Les paramètres	Exercices
	○○○ ○○○	○ ○ ○	○ ○ ○○○	

### Exercices

- Écrire un programme lit une liste de nombres positifs terminée par 0 et affiche à l'écran
  - leur somme
  - leur moyenne
  - la plus grande valeur saisie
  - la plus petite valeur saisie

Donnez deux versions : une utilisant la boucle `while` et l'autre utilisant la boucle `repeat`.

- Écrire un programme qui convertit une chaîne de caractères représentant convenablement un nombre en un nombre réel. La chaîne est saisie au clavier. Donner une version avec le `while` et l'autre avec le `repeat`.