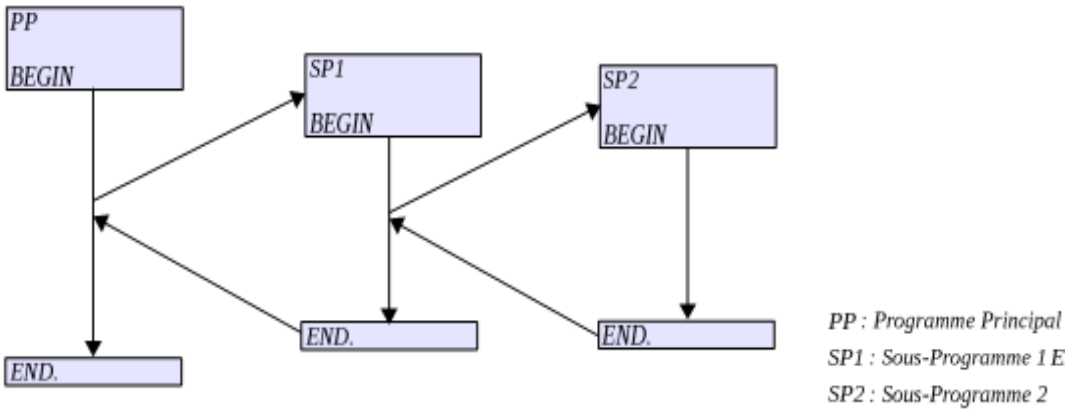
CHAP1 FONCTIONS ET PROCEDURES

1-INTRODUCTION :

En général, les problèmes qui se pose en pratique sont suffisamment complexes et nécessitent leur décomposition en sous problèmes plus faciles à résoudre séparément.. D'où vient l'idée de décomposition d'un programme complexe en sous-programmes (fonctions et procédures). De même lorsqu'une partie de code doit être exécutée plusieurs fois, à des endroits différents, ou réutilisée ultérieurement on pourra ne l'écrire qu'une fois et lui donner un nom en faisant une Fonction ou une Procédure, pour éviter les redondances de codes dans le programme. Par exemple pour calculer la somme suivante : S_n=1 !+2 !+3 !+4 !+5 !+6 !+.....+n ! , on doit calculer le factoriel n fois (pour le 1 ;2 ;3 ;4 ;.... ;n). Au lieu d'écrire le code de calcul du factoriel n fois dans le programme, on définir une fonction ou procédure qui calcul celui-ci et on fait appel au besoin.

2-NOTION DE SOUS PROGRAMME :

Un sous programme (procédure ou fonction) est un programme à l'intérieur d'un autre programme. Il possède la même structure que le programme principal. Il peut être appelé par le programme principal ou par un autre sous programme pour réaliser un certain traitement et retourner des résultats. Le schéma illustre le mécanisme d'appel et retour des sous-programmes.



Il existe des procédures et des fonctions prédéfinies simples comme write(x), read(x), sqrt(x), ou cos(x). Si on besoin d'un traitement que le PASCAL n'offre pas dans sa bibliothèque, on peut définir nos propres procédures et fonctions.

3- Déclaration de fonction

La déclaration d'une fonction s'écrit toujours avant le corps principal du programme et avant toutes les autres procédures et fonctions qui l'utilisent. La syntaxe de la déclaration est la suivante :

```
Function nomdefonction (nomvar1 : type1 ; nomvar2 : type2 …) : type de la fonction;
var (* Déclaration d'éventuelles variables locales *)
  nomvar1 : type1;
  nomvar2 : type2;
  …
begin
 (* Corps de la fonction *)
  instruction1;
  instruction2;
   …
  nomdefonction := expression; (* Affectation de la valeur finale *)
end;
```

-Comme leur nom l'indique, les variables "locales" ne peuvent servir qu'à un calcul local, à l'intérieur de la fonction, elles ne sont donc pas connues ailleurs ! Grâce à cette propriété, on peut déclarer une variable locale avec un nom déjà utilisé pour une variable d'une autre fonction. Celles-ci sont indépendantes l'une de l'autre et ne sont accessibles qu'à l'intérieur de la fonction où elles ont été déclarées. Attention, les variables déclarées au début du programme sont dites globales, ce qui signifie qu'elles sont connues dans tout le programme.

En cas d'ambiguïté avec une variable locale, c'est la variable locale qui est reconnue.

- Les paramètres d'une fonction ont les mêmes propriétés que les variables locales.

• Appel d'une fonction

Toute fonction qui a été déclarée peut être utilisée dans une expression. Il suffit d'écrire son nom suivi d'une liste de "paramètres d'appel" entre parenthèses séparés par des virgules. Pour que l'appel de la fonction soit correcte, il doit y avoir autant de paramètres d'appel que de paramètres déclarés et le type de chacun d'eux doit être rigoureusement identique.

Exemple1 : fonction calculant le volume d'un cube

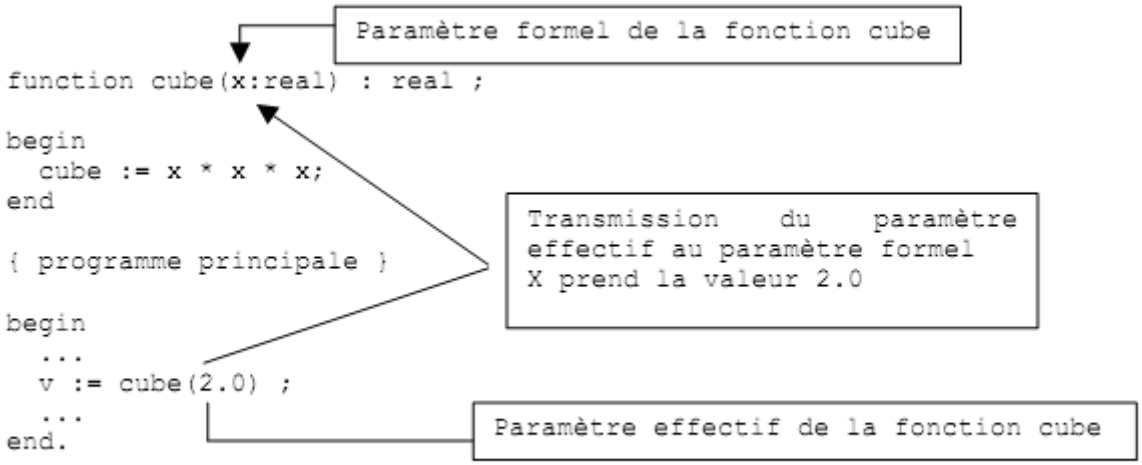
```
Function cube( x : real ) : real ;

begin
  cube := x * x * x ;
end ;
```

Remarques sur l'exemple :

-le mot clé **real** aprés le nom de la fonction indique que la fonction cube renvoie une valeur de type real. Les valeurs de retour des fonctions sont renvoyées au programme appelant via l'instruction contenant le nom de la fonction : **cube :=...**

-Le nom de la fonction (ici cube) est complété par une paire de paranthèses avec des paramètres. Cette fonction a un patamètre, à savoir une variable **real** nommée **x,** accompagnée de la spécification de son type. Les paramètres spécifiés dans la définition de la fonction sont qualifiés de **paramètres formels**. Il faut les distinguer des paramètres qui seront transmis à la fonction lorsqu'on l'appellera qui sont dits **paramètres effectifs** ou arguments de la fonction.



Exemple2 : fonction qui calcule le factoriel .

```
function fact(n:integer) : integer;
     var
          i, f:integer;
begin
     f:=1;
     for i:=1 to n do
          f := f*i;
     fact := f; {toujours la dernière instruction <nom_fonction> := <résultat>}
end;
```

La fonction **fact** prend comme paramètre un nombre n et retourne son factoriel. La dernière instruction de la fonction est : fact := f ; lors de l'appel de cette fonction on doit passer des paramètres par exemple : fact(5) qui retourne la valeur 120.

Exemple de programme comportant la fonction fact :

```
program factoriel; (* Calcul des 10 premières factorielles *)

  var j : integer; (*j est variable globale*)

  function  fact ( n : integer ) : real ;                    paramètre formel
     var  i : integer; (*i est variable loale à la fonction*)
          f : real ; (*f est variable loale à la fonction*)
  begin
    f := 1;
    for i:=1 to n do f := f*i;
    fact := f;
  end;

 begin (* Le programme commence ici *)
    for  j:=1  to  10  do  writeln (fact( j ));
 end.
                                          Paramètre effectif
```

4- Déclaration de procédure

Comme pour une fonction, l'intérêt d'une procédure est de pouvoir regrouper les instructions complexes d'un traitement et de lui donner un nom. Toutefois, une fonction renvoie une valeur, alors qu'une procédure ne le fait pas. Une procédure joue en fait le rôle d'une méta-instruction regroupant plusieurs instructions.

Une procédure se déclare comme une fonction, sauf qu'elle n'a pas de type, ne renvoie pas de valeur et la déclaration commence par "procedure" suivi de l'identificateur. Comme pour une fonction, une procédure doit être déclarée avant le bloc principal et avant toute procédure ou fonction y faisant référence.

```
procedure nomdelaprocédure (nomvar1 : type1 ; nomvar2 : type2 …);
var (* Déclaration d'éventuelles variables locales *)
  nomvar1 : type1;
  nomvar2 : type2;
  …
begin
 (* Corps de la procédure *)
  instruction1;
  instruction2;
   …
end;
```

Exemple de programme comportant une procédure :

```
program facto; (* Calcul des 10 premières factorielles *)

  var i : integer;

  procedure  ecritfactorielle ( n : integer );      (* Affiche la valeur de n!  *)
       var  j : integer;
        fact : real;
  begin
    fact := 1;
    for j:=1 to n do fact := fact*j;
    writeln ('La factorielle de ', n , ' est ', fact );
  end;

begin (* Le programme commence ici *)
   for  i:=1  to  10  do  ecritfactorielle ( i );
end.
```

5- Notion de paramètres formels et paramètres effectifs :

Les paramètres **formels** sont les paramètres utilisés dans la déclaration des procédures ou fonctions. Les paramètres formels sont séparés par des **point-virgules.**

Lors de l'appel à une fonction ou une procédure on doit donner des valeurs ou des variables à la place des paramètres formels : ce sont les paramètres **effectifs**. Les paramètres effectifs sont séparés par des **virgules**.

Donc les paramètres formels sont ceux utilisés dans la déclaration des procédures et fonctions. Par contre les paramètres effectifs sont ceux utilisés lors de l'appel aux procédures et fonctions.

6- Passage des paramètres (transmission des paramètres) :

La communication entre les fonctions et procédures entre elles et avec le programme principal se fait par le mécanisme de transmission des paramètres. On distingue deux types de passage ou transmission de paramètres :

-paramètres transmis par valeur.

-paramètres transmis par variable ou par référence.

6-1-Transmission des paramètres par valeur :

Pour la transmission des paramètres par valeurs il suffit de déclarer les paramètres formels sans le mot clé **var.** Comme dans l'exemple de la déclaration de la fonction :

```
function  fact ( n : integer ) : real ;
```

le paramètre n est transmis par valeur. Dans le paramètre effectif correspondant on peut passer une valeur constante, une expression, une variable… à condition que le paramètre formel et le paramètre effectif soient de même type, exemple :

fact(5) ; fact(2+3) ; fact(x+4) ; fact(a+b*2);

Dans la transmission des paramètres par valeur, l'expression est évaluée et sa valeur est affectée au paramètre déclaré. les changements réalisés sur les paramètres formels n'affectent pas les paramètres effectifs.

6-2-Transmission des paramètres par variable (ou par référence) :

On utilise la transmission par variable lorsqu'on veuille que la variable du paramètre formel dans la fonction ou la procédure affecte le paramètre effectif correspondant au niveau du programme appelant. Dans ce cas le paramètre formel doit être précédé du mot-clé **var**. Le paramètre effectif correspondant **doit être obligatoirement une variable.**

En d'autres termes, on utilise une transmission par variable lorsqu'on veut recueillir le résultat dans cette variable.

Exemple 1: Soit la procédure produit suivante :

```
Procédure produit(a, b : real ; var p : real) ;
Begin
  P := a*b ;
End ;
```

Dans cette procédure, les parmètres a et b sont définis sans utiliser le mot-clé **var**, donc la transmission lors de l'appel de cette procédure sera par valeur. Par contre, le paramètre p est défini par le mot clé **var**, donc la transmission sera par variable (le paramètre effectif doit être une variable).
Exemple d'appel : produit(5,6,x) ; 5 et 6 sont des paramètres effectifs transmis par valeur. Le paramètre effectif x est transmis par variable ou référence, le résultat du produit sera sauvgardé dans la variable x.

Exemple2 :

```
program test; { Juste pour comprendre l'influence de var devant le paramètre déclaré }
  var x, y : integer;
  procedure affiche ( var r : integer ; s : integer );
  begin
     writeln ( 'x vaut ', x , ' , r vaut ', r , '  , y vaut ', y , '  et s vaut ', s);
     r := 1; s:=1;
     writeln ( 'x vaut ', x , ' , r vaut ', r , '  , y vaut ', y , '  et s vaut ', s);
  end;
begin
  x := 0; y := 0;
  affiche ( x,y );
end.
```

L'affichage à l'écran est : x vaut 0, r vaut 0, y vaut 0 et s vaut 0
x vaut 1, r vaut 1, y vaut 0 et s vaut 1

Commentaires :
x et y sont initialisés à 0 au début du programme. A ce moment là, r et s ne sont pas connus.
A l'appel de la procédure affiche, x est mis en correspondance avec r, donc ils valent tous les deux 0 et y est mis en correspondance avec s, donc ils valent 0 également, ce qui explique le premier affichage.
Pour r, comme c'est un passage par référence (il y a "var" devant), toute modification est répercutée sur x, donc r et x prennent la valeur 1. En ce qui concerne s, c'est un passage par valeur, donc s prend la valeur 1 mais y n'est pas modifié, ce qui explique le deuxième affichage.

Exercices

1) Écrire une procédure permettant de déterminer si un nombre est premier. Elle comportera deux arguments : le nombre à examiner, un indicateur booléen précisant si ce nombre est premier ou non.
2) Écrire la procédure précédente sous forme de fonction.(facultatif)
3) Écrire une fonction calculant la norme d'un vecteur à 3 composantes réelles.
4) Écrire la fonction précédente sous forme d'une procédure.