Algorithmique et Programmation

L2 MPI-MI

Dr Ousmane DIALLO

ChapN: Les sous programmes

6.1. La programmation modulaire

6.1.1. Définition

- Si on a un gros programme à mettre au point, et que certaines parties sont semblables ou d'autres très complexes, alors il faut absolument structurer son programme pour ne pas risquer d'y passer trop de temps, lors de modifications ultérieures, ce qui en facilitera la maintenance.

→ il faut adopter la stratégie "diviser pour régner" et chercher à utiliser au maximum "l'existant".

- La programmation modulaire consiste à décomposer le problème initial en sous-problèmes de moindre complexité. Cette décomposition se poursuit jusqu 'à l'obtention de sous-problèmes plus compréhensibles et plus simples à programmer.
  - On parle d'analyse descendante (ie, on part du niveau le plus complexe et on aboutit à un niveau le plus simple possible du problème).

ChapN: Les sous programmes

6.1. La programmation modulaire

6.1.1. Définition

- Par ailleurs on est souvent amené à effectuer plusieurs fois un même ensemble de traitements dans un programme. Cet ensemble de traitements constitue un sous-programme (ou module).
- Cependant pour éviter cette réécriture, on attribue un nom à ce dernier et à chaque fois que son utilisation est souhaitée, on lui fait appel par l'intermédiaire de ce nom.

Avantages :

- la lisibilité
- la structuration
- possibilité d'utiliser un identificateur significatif

ChapN: Les sous programmes

6.1. La programmation modulaire

6.1.1. Définition

Exemple :

```
Begin
      bloc1
      bloc2
      …
      bloc1
      …
      bloc2
      bloc1
      …
end.
```

où bloc1 et bloc 2 peuvent être des blocs de plusieurs dizaines de lignes de code.

on préférera écrire :

```
procedure P1
begin
      bloc1
end;
procedure P2
begin
      bloc2
end;
begin
      P1;
      P2;
      …
      P1;
      ….
      P2;
      P1;
      …
end.
```

ChapN: Les sous programmes

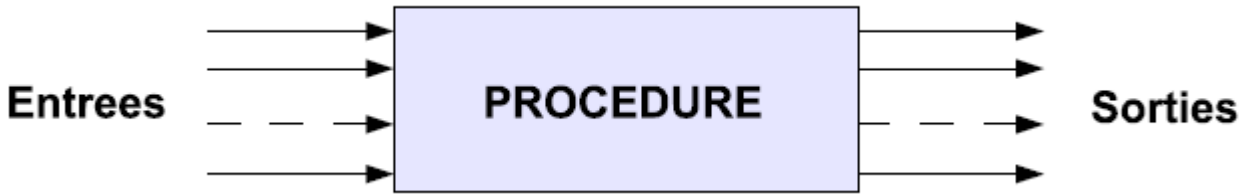
6.1. La programmation modulaire

6.1.2. Les différents types de modules

❑En Pascal, on distingue deux types de modules :

•**Les procédures** : Il s'agit de sous-programmes *nommés* qui représentent une ou plusieurs actions, et qui peuvent calculer et ***retourner une ou plusieurs valeurs***.

- Une procédure ***P*** doit être définie une seule fois dans un algorithme ***A*** et peut appelée plusieurs fois dans ***A*** ou par une autre procédure de ***A***.



ChapN: Les sous programmes

6.1. La programmation modulaire

6.1.2. Les différents types de modules

❑En Pascal, on distingue deux types de modules :

•**Les Fonctions** : Il s'agit de sous programmes *nommés* et *typés* qui calculent et ***retournent une et une seule valeur***. Le type de cette valeur est celui de la fonction.

- De même une fonction ***F*** doit être définie une seule fois dans l'algorithme ***A*** et peut être appelée plusieurs fois par ***A*** et par toutes les autres procédures et fonctions de *A*.

ChapN: Les sous programmes

6.2. Les procédures

6.2.1. Syntaxe et déclarations

- Les déclarations s'effectuent dans l'ordre suivant :

EN-TETE

DECLARATIONS

1. CONSTANTES
2. TYPES
3. VARIABLES
4. FONCTIONS / PROCEDURES

BLOC D'INSTRUCTIONS EXECUTABLES

ChapN: Les sous programmes

6.2. Les procédures

6.2.1. Syntaxe et déclarations

- **Syntaxe de déclaration:** La définition d'une procédure comprend trois parties :
  - **Partie 1** : *L'entête* :

    Procedure *Nom_proc[(Identif1[, …]:Type1[;[var] Identif2[, …]:Type2])];*

  - **Partie 2** : *Déclaration des variables locales*

    Nom_variable_i : type_variable_i;

  - **Partie 3** : *Corps de la procédure*

    Begin

    <instruction1>;

    <instruction2>;

    …

    <instructionN>

    End;

ChapN: Les sous programmes

6.2. Les procédures

6.2.1. Syntaxe et déclarations

- Les identificateurs qui suivent le nom de la procédure constituent l'ensemble des ***paramètres formels***.
- Le mot ***var*** (optionnel) n'est utilisé que pour les paramètres de sortie et d'entrée/sortie.

- Une fois la procédure déclarée, elle peut être utilisée dans le programme principal par un "*appel*" à cette procédure, à partir du programme principal ou à partir d'une autre procédure.
- **Appel d'une procédure**

Nom_procedure(expression { , expression }) ;

ChapN: Les sous programmes

6.2. Les procédures

Exemples

```
(*Ecrire un programme qui calcule successivement la moyenne de la taille de trois individus ainsi que la
moyenne de leur poids et celle de leur âge. Version sans emploi de procédure (à rejeter)*)

Program Moyenne;
Var A, B, C, M : real;
Begin
        writeln ('Entrez la taille des 3 individus : ');
        readln (A, B, C);
        M := (A+B+C) / 3;
        writeln ('La moyenne est : ', M:8:2);
        writeln('Entrez le poids des 3 individus : ');
        readln (A, B, C);
        M := (A+B+C) / 3;
        writeln ('La moyenne est : ', M:8:2);
        writeln('Entrez l' 'âge des 3 individus : ');
        readln (A, B, C);
        M := (A+B+C) / 3;
        writeln ('La moyenne est : ', M:8:2);
End.
```

Cette version n'est pas idéale, car nous constatons qu'une même partie du programme est écrite trois fois. Il est préférable de regrouper ces instructions dans une procédure. La nouvelle version, avec emploi d'une procédure, est la suivante :

ChapN: Les sous programmes

6.2. Les procédures

Exemples

```
Program MOYENNE_BIS;
Var A, B, C, M : real;
Procedure CALCUL;
Begin
        readln (A, B, C);
        M := (A + B + C) / 3;
        writeln ('La moyenne est ', M:8:2);
End;
Begin   { Début du programme principal }
        writeln ('Entrer les tailles de 3 individus');
        CALCUL;
        writeln ('Entrer les poids de 3 individus');
        CALCUL;
        writeln('Entrez l' âge des 3 individus : ');
        CALCUL;
End.
```

ChapN: Les sous programmes

6.2. Les procédures
Exemples

```
(*Pour calculer la somme de deux matrices A et B, il faut lire d'abord ces deux matrices. Ainsi au lieu d 'écrire deux fois
le sous-programme de lecture d'une matrice, on écrit plutôt une procédure et on l'utilise pour saisir A et B.*)

Program SOMME_MATRICIELLE;
Const nmax=50;
Type MATRICE = ARRAY[1..nmax,1..nmax] of integer;
Var A, B, S : MATRICE;
        n : integer;
{Procédure de saisie d'une matrice }
Procedure SAISIE(Var M : MATRICE; dim : integer);
Var i, j : integer;
Begin
    For i:=1 To dim Do
        For j:=1 To dim Do
        Begin
            write('Donnez l' 'élément M[',i,j,'] ');
             readln(M[i,j]);
        End;
End;
```

ChapN: Les sous programmes

6.2. Les procédures

Exemples

```
                                        {Procédure de calcul de la somme de deux matrices }
Procedure SOMME(M1 : MATRICE;M2 : MATRICE; var C : MATRICE , dim : integer);
Var i, j : integer;
Begin
    For i:=1 To dim Do
        For j:=1 To dim Do
            C[i,j]:=M1[i,j]+M2[i,j];
End;

                                        {Procédure d'affichage d'une matrice }
Procedure AFFICHAGE(M : MATRICE; dim : integer);
Var i, j : integer;
Begin
    For i:=1 To dim Do
        Begin
            For j:=1 To dim Do
                Write(M[i,j],' ');
                writeln;
        End;
End;
```

6.2. Les procédures

Exemples

```
Begin    { Début du programme principal }
     write ('Entrer la dimension des deux matrices ');
     readln(n);
     writeln (' Lecture de la première matrice ');
     SAISIE(A,n);
     writeln (' Lecture de la seconde matrice ');
     SAISIE(B,n);
     SOMME(A,B,S,n);    {Calcul de la somme de A et B}
     AFFICHAGE(S,n);    {affichage du résultat}
     readln;
End.
```

6.3. Les fonctions

6.3.1. Utilité des fonctions

- D'une manière générale :
  - Les procédures ont pour effet de modifier l 'état des variables d'un programme. Elles constituent une partie du programme qui se suffit à elle-même. Une procédure peut ne pas retourner des résultats au (sous-) programme appelant (cas de la fonction CALCUL dans l'exemple précédent).
  - Les fonctions permettent de définir le calcul d'une valeur d'un type donné et de rendre un résultat au (sous-) programme. Le type de la valeur retournée est très souvent un type de données simple.
- On distingue deux catégories de fonction en Turbo-Pascal :
  - **Les fonctions implicites**.

  Parmi les fonctions implicites, encore appelées primitives de base du système de programmation on distingue :
    - ✓ les fonctions usuelles représentées par les opérateurs et,
    - ✓ les fonctions standard prédéfinies telles que **Sqr**, **Sqrt**, **Sin**, etc.
  - **Les fonctions explicites**.
  - les fonctions explicites regroupent toutes les autres fonctions calculables à définir par l'utilisateur lui-même. En général, ces fonctions sont de la forme :
  - *F(X1,X2,…,XN) où* F est le nom de la fonction et X1,X2,…,XN les arguments

ChapN: Les sous programmes

6.3. Les fonctions
6.3.2. Définition

- Une fonction est un bloc d'instructions qui doit retourner une valeur de type simple au point d'appel.
- En général, le rôle d'une fonction est le même que celui d'une procédure.
- Il y a cependant quelques différences :
  - Arguments de la fonction : paramètres typés
  - Un ensemble d'arrivée typé
  - Renvoi d'une valeur unique
- Type de la fonction :
  - scalaire, réel, string
  - Intervalle
  - pointeur
- Le type doit être précisé par un identificateur, et non par une déclaration explicite.
- On déclare les fonctions au même niveau que les procédures (après VAR, TYPE, CONST)

ChapN: Les sous programmes

6.3. Les fonctions

Syntaxe de déclaration

- La définition d'une fonction comprend aussi trois parties :

Partie 1 : *L'entête*

- *Function Nom_fonc[(Ident1[,…]:Type1[;[var] Ident2[,…]:Type2])]:Typfonc;*

Partie 2 : Déclaration des variables locales

- *Nom_variable_i : type_variable_i;*

Partie 3 : *Corps de la fonction.*

```
Begin
    <instruction1>;
    <instruction2>;
       …
    nom_fonction:=<expression> {valeur retournée}
       …
     <instructionN>
End;
```

ChapN: Les sous programmes

6.3. Les fonctions

- L'affectation ***nom_fonction:=<expression>*** ne peut se faire qu'une seule fois par appel μa la fonction.
- La principale différence avec une procédure est que la fonction renvoie toujours une valeur (un résultat) comme après son appel, d'où la nécessité de déclarer un type pour cette fonction (Toute fonction *« vaut »* quelque chose.
- L'identificateur de fonction est donc considéré comme une expression, c'est-à-dire comme une valeur (calculée par la fonction elle-même).
- Ainsi, le simple fait d ‘écrire l'identificateur de fonction avec ses paramètres a pour conséquence d'appeler la fonction, d'effectuer le traitement contenu dans cette fonction, puis de placer le résultat dans l'identificateur de fonction.

6. 3. Les fonctions

Exemples

```
(*Ecrire un programme qui calcule la puissance d'un nombre réel.*)

program EXPOSANT ;
var
   N : integer;
   Y, T : real;
function PUISSANCE (X : real; N : integer) : real;
begin
     Y := 1;
     if N > 0 then
                  for I := 1 to N do
                      Y := Y * X
                else
                  for I := N to -1 do
                      Y := Y / X;
PUISSANCE := Y;   { affectation de la valeur à l'identificateur}
end;              { Fin du code de la fonction }
```

ChapN: Les sous programmes

6. 3. Les fonctions

Exemples

```
                        { Debut du programme principal}
begin
        readln (T, N); { appel de la fonction dans le writeln }
        writeln('La valeur de ', T,' puissance ', N,' est : ', PUISSANCE(T,N));
end. { Fin du programme }
```

ChapN: Les sous programmes

6.4. Différence entre procédure et fonction

- Une procédure peut être considérée comme une instruction composée que l'utilisateur aurait créée lui-même.
  - On peut la considérer comme un **petit programme**.
- Une fonction quant à elle renvoie **toujours une « valeur »**
  - comprendre le mot valeur comme objet de type simple, comme par exemple un entier ou un caractère.
- Elle nécessite donc un type (entier, caractère, booléen, réel, etc…).

NB : Il est interdit d'utiliser l'identificateur d'une fonction comme nom de variable en dehors du bloc correspondant à sa déclaration.

ChapN: Les sous programmes

6.4. Différence entre procédure et fonction

Exemples: *Examinons le programme suivant*

```
program exemple;
var x,y : integer;
function double (z : integer) : integer;
   begin
      double := z*2;
   end;
begin
      Readln(x);
      y := double(x);
      double := 8; {erreur µa cette ligne lors de la compilation}
end.
```

- *Ce programme ne pourra pas fonctionner, car on lui demande d'affecter la valeur 8 à la fonction* ***double****. Or, Il est interdit d'utiliser l'identificateur d'une fonction comme nom de variable en dehors du bloc correspondant à sa déclaration, d'où l'ERREUR.*

ChapN: Les sous programmes

6.4. Différence entre procédure et fonction

Exemples:

```
PROGRAM difference;
VAR a, b, c, d : real;
PROCEDURE Produit (x, y : real; var z : real);
 BEGIN
     z := x * y;
END;

Produit (a, b, c);
Produit (a-1, b+1, d);
```

```
FUNCTION Produit (x, y : real) : real;
VAR res : real;
BEGIN
      res := x * y;
       Produit:=res;
END;

BEGIN
   write ('a b ? '); readln (a, b);
          c := Produit (a, b);
          d := Produit (a-1, b+1);
    writeln ('c = ', c, ' d = ', d);
END.
```

ChapN: Les sous programmes

6.5. Variables globales et variables locales
6.5.1. Déclarations

- Lorsque la déclaration est effectuée en en-tête du programme, on parle de **variable globale**.
- Dans le cas contraire, où la déclaration est faite à l'intérieur même de la procédure ou de la fonction, on parle de **variable locale**.

6.5. Variables globales et variables locales
Exemple

*(*Reprenons le programme qui calcule la puissance d'un nombre réel. Nous désirons de plus calculer la puissance T^N avec des exposants (N) variant de -N µa +N, N étant le nombre entier entré par l'utilisateur. Par exemple, si on tape 3 pour T et 2 pour N, le programme calculera les valeurs 3^{-2}, 3^{-1}, 3^0, 3^1, et 3^2 Cette fois-ci, nous utiliserons des variables locales, ce qui est bien meilleur*)*

```
Program EXPOSANT ;
var I, N : integer;
      T : real;
Function PUISSANCE (X : real; N : integer) : real;
Var              { utilisation de variables locales }
   Y : real;
    I : integer;
Begin            { Code de la fonction }
     Y := 1;
     if N > 0 then
                  for I := 1 to N do
                     Y :=Y * X
               else
                  for I := -1 downto N do
                     Y :=Y / X;
       PUISSANCE :=Y;
end;             { Fin du code de la fonction }
```

ChapN: Les sous programmes

6.5. Variables globales et variables locales
Exemple

```
                                        { Debut du programme principal}
begin
      readln (T, N);
      for I := -N to N do
            writeln (PUISSANCE (T, I));   { appel de la fonction }
end.
```

ChapN: Les sous programmes

6.5. Variables globales et variables locales
6.5.2. Portée des variables

❖ **Variable locale :** portée limitée à la procédure ou à la fonction en question

❖ **Variable globale :** active dans le bloc principal du programme, mais également dans toutes les procédures et fonctions du programme.

❑ Chaque variable située dans un bloc de niveau supérieur est active dans toutes les procédures de niveau inférieur.

❑ Dans le cas où un même nom est utilisé pour une variable globale et pour une variable locale, cette dernière à la priorité dans la procédure ou dans la fonction où elle est déclarée (niveau local).

ATTENTION : Dans le cas où un même nom est utilisé pour une variable locale et pour une variable globale, le programme considère ces variables comme deux variables différentes (mais ayant le même nom).

6.5. Variables globales et variables locales
6.5.2. Portée des variables
Exemples

```
program PORTEE;
Var
    A, X : real;
procedure INTERNE;
Var
   B, X : real;
Begin
      B := A / 2;
      writeln (B);
      writeln (A + X);
end;
begin { programme principal }
     readln (A, X);
     writeln (A / 2);
      writeln(A + X);
       INTERNE;
end.
```

ChapN: Les sous programmes

6.5. Variables globales et variables locales
6.5.2. Portée des variables

Question : que se passe-t-il à l'exécution ?

Réponse : le programme ne fonctionnera pas comme on l'espérait, car X est considérée comme une variable locale dans INTERNE, et cette variable locale n'a pas été initialisée ! Dans la procédure INTERNE, X n'aura donc aucune valeur (il ne faut surtout pas la confondre avec le X du programme principal, qui est une variable globale).

- Pour que le programme fonctionne correctement, il faudra alors ne pas déclarer X comme locale dans INTERNE.

ChapN: Les sous programmes

6.5. Variables globales et variables locales
6.5.3. Locale ou globale ?

- Il vaut toujours mieux privilégier les variables locales aux variables globales.
- ***Inconvénients d'une utilisation systématique de variables globales :***
  - manque de lisibilité
  - présence de trop nombreuses variables au même niveau
  - ne facilite pas la récursivité
  - risque d'effets de bord si la procédure modifie les variables globales
- ***Avantages d'une utilisation de variables locales :***
  - meilleure structuration
  - diminution des erreurs de programmation
  - les variables locales servent de variables intermédiaires (tampon) et sont « oubliées »(effacées de la mémoire) à la sortie de la procédure.

ChapN: Les sous programmes

6.5. Variables globales et variables locales
6.5.3. Locale ou globale ?

- Définition

```
Procedure NIVEAU_SUP;
VAR X,Y, …

Procedure NIVEAU_INF;
var X,Z,…
Begin
      X:=…
      Y:=…
      Z:=…
end;
Begin
end.
```

6.5. Variables globales et variables locales
6.5.3. Locale ou globale ?

- La variable X déclarée dans NIVEAU_SUP est locale à cette procédure.
- Dans la procédure NIVEAU-INF, cette variable est occultée par l'autre variable X, déclarée encore plus localement.
- La portée de Y est celle de toute la procédure NIVEAU-SUP, sans restriction (elle est locale à NIVEAU-SUP, et donc globale dans NIVEAU-INF).
- En revanche, Z a une portée limitée à la procédure NIVEAU-INF (elle est locale à NIVEAU-INF, et ne peut être utilisée dans NIVEAU-SUP).

ChapN: Les sous programmes

6.6. Les paramètres
6.6.1. Définition
Problématique: Exemple

- Supposons qu'on ait à résoudre l 'équation du second degré en divers points du programme :
  - la première fois $Rx^2 + Sx + T = 0$,
  - la deuxième fois $Mx^2 + Nx + P = 0$,
  - la troisième fois $Ux^2 + V x + W = 0$.

- Comment faire en sorte qu'une **même procédure** puisse les traiter toutes les trois, c'est-à-dire **travailler sur des données différentes** ?

ChapN: Les sous programmes

6.6. Les paramètres
6.6.1. Définition
Problématique: Exemple

- La1ère possibilité : utiliser des variables globales A, B, C pour exprimer les instructions dans la procédure, et, avant l'appel de la procédure, faire exécuter des instructions telles que :
  - A := R, B := S, C := T, etc.
  - problème des variables globales, multiplication des affectations, etc … . Solution à écarter !
- 2ième possibilité : définir une procédure de résolution d'une équation du second degré avec une liste d'arguments, et transmettre les données à l'aide de paramètres.
- La déclarations sera :
  - procedure SECOND_DEGRE(A,B,C:integer); à l'appel, on écrira :
  - SECOND_DEGRE (M, N, P); { *appel avec les valeurs de M, N et P* }
  - SECOND_DEGRE (R, S, T); { *appel avec les valeurs de R, S et T* }

ChapN: Les sous programmes

6.6. Les paramètres
6.6.1. Définition

- Lors de la déclaration de la procédure, donner une liste d'arguments (paramètres) signifie une transmission de l'information entre le programme principal et le sous-programme.
- Il faut préciser le type des arguments par un identificateur et par une déclaration de type.

Exemple

```
Procedure OK (x,y : real);    Déclaration correcte
préférer : Procedure OK (tab : tableau);
à :
Procedure OK (tab : array[1..100] of integer);
```

ChapN: Les sous programmes

6.6. Les paramètres
6.6.2. paramètres formels et paramètres effectifs

- Dans la déclaration de la procédure ***SECOND_DEGRE*** , les paramètres A, B et C sont appelés ***paramètres formels*** dans la mesure où ils ne possèdent pas encore de valeurs.
- Lors de l'utilisation de la procédure dans le programme principal (donc à l'appel) M, N et P sont des ***paramètres effectifs*** (ou réels).
  - Lors de l'appel de la procédure, il y a remplacement de chaque paramètre formel par un paramètre effectif, bien spécifié.
- Soit la déclaration de procédure suivante :

Procedure SECOND_DEGRE (A, B, C : integer);

- *Lors de l'appel de la procédure, si on écrit :*

SECOND_DEGRE (M, N, P);

- *A l'entrée du bloc de la procédure, A prendra la valeur de M, B prendra celle de N et la variable C sera affectée de la valeur de P.*

ChapN: Les sous programmes

6.6. Les paramètres
6.6.2. paramètres formels et paramètres effectifs

- Dans une procédure, le nombre de paramètres formels est exactement égal au nombre de paramètres effectifs. De même à chaque paramètre formel doit correspondre un paramètre effectif de **même type**.
- Un paramètre formel est toujours une variable locale et ne peut être utilisé en dehors de la procédure (ou de la fonction) où il est défini. En revanche un paramètre effectif peut être une variable globale.
- En Pascal, lors de la déclaration d'une procédure, il est possible de choisir entre deux modes de transmission de paramètres :
  - *passage par valeur* et
  - *passage par adresse* ;

ChapN: Les sous programmes

6.6. Les paramètres
6.6.3. passage de paramètre par valeur

- Rappelons d'abord que :
- Déclarer une variable de nom X → réserver un emplacement mémoire pour X.
- Ainsi lors de l'appel d'une procédure un emplacement mémoire est réservé pour chaque paramètre formel. De même un emplacement mémoire est aussi réservé pour chaque paramètre effectif lors de sa déclaration.
- Cependant lors de l'appel, ***les valeurs des paramètres effectifs sont copiées dans les paramètres formels***.

ChapN: Les sous programmes

6.6. Les paramètres
6.6.3. passage de paramètre par valeur

- Ainsi l'exécution des instructions de la procédure se fait avec les valeurs des paramètres formels et toute modification sur ces dernières ne peut affecter en aucun cas celles paramètres effectifs.
- Dans ce cas on parle de *passage par valeur*.
  - C'est le cas dans la procédure *SAISIE* (paragraphe 5.3.1) du paramètre *dim* et dans la procédure *AFFICHAGE* (paragraphe 5.3.1) des paramètres *M* et *dim*.

NB: dans ce type de passage, les valeurs des paramètres effectifs ***sont connues*** avant le début de l'exécution de la procédure et jouent le rôle uniquement d'entrées de la procédure.

Exemple:

procedure ID_PROC (X,Y : real; I : integer; TEST: boolean);

ChapN: Les sous programmes

6.6. Les paramètres
6.6.3. passage de paramètre par valeur

Points importants :

•Les paramètres formels sont des variables locales à la procédure, qui reçoivent comme valeur initiale celles passées lors de l'appel On parle alors d'**AFFECTATION**.

Exemple **: ID_PROC (A, B, 5, true); X** a alors pour valeur initiale celle de **A**, *Y* celle de **B**. **I** a pour valeur initiale **5**, et **TEST true**

•Le traitement effectué dans la procédure, quel qu'il soit, ne pourra modifier la valeur des paramètres effectifs. Exemple : après exécution de **ID_PROC**, **A** et **B** auront toujours la même valeur qu'auparavant, même s'il y a eu des changements pour ces variables, dans la procédures.

•Le paramètre spécifié lors de l'appel peut être une expression.

Ainsi**, ID_PROC (3 / 5, 7 div E, trunc (P), true) ;** est un appel correct.

6.6. Les paramètres
6.6.3. passage de paramètre par valeur

Exemple:

```
(*Nous cherchons à écrire un programme qui échange les valeurs de deux variables saisies par l'utilisateur.*)

program TEST;
var
   A, B : real;
procedure ECHANGE (X,Y : REAL);
var
   T : real;
begin
      T := X;
      X := Y;
      Y := T;
       Writeln(X,Y);
end;
Begin
       readln (A, B);
       ECHANGE (A, B);
       writeln (A, B);
End.
```

ChapN: Les sous programmes

6.6. Les paramètres
6.6.3. passage de paramètre par valeur

- *Cette solution est mauvaise. En effet, une simulation de son exécution donnera :*

A = 5 B = 7 (saisie)

X = 7 Y = 5

A = 5 B = 7 (A et B restent inchangés !)

- *Le résultat de l'action accomplie par la procédure n'est pas transmis au programme appelant, donc **A** et **B** ne sont pas modifiés.*
- ***La Transmission est unilatérale.***

Avantage : grande liberté d'expression dans l 'écriture des paramètres effectifs.

- Cela évite les erreurs et les effets de bord (que nous verrons plus loin).
- Tout paramètre est utilisé, pour passer des valeurs dans la procédure, mais n'est jamais modifié.
- Si on désire récupérer l 'éventuelle modification du paramètre, il faut alors utiliser un autre procédé, décrit dans la section suivante (passage par adresse).

ChapN: Les sous programmes

6.6. Les paramètres
6.6.3. passage de paramètre par valeur

Exemple: *Ceci est un programme qui ne fait pas grand chose et dont le seul intérêt est d'illustrer le passage de paramètres ainsi que la notion d'effet de bord.*

```
program Effet-de_bord;
var
   i,j : integer;
function double (i : integer) : integer;
  begin
     double := 2 * i;
   end;
function plus_un (j : integer) : integer;
   Var
     i: integer;
     begin
        i := 10;
        plus_un := j + 1;
      end;
```

```
function moins_un (j : integer) : integer;
   begin
      i := 10;
      moins_un := j - 1;
    end;
Begin      {programme principal}
   i := 0; j := 0;       {i=0 ; j=0}
   j := plus_un(i);      {i=0 ; j=1}
   j := double(j);       {i=0 ; j=2}
   j := moins_un(j);     {i=10 ; j=1}
End.
```

ChapN: Les sous programmes

6.6. Les paramètres
6.6.3. passage de paramètre par valeur

- *La variable i a été modifiée dans la procédure, alors que l'on s'attendait à des modifications sur j. On appelle cela un effet de bord. Un tel phénomène peut être très dangereux (difficulté à retrouver les erreurs).*
- Effet de bord Voici le scénario catastrophe à éviter
  - On est dans une procédure P et on veut modifier une variable x locale à P.
  - Il existe déjà une variable globale ayant le même nom x.
  - On oublie de déclarer la variable locale x au niveau de P
  - A la compilation tout va bien !
  - A l'exécution, P modifie le x global alors que le programmeur ne l'avait pas voulu.
  - Conséquence : le programme ne fait pas ce qu'on voulait, le x global a l'air de changer de valeur tout seul !

→ Erreur très difficile à détecter ; être très rigoureux et prudent !

ChapN: Les sous programmes

6.6. Les paramètres
6.6.4. passage de paramètre par adresse

- En pascal, la différence principale entre le passage par valeur et le passage par adresse c'est que dans ce dernier, ***un seul emplacement mémoire*** est réservé pour le paramètre formel et le paramètre effectif correspondant.
- Autrement dit, dans ce cas chaque paramètre formel de la procédure utilise directement l'emplacement mémoire du paramètre effectif correspondant.
  - Par conséquent toute modification du paramètre formel entraîne la même modification du paramètre effectif correspondant.
- Soulignons que dans ce mode de passage de paramètres, les valeurs des paramètres effectifs peuvent être ***inconnues*** au début de l'exécution de la procédure.
- Cependant un paramètre formel utilisant ce type de passage ne peut être que le résultat de la procédure.

ChapN: Les sous programmes

6.6. Les paramètres
6.6.4. passage de paramètre par adresse

- Pour spécifier dans une procédure qu'il s'agit du mode par adresse, il suffit, lors de la déclaration de la procédure, d'ajouter le mot clé ***VAR*** devant la déclaration du paramètre concerné.
- C'est le cas par exemple du paramètre ***S*** dans la procédure ***SOMME*** de l'exemple du paragraphe 6.3.1.

Exemple

```
procedure ID_PROC (var X,Y : real; Z : integer);
```

ChapN: Les sous programmes

6.6. Les paramètres
6.6.4. passage de paramètre par adresse

Points importants :

- Lors de l'appel, des paramètres réels sont substitués aux paramètres formels :

SUBSTITUTION

- Seule une variable peut être substituée aux paramètres réels, il est impossible de faire l'appel avec une constante ou une expression évaluable !

Exemple

ID_PROC (U,V, 7); est correct.

ID_PROC (4, A - B, 8); est tout µa fait incorrect.

- Tout changement sur le paramètre formel variable change aussi le paramètre effectif spécifié lors de l'appel.

ChapN: Les sous programmes

6.6. Les paramètres
6.6.4. passage de paramètre par adresse

Exemple:

```
program essai;
Var
   i : integer;
procedure double (x : integer ; var res : integer);
begin
      res := 2 * x;
end;
Begin
      i := 1;            {i=1}
      double (5,i);      {i=10}
End.
```

Dans cet exemple, x est un paramètre transmis par valeur (donc non variable), alors que res est un paramètre passé par adresse (donc considéré comme variable).

Principe :

s'il y a nécessité de renvoyer les modifications au programme appelant, on emploi des paramètres variables.

6.6. Les paramètres
6.6.4. passage de paramètre par adresse

Exemple:

```
(*Reprenons et corrigeons le programme qui échange les valeurs
de deux variables saisies par l'utilisateur.*)
program TEST_BIS;
var
   A, B : real;
procedure ECHANGE (var X,Y : real);
var
   T : real;
begin
    T := X;
    X := Y;
    Y := T;
     writeln(X,Y);
end;
```

```
begin
    readln (A, B);
    ECHANGE (A, B);
    writeln (A, B);
end.
```

Une simulation du déroulement du programme donnera :

A = 5 B = 7 (saisie)

X = 7 Y = 5

A = 7 B = 5 (A et B ont été modifiés !)

Le résultat de l'action de la procédure a été transmis au programme appelant !