Ecole Supérieure
des Techniques Industrielles
et des Textiles

52, allée Lakanal - BP 209
59654 Villeuneuve d'Ascq Cedex
Tél 33 (0)3.20.79.90.10
Fax 33 (0)3.20.79.90.11

# TURBO PASCAL

*Eléments de base*

Pré Requis : connaissance de l'analyse structurée

# SOMMAIRE

**1. INTRODUCTION ..... 4**

1.1. HISTORIQUE ..... 4
1.2. PROGRAMMER EN PASCAL ..... 4
1.3. STRUCTURE D'UN PROGRAMME EN PASCAL ..... 5
1.4. TYPES DE DONNEES ELEMENTAIRES ..... 7
1.5. CONSTANTES ..... 8
1.6. VARIABLES ..... 8
1.7. PRESENTATION D'UN PROGRAMME EN PASCAL ..... 9

**2. ENTREES & SORTIES ..... 10**

2.1. SORTIE ..... 10
2.2. ENTREE ..... 11
2.3. LECTURE DIRECTE DU CLAVIER ..... 11

**3. LES OPERATEURS ..... 12**

3.1. OPERATEURS ARITHMETIQUES ..... 12
3.2. OPERATEURS DE COMPARAISON ..... 12
3.3. OPERATEURS LOGIQUES ..... 13
3.4. OPERATEURS DE BITS ..... 13

**4. STRUCTURE DE CONTROLE ..... 14**

4.1. INSTRUCTIONS ALTERNATIVES (TESTS & CHOIX) ..... 14
4.1.1. INSTRUCTION « IF…THEN… ; » ..... 14
4.1.2. INSTRUCTION « IF … THEN…ELSE… ; » ..... 15
4.1.3. INSTRUCTION « CASE…OF…ELSE…END ; » ..... 16
4.2. INSTRUCTIONS REPETITIVES (ITERATIONS) ..... 18
4.2.1. INSTRUCTION « WHILE…DO… ; » ..... 18
4.2.2. INSTRUCTION « FOR…DO…» ..... 18
4.2.3. INSTRUCTION « REPEAT … UNTIL ... ; » ..... 20
4.3. INSTRUCTIONS DE BRANCHEMENT ..... 21

**5. TYPES DE DONNEES COMPLEXES ..... 22**

5.1. TABLEAUX ..... 22
5.1.1. TABLEAUX A UNE DIMENSION ..... 22
5.1.2. TABLEAUX A PLUSIEURS DIMENSIONS ..... 23
5.1.3. STRING ( CHAINES DE CARACTERES) ..... 24
5.2. ENREGISTREMENTS ..... 26
5.2.1. DECLARATION DES ENREGISTREMENTS ..... 26
5.2.2. OPERATIONS SUR LES VARIABLES STRUCTUREES ..... 26

**6. CONSTANTES & TYPES ........ 27**

6.1. DECLARATIONS DES CONSTANTES ........ 27
6.2. DECLARATIONS DE TYPES ........ 27

**7. VARIABLES LOCALES ET VARIABLES GLOBALES ........ 28**

**8. PROCEDURES & FONCTIONS ........ 29**

8.1. DEFINITION DES PROCEDURES ET DES FONCTIONS ........ 29
8.2. DECLARATION GLOBALES DES PROCEDURE ET DES FONCTIONS ........ 34
8.3. PASSAGE DE PARAMETRES ........ 35
8.3.1. PASSAGE PAR VALEUR (CALL BY VALUE) ........ 35
8.3.2. PASSAGE PAR ADRESSE (CALL BY REFERENCE) ........ 36
8.4. FONCTION RECURSIVES ........ 37

**9. GESTION DE FICHIERS ........ 38**

**10. ANNEXES ........ 40**

10.1. ARTICULATION D'UN PROGRAMME ........ 40
10.2. PROCEDURE ........ 40
10.3. FONCTION ........ 40
10.4. PARAMETRES DES PROCEDURES ET FONCTIONS ........ 41
10.5. EXEMPLE DE PROGRAMME ........ 41
10.6. MOTS RESERVES ........ 42
10.7. STRUCTURE DE DONNEES ........ 43
10.8. TYPE DE VARIABLES ........ 43
10.9. EXEMPLE DE DECLARATION DES CONSTANTES, VARIABLES ET TYPES ........ 44
10.10. TABLE ASCII STANDARD (7 BITS) ........ 45
10.11. TABLE ASCII (SECONDE MOITIE) ETENDUE (8 BITS) ........ 46
NOTES PERSONNELLES ........ 47

# 1. Introduction

## 1.1. Historique

Le langage de programmation Pascal est un langage conçu au début des années 1970 par N. Wirth. Depuis, l'utilisation de ce langage s'est développée dans les universités et la communauté scientifique. Son succès, toujours croissant, a montré qu'il s'agit du langage qui, durant les années 1980, a détrôné les langages tels que FORTRAN, les différents dérivés d'ALGOL, de PL/I... Le Pascal est facile à enseigner et à apprendre : il permet d'écrire des programmes très lisibles et structurés, il dispose entre autres de facilités de manipulation de données.

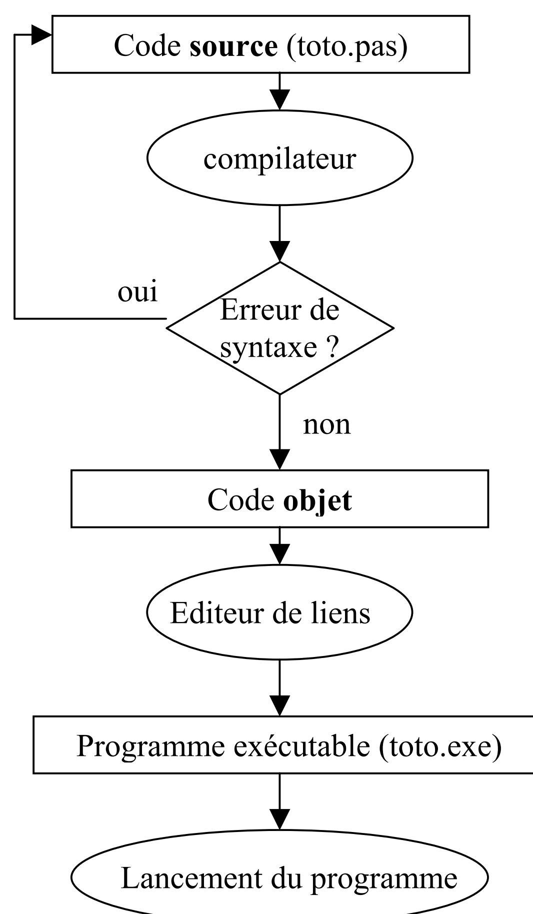
## 1.2. Programmer en Pascal

- **Programme**

Un programme est une suite d'instructions destinées à l'ordinateur. Or le langage de l'ordinateur est un langage machine qui n'utilise que deux symboles : 0 et 1. On utilise donc un langage de programmation, ici le langage Pascal, permettant de produire des programmes lisibles et facilement modifiables. Ces programmes sont traduits en langage machine à l'aide d'un compilateur.

- **Code source**

Un programme en Pascal (toto par exemple) peut être écrit avec un traitement de texte, un éditeur de texte ou directement dans le logiciel de programmation édité par Borland. Le programme ainsi réalisé est stocké sous forme de fichier avec l'extension .pas (toto.pas).




- **Compilation et édition de liens**

La version en langage machine d'un programme s'appelle aussi le code objet du programme.

L'éditeur de lien (linker) est un programme auxiliaire qui intègre, après la compilation du fichier source, le code machine de toutes les fonctions utilisées dans le programme et non définies à l'intérieur. A partir du code objet du programme, l'éditeur de liens génère un fichier exécutable d'extension .exe (toto.exe). Le fichier .exe renferme alors le programme exécutable qui peut être chargé dans la mémoire pour être exécuté.

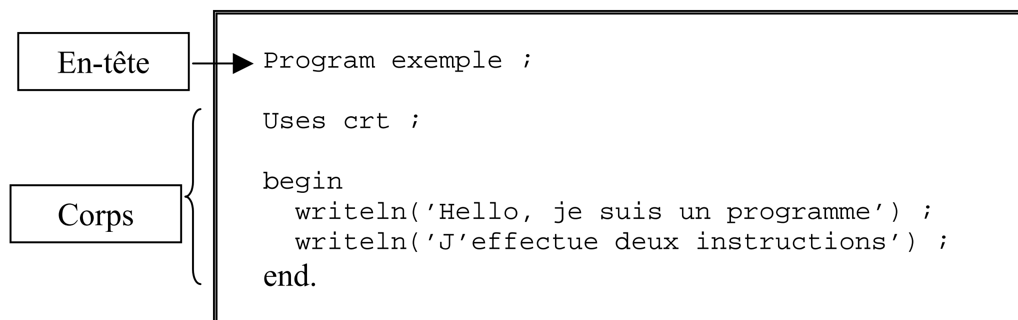
# 1.3. Structure d'un programme en Pascal

Un programme est composé d'une suite d'instructions qui peuvent être assimilées à des phrases du langage. L'étude du langage consiste à apprendre les règles de formation de ces instructions. La structure générale d'un programme en Pascal peut se définir de la façon suivante :

- Une partie en-tête qui contient le nom du programme et ses paramètres éventuels,
- Le corps du programme qui contient des déclarations et une partie instructions

Il existe un ordre prédéfini pour introduire les éléments du corps du programme.

Exemple de programme :




- **Uses**

Les informations dont le compilateur a besoin pour les fonctions prédéfinies ( writeln, …) se trouvent dans des fichiers spéciaux (appelés Unité). Ces fichiers sont inclus dans le fichier source via la commande Uses :

Uses Crt ;

Ces unités sont organisées de façon thématique :

| | |
|---|---|
| dos : | contient les déclarations des fonctions systèmes, |
| graph : | contient les déclarations des fonctions graphiques, |

- **Blocs d'un programme**

Un « **begin** » et un « **end** » constitue un bloc. Un programme en Pascal peut contenir un nombre quelconque de blocs. Ceux-ci renferment toutes sortes d'instructions. Ces blocs peuvent être aussi inclus les uns dans les autres.

- **Fonctions et procédures / programme principal**

Dans les déclarations, on trouve des procédures et des fonctions. Celles-ci sont une portion de programme chargée d'exécuter une tâche spécifique. Chaque programme peut contenir plusieurs fonctions et procédures. Mais tout programme contient au minimum le programme principal dans le corps. Ce programme principal commence par « **begin** » et se termine par « **end.** »

- **Instructions**

Les instructions en Pascal se terminent par un point virgule. Chaque instruction est exécutée l'une à la suite de l'autre. La première instruction exécutée est celle qui se trouve la première dans le programme principal.

- **Commentaires**

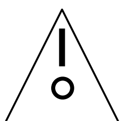
Il est toujours préférable qu'un programme contienne des explications aux endroits complexes afin d'en faciliter la compréhension. En Pascal, on insère ces explications dans le programme : ce sont les commentaires. Ils sont délimités par { et } ou alors par (* et *).

```
Program exemple2 ;

Uses crt ;   { je suis un commentaire }

Begin
  { je suis un autre commentaire mais
     sur deux lignes }
  instruction_1 ;
  ...
  instruction_n ;
end.
```

- **Jeu de caractères du Pascal**

  - Lettres majuscules : ABCDEFGHIJKLMNOPQRSTUVWXYZ
  - Lettres minuscules : abcdefghijklmnopqrstuvwxyz
  - Chiffres : 0123456789
  - Caractère de soulignement (underscore) : _
  - Caractères non visibles : espace,
  - Caractères spéciaux et ponctuation :

| | | | |
|---|---|---|---|
| , | virgule | + | signe plus |
| . | point | * | étoile |
| ; | point-virgule | - | signe moins |
| : | double point | < | inférieur |
| ? | point d'interrogation | > | supérieur |
| ' | apostrophe | = | signe égal |
| " | guillemet | { | accolade ouvrante |
| ( | parenthèse gauche | } | accolade fermante |
| ) | parenthèse droite | | |
| [ | crochet ouvrant | | |
| ] | crochet fermant | | |



Le compilateur Pascal n'est pas sensible à la casse ! Il ne fait pas la distinction entre minuscule et majuscule. L'écriture **begin** est correcte ainsi que les écritures suivantes : **BEGIN** ou **Begin**

- **Identificateurs**

Pour nommer les constantes, les variables ou les fonctions, on utilise une séquence de lettres ou de chiffres (commençant toujours par une lettre ou _ ). ATTENTION, seuls les 32 premiers éléments sont significatifs !

Attention aux mots réservés :

| | | | | |
|---|---|---|---|---|
| absolute | and | array | asm | assembler |
| begin | case | const | constructor | destructor |
| div | do | downto | else | end |
| export | exports | external | far | file |
| for | forward | function | goto | if |
| implementation | in | index | inherited | inline |
| interface | interrupt | label | library | mod |
| near | nil | not | object | of |
| or | packed | private | procedure | program |
| public | record | repeat | resident | set |
| shl | shr | string | then | to |
| type | unit | until | uses | var |
| while | with | xor | virtual | |

# 1.4. Types de données élémentaires

En Pascal, on trouve les types de données suivants :

```
❑ byte :         0..255                    (8 bits non signé)
❑ shortint:      -128..127                 (8 bits signé)
❑ word:          0..65535                  (16 bits non signé)
❑ integer:       -32768..32767             (16 bits signé)
❑ longint:       -2147483648..2147486647  (32 bits signé)
❑ real:          2.9e-39..1.7e38           (64 bits signé) 6 chiffres signif
❑ double         5.0e-324..1.7e308                   15 chiffres signif
❑ boolean :      true ou False
❑ file...of :    fichier de...
❑ string :       chaîne de caractère (maximum 255)
❑ string[num] :  chaîne de caractère de longueur num
❑ char :         caractère (les 256 caractères ASCII)
❑ pointer :      pointeur ;
```

- **Type char (caractère)**

Le type char est utilisé pour représenter un caractère de l'ensemble des caractères représentables. La table de caractère est la table ASCII (American Standard Code for Information Interchange). Un caractère occupe un octet car il faut coder les 256 signes du jeu (étendu) des caractères ASCII.

- **Types de données réelles**

Les nombres décimaux (nombres réels) en Pascal sont dits à virgule flottante ; c'est à dire des nombres dans lesquels la virgule en tant que séparateur entre la partie entière et décimale n'est pas figée. La grandeur d'un nombre réel est donnée par un exposant adéquat. Les nombres à virgule flottante sont des valeurs approchées. La précision dépend de la machine. Le type ***real*** garantie une précision d'au moins 6 chiffres après la virgule, le type ***double*** 15 chiffres

# 1.5. Constantes

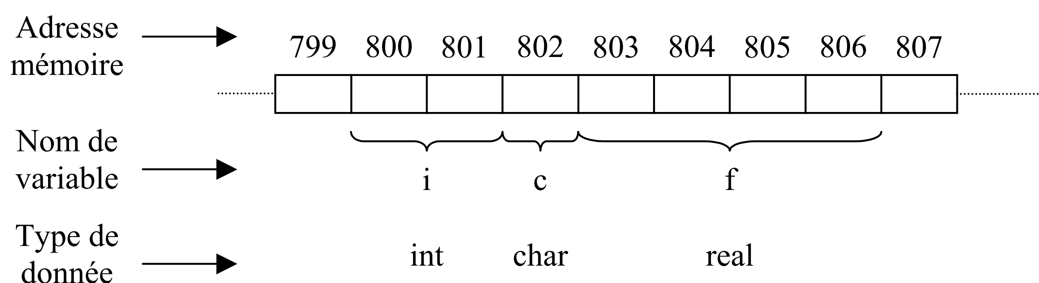
Il existe 4 types de constantes : entière, à virgule flottante, de type caractère et de type chaîne.

Exemple :

```
Const
     Euro = 6.55957 ;
     Com1 = $3f8 ;    {entier exprimé en valeur hexadécimale }
     Base = 100 ;
     Touche = 'a' ;
     Message = 'My name is Bond, James Bond !'
```

# 1.6. Variables

Les variables se différencient des constantes par le fait qu'on peut leur affecter des valeurs qui peuvent être modifiées pendant l'exécution du programme. La mémoire d'un ordinateur est similaire à une suite de « cellules » occupant chacune un octet. Les cellules mémoire ont des numéros croissants dont le premier vaut 0. Ces numéros sont les adresses des cellules mémoire. L'adresse à partir de laquelle une donnée est rangée en mémoire est l'adresse de l'objet, plus précisément l'adresse du début de la donnée :




- **Définition des variables**

Il faut définir la variable avant qu'elle ne soit utilisée dans le programme. Une telle définition détermine le nom et le type de la variable et lui réserve de l'espace mémoire conformément à son type.

Exemple :

```
var
    a, b, c : REAL ; { définit 3 variables de type réel }
    x, n, toto : INTEGER ; { définit 3 variables de type entier }
```

**ATTENTION** :

Un nom est une suite d'un ou de plusieurs caractères qui peuvent être des lettres, des chiffres ou le caractère de soulignement (_), avec la restriction que le nom ne doit pas commencer par un chiffre. Les noms peuvent être de longueur quelconque, mais le compilateur habituellement ne tient compte que des 32 premiers caractères.

- **Affectation**

L'affectation d'une variable se fait au moyen de l'opérateur symbolisé par **:=**

```
c := 'a' ;
```

Exemple avec des variables :

```
    var   c : char ;        {  Définition d'une variable char }
          i, j : integer ;  {  Définition de deux variables int }
          f : real ;        {  Définition d'une variable float }

begin
  c := 'Z' ;    { c reçoit le caractère 'Z' dont le code ASCII est 90 }
  i := 1 ;      { la variable i prend la valeur 1 }
  c := 'A' ;    { c prend la nouvelle valeur : le caractère 'A' }
  j := i ;      { la variable j reçoit la valeur de la variable i donc 1 }
  f := -3.14 ;  { la variable f prend comme valeur -3.14 }
end.
```

Les variables définies dans cet exemple sont **globales** : elles sont utilisables dans tout le programme. Contrairement aux variables globales, les variables locales, définies dans une fonction ou une procédure, ne peuvent être utilisées que dans le bloc **begin...end** où elles ont été définies.

# 1.7. Présentation d'un programme en Pascal

```
program ... ;  { en-tête du programme}*

uses ... ;     { Utilisation des unités / bibliothèques }*
const ... ;    { Déclaration des constantes }*
type ... ;     { Déclaration des types }*
var  ... ;     { Déclaration des variables}*

procedure ... ;     { définition des procédures }*
function  ... ;     { définition des fonctions }*

begin               { programme principal : OBLIGATOIRE }
  instruction ;
  ...
end. {fin de programme }
```

* : facultatif

# 2. Entrées & sorties

Pour transmettre des données saisies au clavier à un programme (entrées) ou pour afficher à l'écran les données par un programme (sorties), il faut faire appel à un ensemble de fonctions appartenant à l'unité d'entrée-sortie. Il faut donc faire apparaître en début de programme l'instruction suivante :

```
Uses crt ;
```

## 2.1. Sortie

- **Définitions :**

- On utilise la fonction **Write** ou **Writeln** pour l'affichage formaté des données,
- Formaté signifie qu'on contrôle la forme et le format des données affichées,
- La fonction admet la syntaxe suivante :

```
   write(argument_1, argument_2,…, argument_n) ;
ou
   writeln(argument_1, argument_2,…, argument_n) ;
```

avec **argument_1, …, argument_n :** les arguments à afficher.

Exemples :

```
Write('Bonjour') ;
Writeln(' cher ami') ;
a := 2+3 ;
Writeln('La somme 2 + 3 donne ',a) ;
```

La fonction **write** écrit ici à l'écran les arguments (chaîne de caractère, constante, variables). La fonction **writeln** exécute la même chose. La seule différence est que, à la fin de l'écriture du dernier argument, il y a passage à la ligne suivante

- **Largeur d'affichage**

La largeur d'une valeur est le nombre de caractères ou de chiffres qui seront affichés. Pour spécifier la largeur, on place un nombre entier suivant la variable à afficher séparé par un double point

Remarques : Si la représentation d'une valeur nécessite moins de position qu'en indique la valeur, celles-ci seront complétées par des espaces.

Exemples :

```
                            Résultat de l'affichage :
pi := 3.14159 ;
writeln(pi);                3.141590000E+00
writeln(pi:20);                3.141590000E+00
writeln(pi:0);              3.1E+00
```

- **Précision**

Comme pour la largeur d'affichage, on peut spécifier la précision des valeurs à afficher. La spécification de précision est un nombre entier qui suit la largeur minimale et est séparé de cette dernière par un double point. Pour les nombres réels, il indique le nombre de chiffres après la virgule.

Exemples :

```
                                  Résultat de l'affichage :
pi := 3.14159 ;
writeln(pi);                      3.141590000E+00
writeln(pi:20:5);                              3.14159
writeln(pi:0:4);                  3.1416
```

## 2.2. Entrée

- **Définitions :**

  - On utilise la fonction **readln** pour la saisie des données depuis le clavier,
  - La fonction **readln** admet la syntaxe suivante :

**`readln(argument_1, argument_2,…, argument_n) ;`**

avec **argument_1, …, argument_n :** les arguments à afficher.

- **Exemple :**

```
Write('Entrez un nombre entier : ') ;
Readln(a) ;
Writeln('vous avez entré la nombre ',a) ;
Write('Entrez 3 nombre réels : ') ;
Readln(b,c,d) ;
```

La fonction **readln** lit des valeurs sur le périphérique d'entrée standard (clavier), les interprète dans le format de la variable et les range dans les arguments spécifiés. A chaque valeur saisie, il faut valider par la touche **entrée** pour que la saisie soit prise en compte.

## 2.3. Lecture directe du clavier

- **Définitions**

Il existe une fonction avec laquelle on peut entrer des données sans valider par la touche **entrée**. Cet entrée manipule uniquement des caractères. Il faut donc déclarer le type CHAR

Exemples :

```
C := readkey ;           { lit une touche au clavier }
C := upcase(readkey) ;   { lit une touche au clavier et si
                           c'est une minuscule, elle sera
                           convertie en majuscule }
```

# 3. Les opérateurs

- **Le Pascal dispose d'opérateurs classifiés en deux types :**

  - les opérateurs unaires (unary operators) admettent un unique opérande,
  - les opérateurs binaires (binary operators) possèdent deux opérandes,

- **Les opérateurs existants sont :**

| | |
|---|---|
| opérateurs arithmétiques | opérateur conditionnel |
| opérateurs de comparaison | opérateur parenthèse |
| opérateurs logiques | opérateur de taille |
| opérateurs de bits | opérateur d'adresse |
| opérateurs d'affectation | opérateur de pointeur |
| opérateurs de chaînes | opérateurs relationnels |

- **Priorités**

Les opérateurs du Pascal sont classés selon des niveaux de priorité, la plus haute priorité valant 1. « Priorité plus haute » signifie une exécution prioritaire de l'opérateur.

## 3.1. Opérateurs arithmétiques

Ces opérateurs procèdent à des opérations arithmétiques sur leurs opérandes.

- **Les opérateurs arithmétiques sont :**

| | Opérateur | Signification | exemple |
|---|---|---|---|
| **Binaires** | + | Addition | a+b |
| | - | Soustraction | a-b |
| | * | Multiplication | a*b |
| | / | Division | a/b |
| | div | Division entière | a div b |
| | mod | Modulo (reste) | a mod b |
| **Unaire** | - | Négation | -a |

exemples : 2+4 2/4 2 mod 4 (1*2)-(3*4) -x 7 div 3

Remarque : Il n'existe pas d'opérateur puissance en Pascal

## 3.2. Opérateurs de comparaison

Les opérateurs de comparaison sont des opérateurs binaires et comparent la valeur de leurs opérandes. Les opérandes n'ont pas besoin d'être du même type. Le résultat de la comparaison donne une valeur booléenne correspondant à la valeur logique **FALSE** (faux) ou alors **TRUE** (vrai)

- **Les opérateurs de comparaison :**

| Opérateur | Signification | Exemple |
|---|---|---|
| = | Opérande 1 **égal** à opérande 2 ? | a=b |
| <> | Opérande 1 **différent** de opérande 2 ? | a <> b |
| <= | Opérande 1 **inférieur ou égal** à opérande 2 ? | a<=b |
| >= | Opérande 1 **supérieur ou égal** à opérande 2 ? | a>=b |
| < | Opérande 1 **inférieur** à opérande 2 ? | a<b |
| > | Opérande 1 **supérieur** à opérande 2 ? | a>b |

# 3.3. Opérateurs logiques

Les opérateurs logiques effectuent les opérations classiques de la logique booléenne. L'évaluation des expressions comportant des opérateurs logiques donne un résultat de type True ou False.

- **Les opérateurs logiques :**

| | Opérateur | Signification | Exemple |
|---|---|---|---|
| Binaire | and | ET | (x>2)and(x<10) |
| | or | OU | (y>0) and (c='a') |
| | xor | OU exclusif | (t=0) xor (u=3) |
| Unaire | not | NON logique | not (fini) |

exemple : L'expression (5<7)and(3>2)a la valeur True.

# 3.4. Opérateurs de bits

Les opérateurs de bits exécutent des opérations logiques ET, OU, OUex, NON et des opérations de décalage sur tous les bits , pris un à un, de leurs opérandes entiers.

- **Les opérateurs logiques de bits :**

| | Opérateur | Signification | Exemple |
|---|---|---|---|
| Binaire | and | ET | 12 and 43 donne 8 |
| | or | OU | 12 or 43 donne 47 |
| | xor | OUex | 12 xor 43 donne 39 |
| Unaire | not | NON logique | Not(12) donne 245 |

- **Les opérateurs de décalage de bits :**

| | Opérateur | Signification | Exemple |
|---|---|---|---|
| Binaire | shr | Décalage de y bits vers la droite | 8 shr 3 donne 1 |
| | shl | Décalage de y bits vers la gauche | 2 shl 9 donne 1024 |

# 4. Structure de contrôle

Le Pascal dispose de 3 groupes de structures de contrôle de flux. Il s'agit d'instructions par lesquelles on peut contrôler le déroulement d'un programme :

- les instructions alternatives (tests et choix)
- les instructions répétitives (itérations)
- les instructions de branchement.

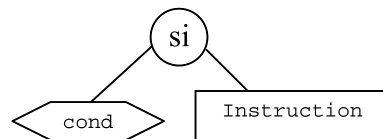
## 4.1. Instructions alternatives (tests & choix)

Il existe 3 instructions de tests qui permettent de ne pas exécuter systématiquement certaines instructions mais seulement dans certains cas prévus par le programmeur. Ces instructions sont :

- **if...then...** ; : test
- **if...then...else** ... ; : test + alternative
- **case...of...else...end ;** : tests multiples

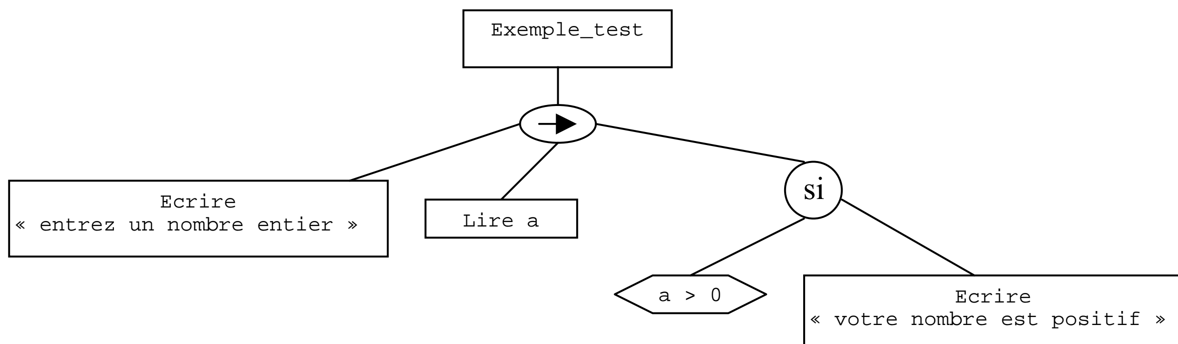
### 4.1.1. Instruction « if…then… ; »

L'instruction ne s'exécute que si la condition est **Vraie**.
La syntaxe est :




```
if expression_logique
  then instruction ;
```

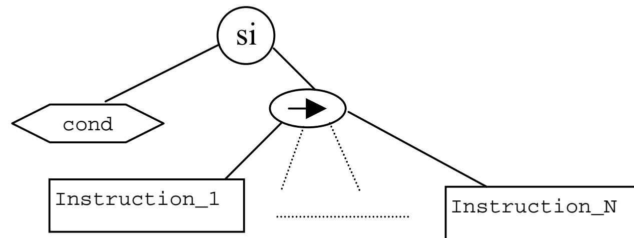
exemple :




```
Program exemple_test ;

Uses Crt ;

Var a : INTEGER ;

begin
  write('entrez un nombre entier : ') ;
  readln(a) ;
  if a > 0
    then writeln('Votre nombre est positif') ;
end.
```

Il peut y avoir plusieurs instructions exécutées si le test est Vrai. ***Then*** sera alors suivi d'un bloc ***begin…end***

```
if expression_logique
   then
     begin
        instruction_1 ;
        ...
        instruction_N ;
     end ;
```




```
Program exemple_test_2 ;

Uses Crt ;

Var a : INTEGER ;
    b : real ;

begin
  write('entrez un nombre entier : ') ;
  readln(a) ;
  if a <> 0
    then
      begin
        b := 1 / a ;
        writeln('l'inverse de a est : ', b) ;
      end ;
end.
```

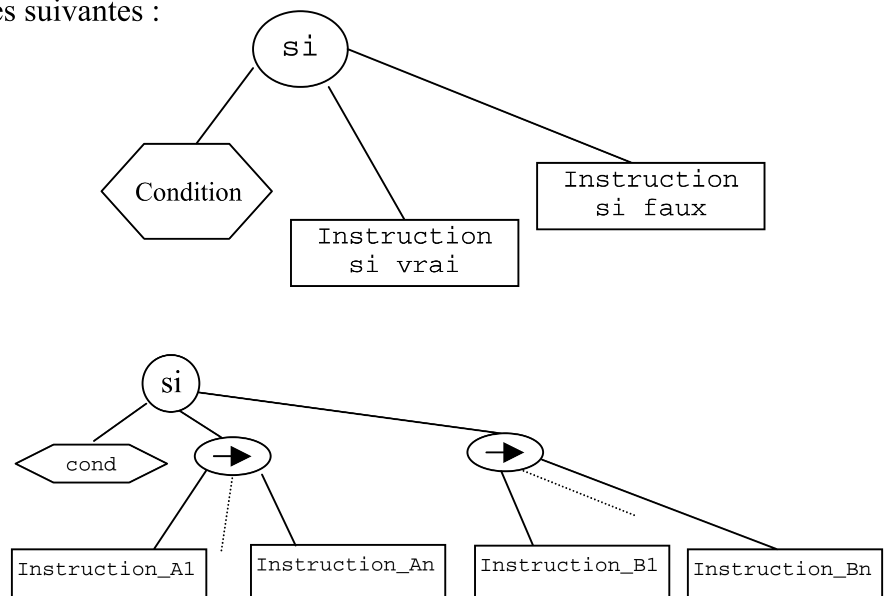
### 4.1.2. Instruction « if … then…else… ; »

Dans certains cas, on pourra faire exécuter des instructions si la condition est vraie ou si elle est fausse. Les deux syntaxes sont les suivantes :

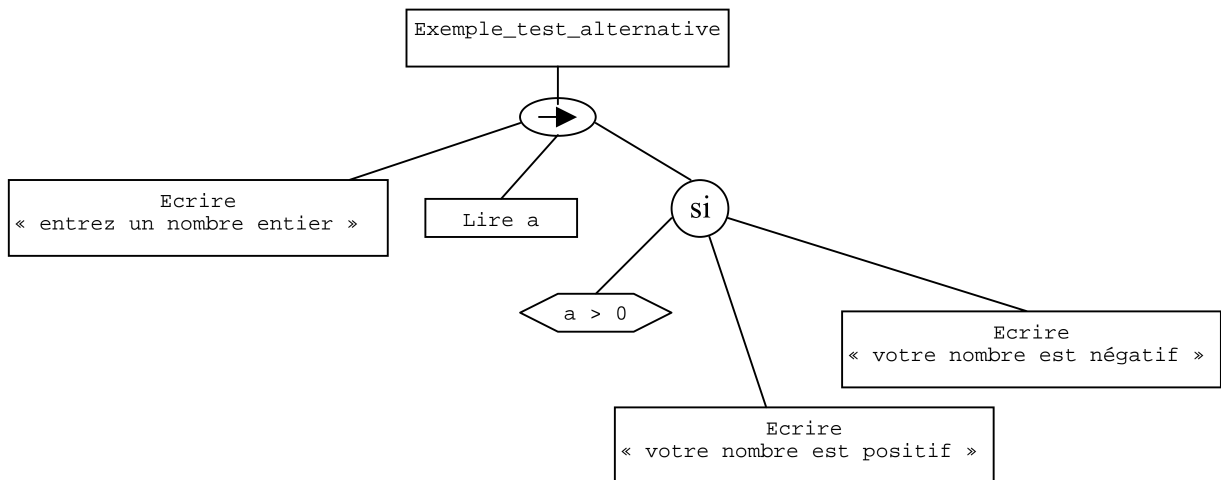
```
if expression_logique
  then instruction
  else instruction ;
```

**OU**

```
if expression_logique
  then
    begin
      instruction_A1 ;
      ...
     instruction_An ;

    end
  else
    begin
      instruction_B1 ;
      ...
      instruction_Bn ;
    end ;
```



exemple :




```
Program exemple_test_alternative ;

Uses Crt ;

Var a : INTEGER ;

begin
  write('entrez un nombre entier : ') ;
  readln(a) ;
  if a > 0
    then writeln('Votre nombre est positif')
    else writeln('Votre nombre est négatif') ;
end.
```

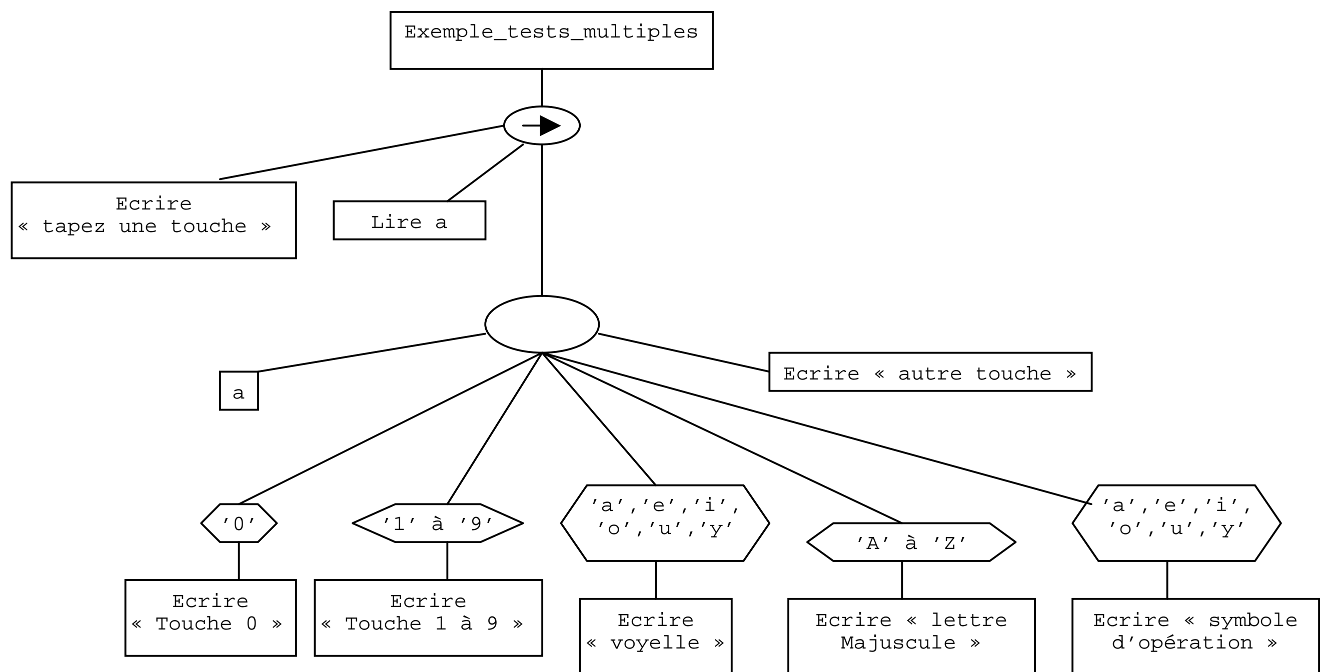
### 4.1.3. Instruction « case…of…else…end ; »

Pour éviter l'imbrication de plusieurs instructions « **if** » quand il y a plus de 2 choix, on préférera utiliser l'instruction de choix multiples « **case** ».

La syntaxe est la suivante :

```
case variable
  of
     etiquette_1 : instruction_1 ;
     ...
     etiquette_n : instruction_n ;
  else
     autre_instruction ;
end ;
```

**Remarques** : le mot « case » doit être suivi d'une variable de type entier ou caractère,
Les étiquettes sont du même type que la variable : constante, caractère, ensemble de constantes ou caractères.




```
Program exemple_tests_multiples ;

Uses Crt ;

Var a : CHAR ;

begin
  write('Tapez une touche: ') ;
  a := readkey ;
  case a of
    '0'                     : writeln('Touche 0 ') ;
    '1'..'9'                : writeln('Touche 1 à 9') ;
    'a','e','i','o','u','y' : writeln('voyelle') ;
    'A'..'Z'                : writeln('lettre majuscule') ;
    '-','+','*','/','='     : writeln('symbole d'opération') ;
    else
      writeln('autre touche') ;
  end ;
end.
```

---

```
Program exemple_tests_multiples_sans_else  ;

Uses Crt ;

Var x : INTEGER ;

begin
  write('Entrer un nombre entier : ') ;
  readln(x)
  case x of
    0      : writeln('vous avez entrez 0') ;
    1..100 : writeln('vous avez entrez un nombre entre 1 et 100') ;
  end ;
end.
```

## 4.2. Instructions répétitives (itérations)

Les itérations, appelées aussi structures répétitives ou boucles (loops) permettent de faire exécuter plusieurs fois certaines phases de programme. Le Pascal dispose de trois types de boucle : **while…do**, **for…do** et **repeat…until.**

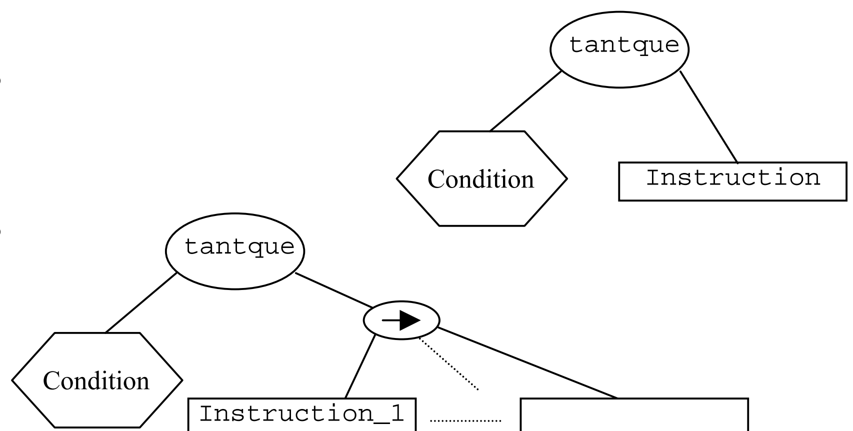
### 4.2.1. Instruction « while…do… ; »

L'instruction **while…do**, étant une boucle, est réalisée autant de fois que la condition reste **VRAIE**. La syntaxe est la suivante :

```
while condition_logique do
  instruction ;
```

**OU**

```
while condition_logique do
  begin
    instruction_1 ;
    ...
    instruction_N ;
  end ;
```




- **Exemple :**

```
var a : integer ;
begin
  a := -4 ;
  while a <> 0 do
    begin
      write(a,' ; ');
      a := a + 1 ;
    end ;
end.
```

```
Résultat :
-4 ; -3 ; -2 ; -1
```

- **ATTENTION à ne pas oublier :**

  - l'initialisation de la boucle **`(dans l'exemple ici a := -4)`**
  - la fin de la condition dans la liste d'instructions à exécuter afin de ne pas obtenir une boucle infinie ! **`(dans l'exemple ici a := a + 1)`**

### 4.2.2. Instruction « for…do…»

La structure **for…do** est une boucle qui teste une condition avant d'exécuter les instructions qui en dépendent. Les instructions sont exécutées tant que la condition remplie est **VRAIE**.

Les syntaxes possibles sont les suivantes :

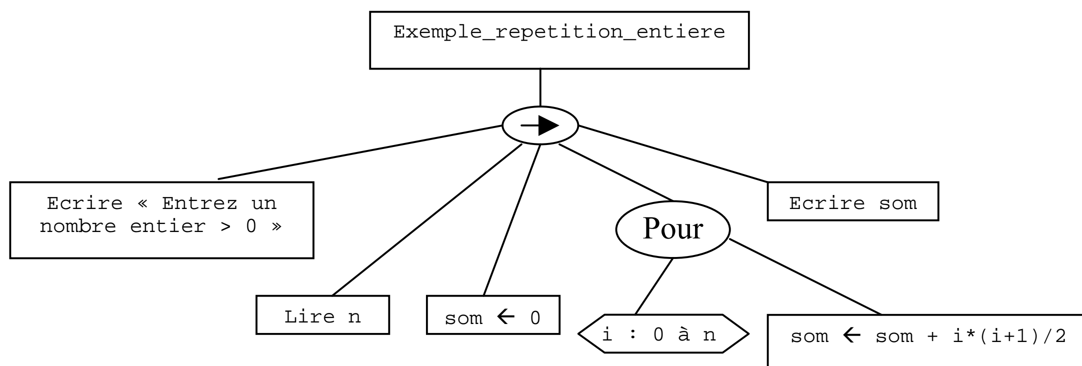
```
     for variable_entière := debut to  fin do instruction ;
ou
     for variable_entière := debut downto  fin do instruction ;

ou

     for variable_entière := debut to fin do
       begin
         instruction_1 ;
         ...
         instruction_N ;
       end ;
```

Ou bien « **downto** »

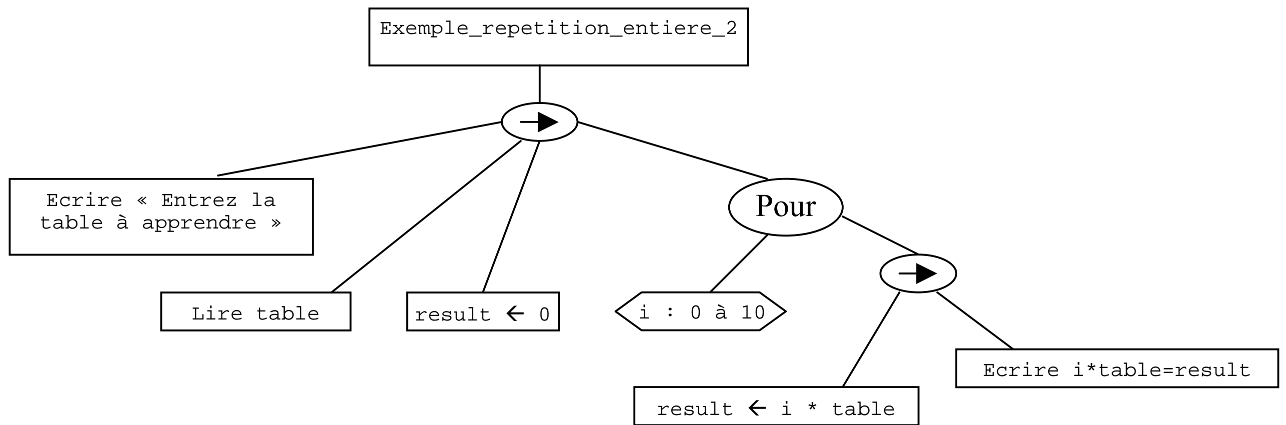
Le morceau « **for variable_entière** := **debut** to **fin** do » représente l'entête de boucle contenant des instructions à exécuter. Cet entête contient :

- une expression initialisant les variables de contrôle qu'il faut initialiser avant d'entrer dans la boucle « **variable_entière** := **debut** »
- la condition de bouclage « to **fin** »

- **Exemple 1**




```
Program exemple_repetition_entiere ;

Uses Crt ;

Var i,n : INTEGER ;
    Som : REAL ;

begin
  write('Entrez un nombre entier > 0 : ') ;
  readln(n) ;
  som := 0 ;
  for i:=0 to n do som := som + i*(i+1)/2 ;
  writeln('la somme vaut : ',som) ;
end.
```




```
Program exemple_repetition_entiere_2 ;

Uses Crt ;

Var result, table : INTEGER ;

begin
  write('Entrez la table à apprendre : ') ;
  readln(table) ;
  result := 0 ;
  for i:=0 to 10 do
    begin
      result := i * table ;
      writeln(i,' * ',table,' = ',result) ;
    end ;
end.
```

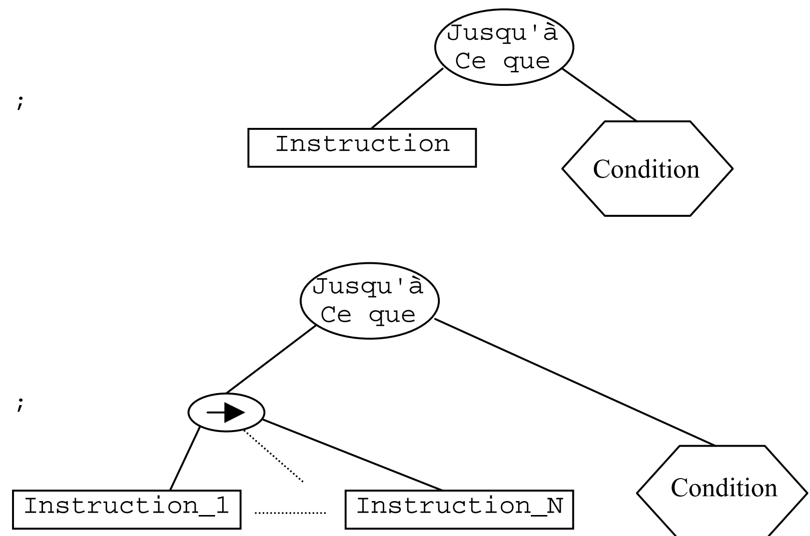
### 4.2.3. Instruction « repeat … until ... ; »

Contrairement aux structures **for…do** et **while…do**, la boucle **repeat…until** teste sa condition après exécution de l'instruction du corps de la boucle. La boucle est répétée jusqu'à ce que la condition devienne **Vrai**. La syntaxe est la suivante :

```
repeat
    instruction ;
until condition_logique ;
```

**OU**

```
repeat
  begin
    instruction_1 ;
     ...
    instruction_N ;
  end
until condition_logique ;
```




**Remarques** : la boucle **repeat…until** est exécutée au moins une fois. Le bloc **begin…end** n'a pas besoin d'être spécifié car le **repeat until** fait office de bloc

- **Exemple**

```
Program essai ;

var x : integer ;

begin
  x := 10 ;
  repeat
    write(x);           (* affichage des nombres de 10 à 1 *)
    x := x-1 ;
  until x=0 ;
end.
```

## 4.3. Instructions de branchement

Les instructions de branchement transfèrent le contrôle du programme d'une instruction à une autre, cette dernière n'étant pas directement écrite après l'exécution précédente effectuée. Il existe 3 instructions de branchement :

- break,
- continue,
- goto,

- **Utilisation :**

  - uniquement à l'intérieur d'une structure **for…do**, **while…do**, **repeat…until**.

- **break** : provoque l'arrêt avant terme de ces instructions.

- **Continue** : cette instruction permet de sauter un seul passage dans la boucle. L'exécution reprend alors au prochain passage dans la boucle, c'est à dire que l'instruction **continue** provoque un « saut » au début du prochain passage dans la boucle. Le programme reprend alors son cours.

- **goto** <étiquette> : <**étiquette**> peut être n'importe quel libellé admis par le langage. L'instruction identifiée par <**étiquette**> doit avoir la syntaxe suivante :
  <étiquette> : <instruction> ;
  La commande de branchement **goto** et l'instruction repérée par <**étiquette**> doivent se trouver dans la même fonction. l'instruction **goto** provoque un saut à un endroit du programme repéré par une étiquette (label). Le programme continue alors à l'instruction qui se trouve à cet endroit là.

**REMARQUE** : ces instructions sont rarement utilisées

# 5. Types de données complexes

Le Pascal dispose de deux types de données complexes : les tableaux et les enregistrements. Un tableau ne contient que des données de type identique, alors qu'un enregistrement peut être composée à partir d'éléments différents.

## 5.1. Tableaux

Un tableau est une variable qui se compose d'un certain nombre de données élémentaires de même type, rangées en mémoire les unes à la suite des autres. Chaque donnée élémentaire représente elle-même une variable. Le type des éléments du tableau peut être n'importe lequel du langage Pascal :

- types élémentaires : integer, longint,, shortint, byte,  real, word, char, string ;
- pointeur,
- tableau,
- enregistrement.

### 5.1.1. Tableaux à une dimension

Un tableau unidimensionnel est composé d'éléments qui ne sont pas eux-mêmes des tableaux. On le considère comme ayant « n » colonnes et une seule ligne. La définition d'un tableau unidimensionnel admet la syntaxe suivante :

**Var  Nom du tableau : array [ensemble] of type ;**

- **Type :** spécifie le type des éléments dont doit se composer le tableau.
- **ensemble** est un intervalle qui détermine le nombre d'éléments du tableau. Les crochets font parti de la syntaxe de la définition.

- **Exemple**

```
Var i : array[1..4]of integer ;
    J : array['a'..'z'] of real ;
    k : array[-8..2]of integer ;
```

- **Indexation**

Un tableau est composé de plusieurs variables individuelles de même type qu'il faut distinguer nommément. Un moyen simple consiste à numéroter les éléments d'un tableau : on les identifie par un nombre ou une lettre, nommé indice ou index. Les valeurs que peut prendre l'index doivent être des entiers ou des caractères. Le nom sous lequel on peut ainsi décrire un élément quelconque du tableau doit obéir à la syntaxe suivante :

**Nom du tableau [index]**

```
exemples :     i [1]  j['a']   k[-3]
```

- **Initialisation**

  - *affectation :*

```
var v : array[1..5] of integer ;
begin
  v[1]:= 0 ;
  v[2]:= 0 ;
  v[3]:= 0 ;
  v[4]:= 0 ;
  v[5]:= 0 ;
end.
```

  - *boucle :*

```
var v : array[1..5] of integer ;
    k : integer ;
begin
  for k := 1 to 5 do
    v[k] := 0 ;
end.
```

### 5.1.2. Tableaux à plusieurs dimensions

- **Définition**

**Var Nom du tableau : array [ensemblee_e1, …, ensemble_eN] of type ;**

Le nombre de dimensions d'un tableau n'est pas limité. Il est fixé par le nombre d'ensembles entre crochets []**.** Ces ensembles renseignent sur le nombre d'éléments dans chaque dimension du tableau.

Exemple

**Var k : array[0..2,0..3] of integer** crée un tableau nommé **k** et possédant 12 éléments de type **integer.** On peut les représenter sous forme de lignes et colonnes :

| k[0,0] | k[0,1] | k[0,2] | k[0,3] |
|---|---|---|---|
| k[1,0] | k[1,1] | k[1,2] | k[1,3] |
| k[2,0] | k[2,1] | k[2,2] | k[2,3] |

- **Initialisation**

  - *affectation :*

```
var v : array[0..2,0..3] of integer ;
begin
  v[0,0]:= 0 ;
  v[0,1]:= 0 ;
  v[0,2]:= 0 ;
  v[0,3]:= 0 ;
  v[1,0]:= 0 ;
  ...
end.
```

- *boucle :*

```
var v : array[0..2,0..3] of integer ;
    i,j : integer ;

begin
  for i := 0 to 2 do
    for j := 0 to 3 do
      v[i,j] := 0 ;
  ...
end.
```

- **Entrée et sortie**

```
Program exemple ;

var v : array[0..2,0..3] of integer ;
    i,j : integer ;

begin

  (* Cette boucle permet de remplir les éléments du tableau *)

  for i:= 0 to 2 do              { lignes    }
    for j:= 0 to 3 do       { colonnes  }
      readln(v[i,j]);

  (* Afficher les valeurs sur trois lignes et quatre colonnes *)

  for i:=0 to 2 do             { lignes }
    begin
      for j:= 0 to 3 do { colonnes }
        write(k[i,j]:5);
      writeln;
    end ;

end.
```

### 5.1.3. String ( chaînes de caractères)

Les chaînes de caractères (string) sont des suites de caractères composées de signes faisant partie du jeu de caractères représentables de l'ordinateur (code ASCII).

En Pascal, les chaînes de caractères, qu'elles soient constantes ou variables, peuvent être considérées comme des tableaux à une dimension, ayant des éléments de type **char.**

La définition « **var s : string[12]** » crée, par exemple, une chaîne de caractères douze éléments de type **char.** On peut y ranger jusqu'à 12 caractères :

| b | e | l | l | e | | c | h | a | î | n | e |
|---|---|---|---|---|---|---|---|---|---|---|---|
| s[1] | s[2] | s[3] | s[4] | s[5] | s[6] | s[7] | s[8] | s[9] | s[10] | s[11] | s[12] |

La définition « **var t : string** » crée, par exemple, une chaîne de caractères pouvant comporter au maximum 256 éléments de type **char.**

## Opération d'entrées-sorties

```
var
  S: String[4];
begin
  S := '';
  Write(S) ;     { rien n'est affiché : la chaîne est vide }
  S := 'toto' ;
  S[1] := 'm' ;
  Write(S) ;      { la chaîne de caractère contient « moto » }
end.
```

---------------------

```
var
  S: String;
begin
  S := 'Honest Lincoln';
  Insert('Abe ', S, 8);  { S contient  'Honest Abe Lincoln' }
end.
```

---------------------

```
var S: String;
 begin
   S := 'ABCDEF';
   S := Copy(S, 2, 3);    { S contient 'BCD' }
 end.
```

---------------------

```
var
   S: String;
 begin
   S := Concat('ABC', 'DEF');    { S contient  'ABCDE' }
 end.
```

---------------------

```
var S: String;
begin
  S := '   123.5';
  { Convertit les espaces en zéros }
  while Pos(' ', S) > 0 do
    S[Pos(' ', S)] := '0';
end.

---------------

var
  S: String;
 begin
  Readln (S);
  Writeln('"', S,'"');
  Writeln('longueur de la chaîne = ', Length(S));
 end.
```

# 5.2. Enregistrements

Pour faire un agenda, on a besoin, par exemple, de connaître pour chaque personne un certain nombre de renseignements que l'on va regrouper dans un unique enregistrement (nom, prénom, ville, tél…). Le Pascal met à disposition une variable enregistrement (record).

## 5.2.1. Déclaration des enregistrements

Avant de pouvoir définir une variable enregistrement, il faut fournir au compilateur un type d'enregistrement décrivant l'aspect de l'enregistrement à créer. On indique, pour cela, les champs de l'enregistrement. Chaque champ est introduit avec son type et son nom.
Exemple de syntaxe d'une définition d'enregistrement :

```
Program agenda ;

Uses crt ;

Type   personne = record
                    Nom : string[15] ;
                    Prenom : string[15] ;
                    age : integer ;
                    solde : real ;
                    sincere : boolean ;
                  end ;

var   x : personne ;
...
```

## 5.2.2. Opérations sur les variables structurées

- **L'opérateur de champ**

L'opérateur de champ est placé entre le nom de la variable structurée et celui du champ concerné :

**Nom_Variable.Nom_Champ**

En reprenant l'exemple cité précédemment, les champs de l'enregistrement sont donc accessibles via les noms :

```
x.nom := 'De Wyliers' ;
x.prenom := 'Carnage à Mazamet' ;
x.age := 33 ;
x.solde := -3200.45 ;
x.sincere := false ;
```

# 6. Constantes & types

Les deux mots clé **const** et **type** permettent de définir des constantes et des types de données,

## 6.1. Déclarations des constantes

Elles se déclarent de manière locales (dans les procédures ou les fonctions ou alors de manière globales (juste après l'ente de programme )

**Exemple :**

```
Const  PI = 3.14 ; UN = 1 ; DIX = 10 ;
       TVA = 0.196 ; Taux = 19.6E-2 ;
       Souligne = '--------' ;
       titre = 'résultats' ;
```

## 6.2. Déclarations de types

La notion de type est liée à la notion de données. Elle représente l'ensemble des valeurs que peut prendre une donnée. Les données sont traitées par l'intermédiaire de variables. C'est donc l'identificateur de la variable associée qui sera défini comme ayant un certain type.

- **Type standard :**

  le type entier (INTEGER), le type réel (REAL), le type booléen (BOOLEAN), le type caractère (CHAR)

- **Type non standard :**

  Permet de spécifier un type particulier : le type intervalle peut être introduit par le mot set (ensemble)

  Exemple:

```
type
      Day = (Sun, Mon, Tue, Wed, Thu, Fri, Sat);
      CharSet = set of Char;
      Digits = set of 0..9;
      Days = set of Day;
      Printemps = (Mars,avril,mai,juin) ;
      Longueurmois = 0..31 ;
      Lettremajus = 'A'..'Z' ;
      Age = 0..120 ;

Var   jour : day ;
      alpha : lettremajus ;
      age_personne : age ;
```

# 7. Variables locales et variables globales

- **Variable globale**

  Une variable globale est définie hors de toute fonction ou procédure. Une variable globale est connue (donc utilisable) dans tout le fichier (soit dans chaque bloc ou chacune des fonctions ou procédure) où elle est définie, cela à partir de l'endroit de sa définition.

- **Variable locale**

  Une variable locale, au contraire, est définie à l'intérieur d'une fonction, d'une procédure ou d'un bloc. Une telle définition de variable est ainsi autorisée au début de n'importe quel bloc. Les variables locales sont connues (donc utilisables) uniquement à l'intérieur de la fonction ou du bloc contenant leur définition.

- **Variables homonymes**

Il est autorisé (mais pas recommandé) de définir des variables de même nom au sein du même programme, du moment que leurs domaines de validité diffèrent.

- Si les domaines de validité de deux variables se recoupent, le compilateur n'accepte naturellement pas des variables portant le même nom.
- Si les domaines de validité de deux variables homonymes sont exclusifs, alors la situation est claire : chaque variable est inconnue dans le domaine de validité de l'autre, et ne peut donc pas y être utilisée. Les deux variables n'entrent pas en conflit.

Si, dans un programme, existent deux variables de même nom, et si la portée de l'une (contenu) est incluse dans celle de l'autre (contenant), alors on privilégie, dans le domaine intérieur (contenu), la variable qui y est définie. On dit également que la variable définie dans le contenu cache celle définie dans le contenant. Hors du contenu, on privilégie la variable définie dans le contenant.

# 8. Procédures & Fonctions

- **Généralités**

- Une procédure ou une fonction est une portion de programme composée d'une ou plusieurs instructions et devant accomplir une certaine tâche. On distingue 2 cas :
  - les procédures ou fonctions prédéfinies des bibliothèques (telles que **writeln** ou **readln),** livrées avec le compilateur, qui sont intégrées au programme seulement lors de l'édition des liens,
  - les procédures ou fonctions personnelles que le programmeur écrit lui-même.

- Un programme Pascal peut comprendre plusieurs procédures ou fonctions personnelles. Mais un programme contient au moins le programme principal « **begin…end.** ». C'est par lui que débute toujours l'exécution du programme.

- Un programme Pascal peut être considéré comme une collection de procédures et de fonctions. Les procédures et les fonctions peuvent être disposées dans un ordre quelconque **à condition qu'elles soient déclarées préalablement**, sinon, il faut respecter un ordre de définition : l'appelant succède l'appelé.

- Contrairement aux procédures, les fonctions peuvent renvoyer au programme (plus précisément à la fonction appelante) le résultat de leur travail, cela sous forme de valeur réutilisable.

## 8.1. Définition des procédures et des fonctions

- **Une définition de fonction spécifie :**

  - le type de la valeur renvoyée par la fonction,
  - le nom de la fonction,
  - les paramètres (arguments) qui sont passés à la fonction pour y être traités,
  - les variables locales utilisées par la fonction,
  - d'autres procédures ou fonctions invoquées par la fonction,
  - les instructions que doit exécuter la fonction.
  - La valeur renvoyée

La syntaxe est :

```
function nom [(paramètres...)] : type renvoyé ;   <-- En-tête de la fonction
 [Définitions des variables locales]
 [Déclaration des proc et fonc supplémentaires]
begin
  Instructions ;     } Corps de la fonction
  nom := ... ;
end ;
```

Remarques : Les [] indiquent que les spécifications sont facultatives

- **Une définition de procédure spécifie :**

  - le nom de la procédure,
  - les paramètres (arguments) qui sont passés à la procédure pour y être traités,
  - les variables locales utilisées par la procédure,
  - d'autres procédures ou fonctions invoquées par la procédure,
  - les instructions que doit exécuter la procédure.

La syntaxe est :

```
procedure nom [(paramètres...)] ;
 [Définitions des variables locales]
 [Déclaration des proc et fonc supplémentaires]
begin
  Instructions ;
end ;
```

*En-tête de la procédure* (→ procedure nom [(paramètres...)] ;)

*Corps de la procédure* (→ begin … end ;)

<u>Remarques</u> : Les [] indiquent que les spécifications sont facultatives

- **Exemple d'une fonction calculant le volume d'un cube :**

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

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

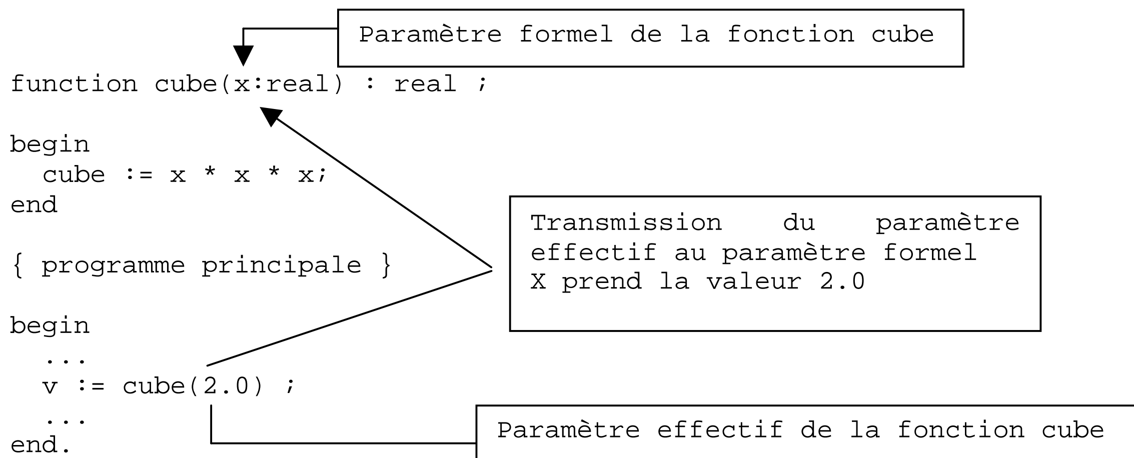
*<u>Remarques sur l'exemple :</u>*

- <u>Valeur de retour :</u>
  Le mot clé **real** après le nom de la fonction indique que la fonction cube renvoie une valeur de type **real**. Les valeurs de retour des fonctions sont renvoyées à la fonction appelante via l'instruction contenant le nom de la fonction : **cube := …**

- <u>Paramètres :</u>
  Le nom de la fonction (ici cube) est complété par une paire de parenthèses avec paramètres. Les parenthèses ne sont pas obligatoires. La fonction **cube** a exactement un paramètre, 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. Ces derniers sont dits paramètres effectifs ou arguments de la fonction.

- <u>L'instruction « **cube :=** » :</u>

La fonction **cube** calcule le cube d'une valeur et retourne le résultat à la fonction appelante. Cela se fait par l'intermédiaire de l'instruction **nomdelafonction := .** Cette instruction met fin à l'exécution des instructions d'une fonction et rend le contrôle du programme à la fonction appelante.




**<u>Exemple 1</u> : Programme calculant le volume d'un cube**

```
Program calcul_volume_cube ;

Var
     e : real ;            (* longueur de l'arête du cube   *)
     volume : real ;

{ ---------------------- fonction cube ----------------------------- }

Function cube(x :real) : real ;

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

{ ---------------------- programme principal ------------------------- }

begin
  write('Longueur de l'arête : ');
  readln(e) ;
  volume := cube(e);    { Appel de la fonction 'cube' et affectation de }
                        { la valeur retournée à la variable 'volume'    }
  writeln('Le volume du cube est : ', volume);
end.
```

- **<u>Exemple 2</u> : Fonction avec plusieurs paramètres**

```
{ Programme calculant le volume d'un parallélépipède  }

program para ;

Var
     long, larg, haut : real ;
     volume : real ;

{ ----------------------- fonction cube ------------------------------ }

Function parallel( x, y, z : real) : real ;

begin
  parallel := x * y * z ;
end ;
{ ----------------------- programme principal ------------------------- }

begin
  write('Longueur, largeur, hauteur : ');
  readln(long, larg, haut) ;
  volume := parallel(long,larg,haut) ;  { Appel de la fonction }
  writeln('Le volume du parallélépipède est : ', volume);
end.
```

- **<u>Exemple 3</u> : Fonction SANS paramètre**

```
Program calcul_volume_cube_version_2 ;

Var volume : real ;

{ ----------------------- fonction cube ------------------------------ }

Function cube : real ;

Var  e : real ;

begin
  write('Longueur de l'arête : ');
  readln(e) ;
  cube := e * e * e ;
end ;

{ ----------------------- programme principal ------------------------- }

begin
  volume := cube ;   { Appel de la fonction 'cube' et affectation de }
                     { la valeur retournée à la variable 'volume'   }
  writeln('Le volume du cube est : ', volume);
end.
```

- **<u>Exemple 4</u> : procédure avec plusieurs paramètres**

```
{ Programme calculant le volume d'un parallélépipède  }

program para2 ;

Var
     long, larg, haut : real ;
     volume : real ;

{ ---------------------- procedure cube ----------------------------- }

procedure parallel( x, y, z : real)  ;

begin
  volume := x * y * z ;
end ;
{ ---------------------- programme principal ------------------------ }

begin
  write('Longueur, largeur, hauteur : ');
  readln(long, larg, haut) ;  { saisie multiples : 3 méthodes à voir ! }
  parallel ;  { appel de la procedure }
  writeln('Le volume du parallélépipède est : ', volume);
end.
```

- **<u>Exemple 5</u> : procedure SANS paramètre**

```
Program calcul_volume_cube_version_2_bis ;

Var volume : real ;

{ ---------------------- fonction cube ----------------------------- }

procedure cube ;

Var  e : real ;

begin
  write('Longueur de l'arête : ');
  readln(e) ;
  volume := e * e * e ;
end ;

{ ---------------------- programme principal ------------------------ }

begin
  cube ;   { Appel de la procedure 'cube' }
  writeln('Le volume du cube est : ', volume);
end.
```

# 8.2. Déclaration globales des procédure et des fonctions

La définition d'une procédure ou d’une fonction doit être globale, c'est-à-dire se faire hors de toute procédure ou fonction. Mais, à part cela, elles peuvent intervenir n'importe où dans le programme. Jusqu'à présent, les définitions personnelles étaient toujours placées avant le programme principal. D'une manière plus générale, les procédures et fonctions étaient définies, dans le programme, avant le programme principal.

- **Exemple :**

```
Program exemple ;

Uses crt ;

Var v : real ;

function first : real ; forward ;              { déclaration globale }
function second(x :real) : real ; forward ;    { des fonctions }

{ ---------------------------------------------------------------------- }

function first ; { définition de la fonction 'first' }

var e : real ;

begin
     write('Arête ? ');
     readln(e);
     first := second(e)  ; (* appel de la fonction 'second' *)
end ;

{ ---------------------------------------------------------------------- }

function second ; { définition de la fonction 'second' }

begin
  second := x*x*x ;
end ;

{ --------- programme principal ------------------ }

begin
  v := first ;
  write('Volume = ',v) ;
end.
```

# 8.3. Passage de paramètres

Les paramètres effectifs d'une procédure ou d'une fonction sont des expressions qui doivent correspondre en nombre et en type aux paramètres formels spécifiés dans la définition de la procédure ou de la fonction. Il existe deux méthodes pour transmettre des paramètres effectifs à une procédure ou une fonction:

- si la procédure ou la fonction reçoit, comme paramètre, la valeur d'une donnée (plus précisément, une copie de celle-ci), alors on parle de transmission par valeur.

- si la procédure ou la fonction reçoit, non pas la valeur de la donnée comme paramètre, mais son adresse, alors on parle de transmission par adresse.

## 8.3.1. Passage par valeur (call by value)

La procédure ou la fonction appelée reçoit une copie de la valeur de l'objet passé comme paramètre effectif. Cette copie est affectée au paramètre formel correspondant. La procédure ou la fonction travaille donc sur un duplicata, et non sur l'original de la valeur transmise.

- **Exemple 1 :**

```
Program Variable_1 ;

Uses Crt ;

Var   i, j : INTEGER ;
      k : CHAR ;

{ -------------------------- }

Procedure Fait ( a : INTEGER ) ;

Begin
   j := a + 1 ;
  a := a * 2 ;
   k := pred(k) ;
  Writeln( a , i, j , k );
  j := j - 1 ;
End ;

{ ------------------------------ }

Begin
   i := 5 ; j := 9 ; k := 'B' ;
   Writeln( i , j , k );
   Fait(i) ;
   Writeln( i , j , k );
End.
```

```
Résultat de l'affichage:

5    9    B
10   5    6    A
5    5    A
```

→ **remarque :** Les copies des valeurs des variables passées comme paramètres effectifs sont modifiées dans la procédure appelée, sans que cela se répercute sur les valeurs originales des variables, dans la fonction appelante. La valeur de **i** n'est pas affectée.

### 8.3.2. Passage par adresse (call by reference)

Lorsque l'on veut qu'une procédure ou une fonction puisse modifier la valeur d'une donnée passée comme paramètre, il faut lui transmettre non pas la valeur de l'objet concerné, mais son adresse. Cette technique de transmission des paramètres est dite "passage par adresse". La conséquence en est que la fonction appelée ne travaille plus sur une copie de l'objet transmis, mais sur l'objet lui-même (car la procédure ou la fonction en connaît l'adresse). Le passage des paramètres par adresse permet donc à une procédure ou une fonction de modifier les valeurs des variables.

- **Exemple**

```
Program Variable_1 ;

Uses Crt ;

Var   a, b  : INTEGER ;

{ ------------------------- }


Procedure calc2 ( x : INTEGER ) ;

Var y : INTEGER ;

Begin
      y := x * x ;
      Writeln(x,y) ;
      x := x + 1 ;
      y := y + 1 ;
End ;


{ ----------------------------- }

Begin
      a := 7 ;
      b := 3 ;
      Writeln(a,b);
      Calc2 (a);
      Writeln(a,b);
End.
```

```
Program Variable_2 ;

Uses Crt ;

Var   a, b  : INTEGER ;

{ ------------------------- }


Procedure calc2 (var x:INTEGER ) ;

Var y : INTEGER ;

Begin
      y := x * x ;
      Writeln(x,y) ;
      x := x + 1 ;
      y := y + 1 ;
End ;


{ ----------------------------- }

Begin
      a := 7 ;
      b := 3 ;
      Writeln(a,b);
      Calc2 (a);
      Writeln(a,b);
End.
```

```
Résultat de l'affichage:

7   3
7   49
7   3
```

```
Résultat de l'affichage:

7   3
7   49
8   3
```

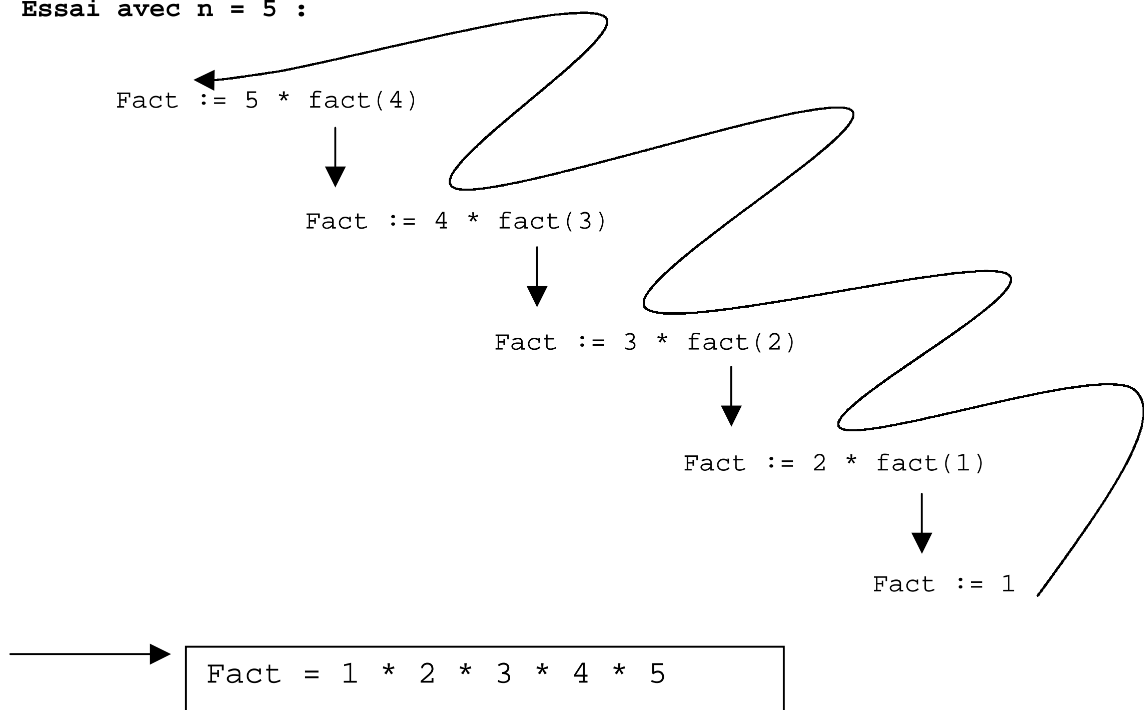
Dans Variable_2, l'adresse de **a** est alors transmise à **calc2** dans **var x :integer**. La procédure n'accède plus maintenant à une copie locale de la valeur de la variable a de la fonction appelante. En revanche, elle intervient sur le contenu de l'emplacement mémoire où est rangée cette variable. De ce fait, la valeur d'origine de **a** es réellement changée.

## 8.4. Fonction récursives

En Pascal, les fonctions peuvent exécuter d'autres fonctions, parmi lesquelles elles-mêmes. En pareil cas, on parle d'appel récursif de fonction ou de fonction récursive.

- **Exemple :**

```
Program factoriel ;

Uses crt ;

Var nombre, resultat : integer ;


Function fact ( n : integer ) : integer ;

Begin
  If n > 1
    then fact := n * fact(n-1)
    else fact := 1 ;
End ;

{ ** programme principal ** }

Begin
  Write('entrer un entier : ') ;
  Readln(nombre) ;
  Resultat := fact(nombre) ;
  Writeln(nombre,' ! = ',resultat) ;
End.
```



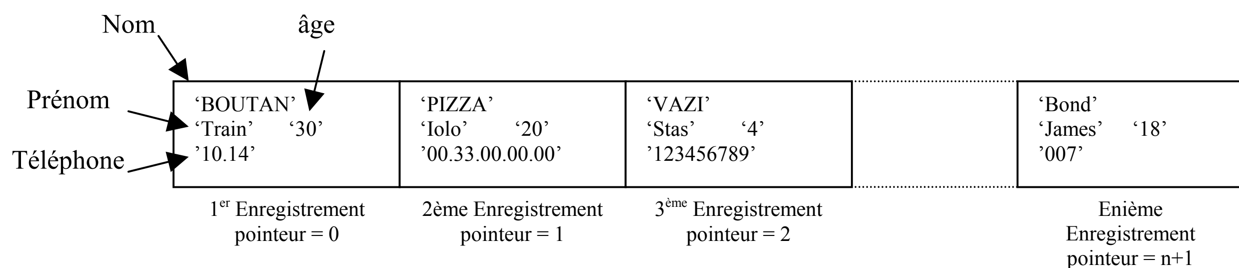
# 9. Gestion de fichiers

Afin de conserver des données sur support physique (disque dur, disquette), on est amené à exécuter des opérations de lecture/écriture.

***Voir TP pour plus de détails***

Résumé :

Exemple 1 d'enregistrements dans un fichier :




**Le type record et la déclaration d'un fichier**

```
type
      coord =  record
                  nom    : string ;
                  prenom : string ;
                  age    : integer ;
                  tel    : string ;
               end ;
var
      personne : coord ;
      f        : file of coord ; {  fichier d'enregistrement }
      g        : text ;          {  fichier de caractères (file of char) }
```

**Création d'un fichier**

*Attention, si celui-ci existe, il est écrasé.*

```
Begin
      assign(f,'toto.dat');
      rewrite(f);
      close(f);
end ;
```

**Ouverture d'un fichier**

*à condition qu'il se trouve dans le répertoire courant, sinon le chemin doit être spécifié du style : 'C:\DOCS\TOTO.DAT'*

```
begin
      assign(f,'C:\docs\toto.dat');
      reset(f);              { Le pointeur de fichier se positionne     }
      ...                    { automatiquement sur l'enregistrement 0   }
      close(f);
end ;
```

**Ouverture d'un fichier (2) avec test d'existence**

*La condition d'existence est ici testée*

```
begin
      assign(f,'toto.dat');
      {$i-} reset(f) {$i+};
      if ioresult <> 0 then write('le fichier est absent') ;
      ...
      close(f);
end ;
```

**Ecriture dans un fichier**

```
begin
      assign(f,'toto.dat');
      reset(f);              { le pointeur de fichier se positionne à 0 }
      personne.nom := 'DUBOIS'
      personne.prenom := 'David'
      personne.tel := '00.00.00.00.00'
      write(f,personne) { après cette instruction, le pointeur de fichier
                          s'incrémente automatiquement de 1 }
      ...
      close(f);
end ;
```

**Lecture d'un fichier**

```
begin
      assign(f,'toto.dat');
      reset(f);              { le pointeur de fichier se positionne à 0 }
      read(f,personne)       { après cette instruction, le pointeur de
fichier
                               s'incrémente automatiquement de 1 }
      write('nom : ', personne.nom) ;
      ...
      close(f);
end ;
```

**fonctions et procédures utilisables avec le traitement de fichier**

`eof(f) :`

fonction qui renvoie le booléen « TRUE » si le pointeur est en fin de fichier sinon renvoie « FALSE »

`filesize(f) :`

fonction qui renvoie un entier donnant le nombre d'enregistrement du fichier

`seek(f,position) :`

procédure plaçant le pointeur de fichier à la valeur « position »

# 10. ANNEXES

## 10.1. Articulation d'un programme

```
program ... ;      { Tête du programme }

uses  ... ; { Utilisation des bibliothèques }
const ... ; { Déclaration des constantes }
type  ... ; { Déclaration des types }
var   ... ; { Déclaration des variables}

procedure   ... ; { Procédures }
function    ... ; { Fonctions }

begin             { programme principal }
  instruction ;
  ...
end. {fin de programme }
```

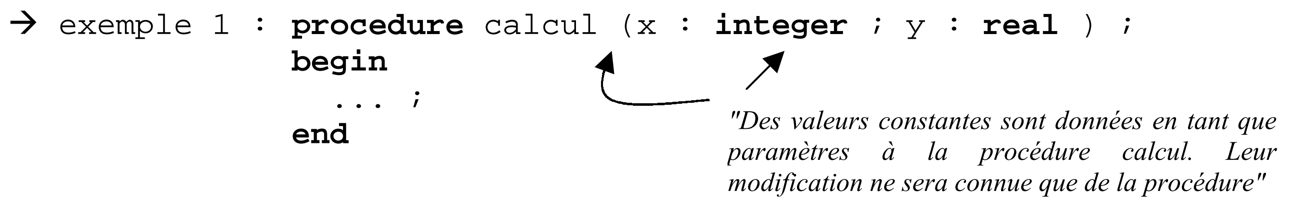
## 10.2. Procédure

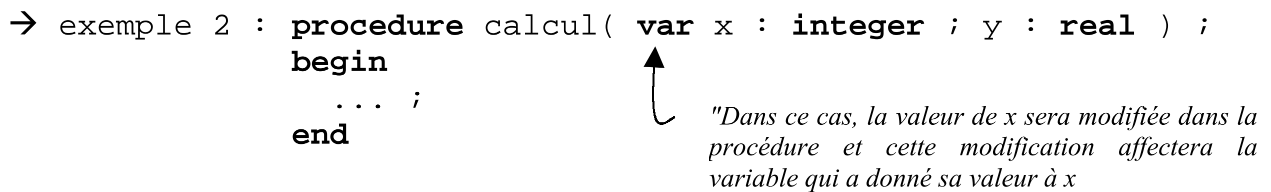
```
procedure NomdelaProcédure ( paramètre1 ; ... )

var nomdevariable : typedevariable ;     { déclaration facultative }

begin
  instruction ;
  ...
end;
```

## 10.3. Fonction

```
function NomdelaFonction ( paramètre1 ; ... ) : typedevariable ;

var nomdevariable : typedevariable ;         { déclaration facultative }

begin
  instruction ;
  ...
end;
```

# 10.4. Paramètres des procédures et fonctions




# 10.5. Exemple de programme

```
Program calcul_interet_simple ;

Uses Crt ;

Const  euro = 6.55957 ;

Var    c,t,i : Real ;
       d : Integer ;
       touche : Char ;

{ --- calcul de l'interet ------------------------------------- }

function calcul_i ( capital : Real; duree : integer; taux : real ) :
real ;

Var interet : Real ;

begin
  interet := capital * taux / 100 ;
  interet := interet * duree / 12 ;  { formule arrondie : I=C*t*d/12 }
  calcul_i := interet ;
end;

{ --- saisie des nombres ----------- }

procedure saisie ;

begin
  clrscr ;
  write('Entrez le capital : ');
  readln(c);
  write('Entrez la duree de placement (en mois): ');
  readln(d);
  write('Entrez le taux de placement : ');
  readln(t);
end;
```

```
{ ----------- programme principal ------------------------------ }

begin
  repeat
    saisie ;
    i := calcul_i(c,d,t);
    writeln('les interets seront de : ',i,' F');
    writeln('ce qui équivaut en Euro à : ',i*euro) ;
    writeln('Voulez-vous faire un autre calcul (O/N) ? :') ;
    repeat
      touche := upcase(readkey);
    until touche ='N' ;
  clrscr ;
end.
```

# 10.6. Mots réservés

Mots réservés :

| | | | |
|---|---|---|---|
| absolute | exports | label | Resident |
| and | external | library | set |
| array | far | mod | shl |
| asm | file | near | shr |
| assembler | for | nil | string |
| begin | forward | not | then |
| case | function | object | to |
| const | goto | of | type |
| constructor | if | or | unit |
| destructor | implementation | packed | until |
| div | in | private | uses |
| do | index | procedure | var |
| downto | inherited | program | virtual |
| else | inline | public | while |
| end | interface | record | with |
| export | interrupt | repeat | xor |

Unité Crt :

| | | | |
|---|---|---|---|
| AssignCrt | GotoXY | NormVideo | TextColor |
| ClrEol | HighVideo | NoSound | TextMode |
| ClrScr | InsLine | ReadKey | WhereX |
| Delay | KeyPressed | Sound | WhereY |
| DelLine | LowVideo | TextBackground | |

Unité dos :

| | | | | |
|---|---|---|---|---|
| DiskFree | FindNext | GetIntVec | PackTime | SetVerify |
| DiskSize | FSearch | GetTime | SetCBreak | SwapVectors |
| DosExitCode | Fsplit | GetVerify | SetDate | UnpackTime |
| DosVersion | GetCBreak | Intr | SetFAttr | |
| Exec | GetDate | Keep | SetFTime | |
| FExpand | GetFAttr | line | SetIntVec | |
| FindFirst | GetFTime | MsDos | SetTime | |

Unité graph :

| | | | |
|---|---|---|---|
| Arc | GetFillSettins | ImageSize | SetAllPalette |
| Bar | GetGraphMode | InitGraph | SetAspectRatio |
| Bar3D | GetImage | InstallUserDriver | SetBkColor |
| Circle | GetLineSettings | InstallUserFont | SetColor |
| ClearDevice | GetMaxColor | Line | SetFillPattern |
| ClearViewPort | GetMaxMode | LineRel | SetFillStyle |
| CloseGraph | GetMaxX | LineTo | SetGraphBufSize |
| DetectGraph | GetMaxY | MoveRel | SetGraphMode |
| Drawpoly | GetModeName | MoveTo | SetLineStyle |
| Ellipse | GetModeRange | OutText | SetPalette |
| FillEllipse | GetPalette | OutTextXY | SetRGBPalette |
| FillPoly | GetPaletteSize | PieSlice | SetTextJustify |
| FloodFill | GetPixel | PutImage | SetTextStyle |
| GetArcCoords | GetTextSettings | PutPixel | SetUserCharSize |
| GetAspectRatio | GetViewSettings | Rectangle | SetViewPort |
| GetBkColor | GetX | RegisterBGIdriver | SetVisualPage |
| GetColor | GetY | RegisterBGIfont | SetWriteMode |
| GetDefaultPalette | GraphDefaults | RestoreCrtMode | TextHeight |
| GetDriverName | GraphErrorMsg | Sector | TextWidth |
| GetFillPattern | GraphResult | SetActivePage | |

# 10.7. Structure de données

- Tableau : ARRAY[...] of ... ;
- Enregistrement: RECORD...END;
- Ensemble: SET OF... ;

# 10.8. Type de variables

- byte : 0..255 (8 bits non signé)
- shortint: -128..127 (8 bits signé)
- word: 0..65535 (16 bits non signé)
- integer: -32768..32767 (16 bits signé)
- longint: -2147483648..2147486647 (32 bits signé)
- real: 2.9e-39..1.7e38 (64 bits signé) 11-12 chiffres
- double 5.0e-324..1.7e308 15 chiffres

- boolean : booléen qui vaut *true* ou *False*
- file...of : fichier de...
- string : chaîne de caractère (maximum 255)
- string[num] : chaîne de caractère de longueur num
- char : caractère (les 256 caractères ASCII)
- pointer : pointeur ;

## 10.9. Exemple de déclaration des constantes, variables et types

```
Const
      Euro = 6.55957 ;
      Com1 = $3f8 ;           { valeur hexadécimale }
Type
      Lettres = set of 'A'..'Z';
      Coord = record
                  Nom : string[15];
                  Prenom : string[15];
                  Age: byte ;
                  Remarque : string;
               End;
Var
      Ok :  boolean ;
      x,y   : real ;
      maj : lettres ;
      i,j,k: integer ;
      agenda : file of coord ;
      personne : coord ;
      c : Char;
      chaine : string ;
      vecteur1 : array[1..10] of real ;
      coule : array[1..10,'a'..'j'] of boolean ;
```

# 10.10. Table ASCII standard (7 bits)

| DEC | HEX | CHAR | | DEC | HEX | CHAR | DEC | HEX | CHAR | DEC | HEX | CHAR |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 00 | | NULL | 32 | 20 | | 64 | 40 | @ | 96 | 60 | ' |
| 1 | 01 | ☺ | SOH | 33 | 21 | ! | 65 | 41 | A | 97 | 61 | a |
| 2 | 02 | ☻ | STX | 34 | 22 | " | 66 | 42 | B | 98 | 62 | b |
| 3 | 03 | ♥ | ETX | 35 | 23 | # | 67 | 43 | C | 99 | 63 | c |
| 4 | 04 | ♦ | EOT | 36 | 24 | $ | 68 | 44 | D | 100 | 64 | d |
| 5 | 05 | ♣ | ENQ | 37 | 25 | % | 69 | 45 | E | 101 | 65 | e |
| 6 | 06 | ♠ | ACK | 38 | 26 | & | 70 | 46 | F | 102 | 66 | f |
| 7 | 07 | • | BEL | 39 | 27 | ' | 71 | 47 | G | 103 | 67 | g |
| 8 | 08 | ◘ | BS | 40 | 28 | ( | 72 | 48 | H | 104 | 68 | h |
| 9 | 09 | ○ | TAB | 41 | 29 | ) | 73 | 49 | I | 105 | 69 | i |
| 10 | 0A | ◙ | LF | 42 | 2A | * | 74 | 4A | J | 106 | 6A | j |
| 11 | 0B | ♂ | VT | 43 | 2B | + | 75 | 4B | K | 107 | 6B | k |
| 12 | 0C | ♀ | FF | 44 | 2C | , | 76 | 4C | L | 108 | 6C | l |
| 13 | 0D | ♪ | CR | 45 | 2D | - | 77 | 4D | M | 109 | 6D | m |
| 14 | 0E | ♫ | SO | 46 | 2E | . | 78 | 4E | N | 110 | 6E | n |
| 15 | 0F | ☼ | SI | 47 | 2F | / | 79 | 4F | O | 111 | 6F | o |
| 16 | 10 | ► | DLE | 48 | 30 | 0 | 80 | 50 | P | 112 | 70 | p |
| 17 | 11 | ◄ | DC1 | 49 | 31 | 1 | 81 | 51 | Q | 113 | 71 | q |
| 18 | 12 | ↕ | DC2 | 50 | 32 | 2 | 82 | 52 | R | 114 | 72 | r |
| 19 | 13 | ‼ | DC3 | 51 | 33 | 3 | 83 | 53 | S | 115 | 73 | s |
| 20 | 14 | ¶ | DC4 | 52 | 34 | 4 | 84 | 54 | T | 116 | 74 | t |
| 21 | 15 | § | NAK | 53 | 35 | 5 | 85 | 55 | U | 117 | 75 | u |
| 22 | 16 | ▬ | SYN | 54 | 36 | 6 | 86 | 56 | V | 118 | 76 | v |
| 23 | 17 | ↨ | ETB | 55 | 37 | 7 | 87 | 57 | W | 119 | 77 | w |
| 24 | 18 | ↑ | CAN | 56 | 38 | 8 | 88 | 58 | X | 120 | 78 | x |
| 25 | 19 | ↓ | EM | 57 | 39 | 9 | 89 | 59 | Y | 121 | 79 | y |
| 26 | 1A | → | SUB | 58 | 3A | : | 90 | 5A | Z | 122 | 7A | z |
| 27 | 1B | ← | ESC | 59 | 3B | ; | 91 | 5B | [ | 123 | 7B | { |
| 28 | 1C | ∟ | FS | 60 | 3C | < | 92 | 5C | \ | 124 | 7C | \| |
| 29 | 1D | ↔ | GS | 61 | 3D | = | 93 | 5D | ] | 125 | 7D | } |
| 30 | 1E | ▼ | RS | 62 | 3E | > | 94 | 5E | ^ | 126 | 7E | ~ |
| 31 | 1F | ▲ | US | 63 | 3F | ? | 95 | 5F | _ | 127 | 7F | |

DEC = Décimal HEX = Hexadécimal CHAR = Caractère

# 10.11. Table ASCII (seconde moitié) étendue (8 bits)

| DEC | HEX | CHAR | DEC | HEX | CHAR | DEC | HEX | CHAR | DEC | HEX | CHAR |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 128 | 80 | Ç | 160 | A0 | á | 192 | C0 | └ | 224 | E0 | α |
| 129 | 81 | ü | 161 | A1 | í | 193 | C1 | ┴ | 225 | E1 | β |
| 130 | 82 | é | 162 | A2 | ó | 194 | C2 | ┬ | 226 | E2 | χ |
| 131 | 83 | â | 163 | A3 | ú | 195 | C3 | ├ | 227 | E3 | π |
| 132 | 84 | ä | 164 | A4 | ñ | 196 | C4 | ─ | 228 | E4 | Σ |
| 133 | 85 | à | 165 | A5 | Ñ | 197 | C5 | ┼ | 229 | E5 | σ |
| 134 | 86 | å | 166 | A6 | ā | 198 | C6 | ╞ | 230 | E6 | µ |
| 135 | 87 | ç | 167 | A7 | ō | 199 | C7 | ╟ | 231 | E7 | τ |
| 136 | 88 | ê | 168 | A8 | ¿ | 200 | C8 | ╚ | 232 | E8 | φ |
| 137 | 89 | ë | 169 | A9 | ⌐ | 201 | C9 | ╔ | 233 | E9 | θ |
| 138 | 8A | è | 170 | AA | ¬ | 202 | CA | ╩ | 234 | EA | Ω |
| 139 | 8B | ï | 171 | AB | ½ | 203 | CB | ╦ | 235 | EB | δ |
| 140 | 8C | î | 172 | AC | ¼ | 204 | CC | ╠ | 236 | EC | ∝ |
| 141 | 8D | ì | 173 | AD | ¡ | 205 | CD | ═ | 237 | ED | ∅ |
| 142 | 8E | Ä | 174 | AE | « | 206 | CE | ╬ | 238 | EE | ∈ |
| 143 | 8F | Å | 175 | AF | » | 207 | CF | ╧ | 239 | EF | ∩ |
| 144 | 90 | É | 176 | B0 | ░ | 208 | D0 | ╨ | 240 | F0 | ≡ |
| 145 | 91 | æ | 177 | B1 | ▒ | 209 | D1 | ╤ | 241 | F1 | ± |
| 146 | 92 | Æ | 178 | B2 | ▓ | 210 | D2 | ╥ | 242 | F2 | ≥ |
| 147 | 93 | ô | 179 | B3 | │ | 211 | D3 | ╙ | 243 | F3 | ≤ |
| 148 | 94 | ö | 180 | B4 | ┤ | 212 | D4 | ╘ | 244 | F4 | ⌠ |
| 149 | 95 | ò | 181 | B5 | ╡ | 213 | D5 | ╒ | 245 | F5 | ⌡ |
| 150 | 96 | û | 182 | B6 | ╢ | 214 | D6 | ╓ | 246 | F6 | ÷ |
| 151 | 97 | ù | 183 | B7 | ╖ | 215 | D7 | ╫ | 247 | F7 | ≈ |
| 152 | 98 | Ÿ | 184 | B8 | ╕ | 216 | D8 | ╪ | 248 | F8 | ° |
| 153 | 99 | Ö | 185 | B9 | ╣ | 217 | D9 | ┘ | 249 | F9 | ● |
| 154 | 9A | Ü | 186 | BA | ║ | 218 | DA | ┌ | 250 | FA | · |
| 155 | 9B | ¢ | 187 | BB | ╗ | 219 | DB | █ | 251 | FB | √ |
| 156 | 9C | £ | 188 | BC | ╝ | 220 | DC | ▄ | 252 | FC | $^{n}$ |
| 157 | 9D | ¥ | 189 | BD | ╜ | 221 | DD | ▌ | 253 | FD | $^{2}$ |
| 158 | 9E | ž | 190 | BE | ╛ | 222 | DE | ▐ | 254 | FE | ■ |
| 159 | 9F | ƒ | 191 | BF | ┐ | 223 | DF | ▀ | 255 | FF | |

DEC = Décimal HEX = Hexadécimal CHAR = Caractère

# Notes personnelles