Support de cours Programmation Pascal

Dr. ASSIE Brou Ida

Table des matières

Chapitre 1 : Introduction et principe élémentaire, langage de programmation Pascal

Un programme est une suite d'instructions définissant des opérations à réaliser sur des données. Pour écrire des programmes, on se sert d'une notation, appelée langage de programmation. Il existe plusieurs types de langages de programmation allant du plus rudimentaire (ceux qui sont propres du jeu d'instructions de l'ordinateur, les langages d'assemblage) au plus évolué (langages de plus haut niveau, permettant de gérer la complexité des problèmes traités grâce à la structuration).

I- Introduction et principe élémentaire aux langages de programmation

Un langage est composé des éléments suivants :

- Un **ensemble de caractères**. Dans la langue française, il s'agit des lettres de l'alphabet, minuscules et majuscules et des signes de ponctuation.
- Un **vocabulaire**. Un mot étant un ensemble ordonné de caractères, un vocabulaire défini l'ensemble des mots autorisés. Dans la langue française, ce sont les mots du dictionnaire, en incluant les différentes possibilités de conjugaison et les accords féminin ou pluriel.
- Une **syntaxe**, autrement dit un ensemble de règles grammaticales. Dans la langue française, nous avons notamment des règles de conjugaison pour accorder le verbe et le sujet, des règles pour positionner l'article par rapport au nom, des règles pour accorder l'adjectif avec le nom auquel il se rapporte, des règles pour introduire des propositions subordonnées, etc.
- Une **sémantique**, autrement dit des règles de bon sens.
- Il existe également dans les langages ce qu'on appelle une **pragmatique**, qui exprime le fait qu'il doit normalement y avoir une cohérence dans les idées.

En informatique, les langages ont également un ensemble de caractères (un alphabet), un vocabulaire et un ensemble de règles syntaxiques qu'il faut respecter pour réaliser un programme.

- **Remarque** : Un programme écrit dans un langage informatique n'est qu'une suite de mots et n'est donc pas directement compréhensible par le processeur. Il faut donc transformer cette suite de mots en une suite d'opérations élémentaires que peut exécuter le processeur. Cette étape s'appelle la **compilation**. C'est lors de la compilation que les erreurs de syntaxe sont détectées. Ensuite, il faut tester le programme et vérifier si celui-ci réalise bien le traitement désiré, autrement dit si sa sémantique est correcte.

II- Langage de programmation Pascal

Un programme Pascal respecte toujours la structure suivante :

Program « Nom_programme » **;**

{Définition éventuelle des constantes et des types ;}

{Début de la déclaration des variables}

Var nom_variable1 : type1, nom_variable2, … : type2 ; *{Il peut y en avoir autant qu'on veut}*

{Fin de la déclaration des variables}

Begin *{Début du corps du programme}*

Instruction1 ;

Instruction2 ;

... ;

End. *{Fin du programme}*

- « program », « type», « var», « begin » et « end» sont des mots du vocabulaire du langage Pascal.
- Nom_programme, nom_variable1, nom_variable2 sont des identificateurs, au même titre que les noms propres de la langue française.
- type1 et type2 sont des types parmi ceux autorisés. Nous y revenons au chapitre 2.
- instruction1 et instruction2 sont des instructions quelconques. Nous en verrons quelques-unes dans ce chapitre.

L'alphabet du Pascal est l'ensemble des caractères du clavier ; mais, il est important de noter, avant de poursuivre, que l'espace et le retour à la ligne sont des séparateurs et que la syntaxe en autorise autant qu'on le désire. En conséquence, on peut aérer les programmes pour en faciliter la lecture. D'autre part, tout ce qui est entre accolades est considéré comme un commentaire et est ignoré par le compilateur. Cela va nous servir pour commenter nos programmes et expliquer chaque étape.

Chapitre 2 : Eléments de base du langage

I- Types, constantes, variables, opérateurs

1. Type

Un type est un domaine de définition. Il existe notamment les types suivants :

- **Integer** : tous les entiers entre -32768 et +32767 (s'ils sont codés sur 16 bits).
- **Real** : les réels, codés sur 4 octets ou parfois sur 8.
- **Char** : les caractères du clavier.
- **Boolean** : c'est un ensemble formé des 2 éléments suivants {true; false}
- **String** : ce type est utilisé pour représenter les chaînes de caractères. 'bonjour' est une chaîne de caractères.

2. Constantes

Une constante est désignée par un identificateur ou nom et une valeur, qui sont fixés en début de programme, entre les mots clés CONST et VAR. La valeur ne peut pas être modifiée, et ne peut pas être une expression. Sa syntaxe est la suivante :

CONST

« Nom_Constante » = « valeur_constante » ;

3. Variable

Une variable, en informatique, a le même sens qu'en mathématique, c'est-à-dire, une variable a les caractéristiques suivantes :

- un **identificateur** ou **nom**. Un identificateur doit commencer par une lettre de l'alphabet, suivie ou non de signes alphanumériques. Par exemple, x, x1 sont des noms de variable possibles.
- un **type** parmi ceux cités plus haut
- une **valeur**

- une **adresse en mémoire**. On ne se servira pas directement des adresses mémoires des variables, mais il est tout de même bon de savoir qu'elles existent, car elles nous permettront d'expliquer certaines notions des derniers chapitres.

NB : Pour différencier le nom des variables du type caractère des caractères eux-mêmes, on met chaque caractère entre 2 apostrophes. Par exemple : 'x' est un caractère, x est une variable.

4. Opérateurs

Il existe de nombreux opérateurs fonctions de tous les types. Nous classons ci-dessous les opérateurs par rapport aux domaines de définition des opérandes et du résultat :

- **Integer x Integer --> Integer** : +, -, *, div, mod
  - x div y est le résultat entier de la division de x par y. Par exemple 5 div 2 = 2
  - x mod y est le résultat du reste de la division entière de x par y. Par exemple, 5 mod 2 = 1
- **Real x Real --> Real** : +, -, *, /
- **Real --> Integer** : trunc, round
  - trunc (x) est la valeur entière de x. Exemple, trunc (4.8) = 4
  - round (x) est la valeur entière approchée de x. Exemple, round (4.8) = 5
- **Real --> Real** : cos, sin, tan, ln, sqrt, abs
  - sqrt (x) est la racine carrée de x *x
- **Char --> Char** : succ, pred
  - succ (x) est le caractère qui suit x dans la table ASCII qui code les caractères.
  - pred (x) est le caractère qui précède x " " " " "
- **Char --> integer** : ord
  - ord (x) donne le rang du caractère x dans la table ASCII.
- **Integer --> Char** : chr
  - chr (x) donne le caractère qui est codé par le nombre x.
- **Real x Real --> Boolean** : >, <, =, >=, <=, <>
  - Les opérateurs de comparaison fournissent un résultat nécessairement vrai ou faux, ce qui correspond au type booléen.

II- Instructions

Une instruction spécifie une opération ou un enchainement d'opérations à exécuter sur des objets. Les instructions sont séparées par des ; et sont exécutées séquentiellement, c'est-à-dire l'une après l'autre, depuis le BEGIN jusqu'au END.

1. Instruction d'affectation

L'instruction d'affectation a pour but d'affecter (ou de donner) une valeur à une variable. Sa syntaxe est la suivante :

« Nom_variable » := « expression » ;

- Nom_variable est le nom d'une variable.
- L'expression est quelconque mais son évaluation doit donner une valeur du même type que celui de la variable.

2. Instructions d'Entrées-Sorties

Pour afficher des caractères à l'écran, il existe deux instructions :

a- Instruction de sortie : « write (liste d'expressions) »

Liste d'expressions représente une ou plusieurs expressions que l'on veut afficher à l'écran.

Exemple : x := 5 ; { On affecte 5 à la variable x, supposée de type integer }

- **write** ('Bonjour chers amis…') ;
- **write** (10, 3*(x-2), '8') ;

A l'exécution de ce code, à l'écran l'on aura :

Bonjour chers amis… 10 9 8

NB : writeln agit exactement comme **write**, mais ajoute un retour à la ligne après le dernier affichage.

b- Instruction d'entrée : « read (nom_variable) ; »

Nom_variable doit être une variable définie dans la déclaration des variables.

A l'exécution du programme, lorsque l'instruction « read » est rencontrée, l'utilisateur doit taper une valeur, puis la touche return (ou entrée). La valeur tapée est donnée à la variable nom_variable et le programme se poursuit.

NB : readln agit exactement comme **read**, mais ajoute un retour à la ligne.

c- Instruction composée

L'instruction composée sert à grouper des instructions dans un même ensemble.

Sa syntaxe est la suivante :

begin
instruction1 ;
instruction2 ;
…
end

III- Structure de contrôle

1- Instruction conditionnelle « if .... then.... else »

Elle sert à effectuer un traitement uniquement lorsqu'une condition bien déterminée est satisfaite. Sa syntaxe est la suivante :

if « expression booléenne » **then «** instruction1 » **else «** instruction2 » ;

NB : if, **then** et **else** sont des mots du langage Pascal.

Ainsi, si l'expression booléenne est évaluée à vrai, c'est instruction1 qui est effectuée, et dans le cas contraire, c'est à dire si l'expression booléenne est évaluée à false, c'est instruction2 qui est effectuée.

Remarque : La partie « else instruction2" est facultatif et on peut emboîter les instructions conditionnelles.

- **Exemple** : Soient x, y et z des variables entières. Ecrire un programme qui permet d'afficher le plus grand des nombres entiers. On a :

```
if (x > y) and (x > z) then writeln('x est le plus grand')
else writeln('x n"est pas le plus grand') ;
```

2- Instruction itérative « for.... »

La boucle for permet d'effectuer des itérations. Sa syntaxe est la suivante :

for « nom_de_variable » := « expression1 » **to** « expression2 » **do** instruction1 ;

La variable de boucle « nom_de_variable » prend tour à tour toutes les valeurs comprises entre valeur1 et valeur2 (valeur1 et valeur2 incluses) définies respectivement par les valeurs de expression1 et expression2. L'instruction « instruction1 » est exécutée autant de fois. La variable « nom_de_variable » est initialisée avec valeur1, puis « instruction1 » est exécutée, puis "nom_de_variable" prend la valeur qui suit valeur1 et « instruction1 » est à nouveau exécutée etc.

- **Exemple** : Ecrire un programme pour l'exécution d'un compte à rebours.

```
program compterebours;
var i : integer;
begin
  for i := 1 to 10 do
  writeln (i);
end.
```

3- Instruction « while... »

Cette instruction signifie « tant que ». Elle permet de répéter l'exécution d'une instruction d'une boucle. Sa syntaxe est la suivante :

while (expression booléenne) **do** instruction ;

Tant que l'expression booléenne est évaluée à vrai, l'instruction est exécutée.

- **Remarque** : Les variables de l'« expression » doivent être initialisées avant le while, pour que au premier passage l'« instruction » puisse être évaluée. Le « while » continue de boucler tant que B n'est pas faux. Pour éviter une boucle infinie, qui

plante le programme, il faut obligatoirement que dans « instruction » il y ait une sous-instruction rendant « expression » fausse à un moment donné.

- **Exemple** : Ecrire un programme qui demande un nombre à l'utilisateur et vérifie qu'il respecte les conditions imposées. On :

```
write (''Tapez au clavier un nombre compris entre 1 et 100');
read ( N );
while ( (N<1) or (N>100) ) do
  begin
        write ('Donnez-moi un autre nombre ! ');
        read ( N );
  end ;
```

4- Instruction « repeat...until... »

Cette instruction signifie répéter ... jusqu'à .... Elle permet comme le while de répéter l'exécution d'une instruction de boucle. Sa syntaxe est la suivante :

repeat « instruction » **until** (expression booléenne) ;

« Instruction » est exécutée, puis « expression booléenne » est évaluée. Si « expression booléenne » est vraie, alors on s'arrête, sinon on recommence.

IV- Exercices

1- Ecrire un programme qui demande à l'utilisateur les coordonnées de 2 points distincts du plan et qui affiche les coordonnées du point milieu.

2- Quel est le résultat à l'écran du programme suivant :

```
program simple;
var x,y : integer;
begin
x := 1; y := 2;
x := x + 1; y := y + x;
writeln ( 'y vaut :', y');
```

end;

3- Ecrire un programme qui demande à l'utilisateur une valeur pour U0, r et n et qui affiche la énième valeur de la suite arithmétique définie par U0 et Un+1 = Un + r. (On rappelle la propriété : Un = U0 + n.r)
4- Ecrire un programme qui demande à l'utilisateur les valeurs de 2 entiers x,y, qui permute leur valeur et qui les affiche.

Chapitre 3 : Procédures – Fonctions

Une procédure se déclare comme une fonction, sauf qu'elle n'a pas de type, ne renvoie pas de valeur et la déclaration commence par "procédure" suivi de l'identificateur.

Une fonction est une procédure qui renvoie un résultat, de manière à ce qu'on puisse l'appeler dans une expression.

Une procédure ou une fonction doit être déclarée avant le bloc principal et avant toute procédure ou fonction y faisant référence.

I- Procédures

Une procédure est un sous-programme qui permet de découper un programme en plusieurs morceaux.

Chaque procédure définit une nouvelle instruction, que l'on peut appeler en tout endroit du programme.

NB : Une procédure n'a pas de type, ne renvoie pas de valeur et la déclaration commence par "procedure" suivi de l'identificateur.

1. Procédures sans paramètre

Il s'agit simplement de donner un nom à un groupe d'instructions. Ensuite, l'appel de ce nom à divers endroits du programme provoque à chaque fois l'exécution de ce groupe d'instructions.

La syntaxe de déclaration est la suivante :

procedure Nom_Procedure;
var *(Déclaration des variables locales)*
begin
(blocs d'instructions)
end;

NB : Les variables locales d'une procédure ne sont utiles que dans cette procédure et ne peuvent pas être en dehors de celle-ci. Une variable déclarée localement n'existe que pendant l'exécution de la procédure, et ne sert que à cette procédure.

Le programme principal n'a jamais accès à une variable locale de procédure. Une procédure n'a jamais accès à une variable locale d'une autre procédure et améliore la lisibilité du programme.

- **Exemple :** Ecrire un programme qui permet d'échanger 2 entiers x, y à partir d'une procédure.

```
PROGRAM exemple1;
VAR x, y, t : integer;

{ Declaration de la procedure Echange_xy }
PROCEDURE Echange_xy;
BEGIN
    { Corps de la procedure }
    t := x; x := y; y := t;
END;

BEGIN
    { Programme principal }
    x := 3; y := 4;
    writeln (x, ' ', y);
    Echange_xy;   { 1er appel de la procedure }
    writeln (x, ' ', y);
    Echange_xy;   { 2eme appel de la procedure }
    writeln (x, ' ', y);
END.
```

2. Procédures avec paramètre

Un paramètre représente une valeur que la procédure vous attend à fournir lorsque vous l'appelez. La déclaration de la procédure définit ses paramètres.

La syntaxe de déclaration est la suivante :

procedure Nom_Procedure (Parametre1 :type1 ;... ; Parametre n :type n); (*paramètres)*
var *(Déclaration des variables locales)*
begin
(blocs d'instructions)
end;

- **Exemple :** Ecrire un programme qui permet de faire un produit de 2 réels x, y à partir d'une procédure avec paramètre.

```
PROGRAM exemple5bis;
VAR a, b, c, d : real;

PROCEDURE Produit (x, y : real; var z : real);  { parametres }
BEGIN
    z := x * y;
END;

BEGIN
    write ('a b ? '); readln (a, b);

    Produit (a, b, c);                          { passage de }
    Produit (a-1, b+1, d);                      { parametres }

    writeln ('c = ', c, ' d = ', d);
END.
```

Il y a deux sortes de passage de paramètres : le passage par valeur et le passage par référence.

- ✓ **Passage par valeur** : à l'appel, le paramètre est une variable ou une expression. C'est la valeur qui est transmise, elle sert à initialiser la variable correspondante dans la procédure.
- ✓ **Passage par référence** : à l'appel, le paramètre est une variable uniquement. C'est l'adresse mémoire (la référence) de la variable qui est transmise, non sa valeur. La variable utilisée dans la procédure est en fait la variable de l'appel, mais sous un autre nom.

NB : C'est le mot-clé « var » qui dit si le passage se fait par valeur (pas de var) ou par référence (présence du var).

- **<u>Remarque</u>** : - On peut très bien appeler une procédure P1 depuis une procédure P2, mais il faut que la procédure P1 aie été déclarée avant la procédure P2. Par exemple, si nous reprenons l'exemple « Ecrire un programme qui permet d'échanger 2 entiers x, y à partir d'une procédure ».

```
PROGRAM exemple2;
VAR x, y, t : integer;

PROCEDURE Affiche_xy;
BEGIN
    writeln (x, ' ', y);
END;

PROCEDURE Echange_xy;
BEGIN
    t := x; x := y; y := t;
    Affiche_xy;
END;

BEGIN
    x := 3; y := 4;
    Affiche_xy;
    Echange_xy;
    Echange_xy;
END.
```

- On peut aussi appeler une procédure depuis elle-même : c'est la récursivité.

II- Fonctions

Une fonction est une procédure qui renvoie un résultat, de manière à ce qu'on puisse l'appeler dans une expression. Sa syntaxe de déclaration est la suivante :

```
FUNCTION nom_fonction : type_resultat;
BEGIN
    { ... corps de la fonction ... }

    { Résultat de la fonction, du type type_resultat }
    nom_fonction := expression;
END;
```

L'instruction « nom_fonction :=expression ; » renvoie le résultat de la fonction. Tout ce que l'on a dit sur le paramétrage des procédures reste valable pour les fonctions.

- **Exemple** :

```
program equilateral;
var     cote1, cote2, cote3 : real;
        x1,y1, x2, y2, x3, y3 : real;

{ ********* Début de la déclaration de la fonction distance ********** }
function distance (x1 : real; y1 : real; x2 : real; y2 : real ) : real;
{ Pas besoin de variable locale }
begin
        distance := sqrt ( sqr(x2-x1) + sqr(y2-y1) );
end;
{ ************* fin de la déclaration de la fonction **************}

begin  (* Corps principal du programme. *)
        writeln ('Entrez les coordonnées des 3 points du triangle ');
        readln(x1); readln(y1); readln(x2); readln(y2); readln(x3); readln(y3);
        cote1 := distance (x1, y1, x2, y2);          { Appel de la fonction distance }
        cote2 := distance (x2, y2, x3, y3);          { Appel de la fonction distance }
        cote3 := distance (x3, y3, x1, y1);          { Appel de la fonction distance }
        if (cote1 = cote2) and (cote2 = cote3)
                then writeln ('Le triangle est équilatéral')
                else writeln ('Le triangle n''est pas équilatéral');
end.
```

IV- Exercices

1. Ecrire un programme qui calcule les 10 premières factorielles en utilisant une fonction.
2. Ecrire un programme qui calcule les 10 premières factorielles en utilisant une procédure.

Chapitre 4 : Types énumérés, tableaux et enregistrements

Jusqu'à présent, nous nous sommes intéressés aux variables et à la structure d'un programme.

Ce chapitre fera l'objet de la connaissance de la nature des valeurs que peut prendre une variable dans un ensemble de variables. Par exemple, pour représenter un mois de l'année on peut décider de prendre des entiers avec 1 pour janvier, 2 pour février, etc. ou des couleurs (jaune, rouge, vert, bleu, …). Il s'agit donc de la notion de types :

- Énumérés,
- Tableaux,
- Enregistrements.

I- Types énumérés

1- Exemple de problème

- **Problème :** Comment coder les 7 jours de la semaine. Est-ce que l'on prend les entiers de 1 à 7 ou de 0 à 6 ? Si oui, le premier numéro (1 ou 0) représente-t-il lundi ou un autre jour ?
- **Principe d'un type énuméré :** Les types énumérés permettent alors à l'utilisateur de définir son propre type en indiquant explicitement l'ensemble de ses valeurs.

2- Définition d'un type énuméré

Un type énuméré est un type défini par l'utilisateur en indiquant les valeurs que peuvent prendre les variables de ce type.

La syntaxe de déclaration d'un type énuméré est la suivante :

Type « Nom_type » = (Valeur 1, Valeur 2, ..., Valeur n) ;

NB : Le type énuméré est codé sur un entier. Il s'agit d'un type ordinal et on peut utiliser les opérateurs pred, succ et ord.

- **Exemple :** En reprenant le problème ci-dessus, nous pouvons définir le type énuméré semaine comme suit :

  Type Semaine = (Lundi, Mardi, Mercredi, Jeudi, Vendredi, Samedi, Dimanche) ;

- **Remarque :** Pour plus de lisibilité du code, le programmeur utilisera le nom symbolique pour désigner une valeur particulière du type et c'est le compilateur qui aura la charge de définir un codage pour ces noms symboliques.

II- Enregistrements

Un type enregistrement permet de définir des types structurés qui ne peuvent pas être représentés par les types élémentaires que nous avons déjà vus tels que Entier, Booléen, Réel, …. Les données d'un type enregistrement ne peuvent pas être rangées (logiquement) dans un tableau.

Par exemple, une date composée d'un numéro de jour (1..31), d'un numéro de mois (1..12) et d'une année est un exemple d'enregistrement.

1- Définition d'un type enregistrement

Un type enregistrement permet de manipuler simultanément un ensemble de données logiquement reliées. C'est le produit cartésien de plusieurs types T_1, T_2, ..., Tn.

La syntaxe de déclaration d'un type enregistrement est la suivante :

```
type    nomdelastructure = record
                nomduchamp1 : type1;
                nomduchamp2 : type2;
                ...
        end;
```

- **Exemple** : Les types enregistrements « personne » et « point » sont définis comme suit :

```
type    t_personne = record
                nom : string;
                age : integer;
        end;
        t_point = record
                x : integer;
                y : integer;
        end;
```

2- Déclaration d'une variable de type enregistrement

Un type enregistrement est un type qui permet de déclarer des variables. Soit le type enregistrement « personne », une variable « joueur1 » de type « personne » est définie par :

```
var     joueur1, joueur2 : t_personne;
```

3- Manipulation d'une variable de type enregistrement

L'opérateur d'affectation permet de manipuler les variables de type enregistrement. Il consiste à copier toutes les composantes de l'enregistrement.

Nous avons pour la manipulation de l'enregistrement « personne », l'exemple qui suit :

```
writeln ('Entrez votre nom :');
readln ( joueur1.nom );
```

III- Tableaux

Dans tout ce qui précède, nous avons utilisé des variables simples d'un type particulier. Cependant, il est fréquent dans certains cas de devoir manipuler un grand nombre de données du même type auxquelles on va appliquer les mêmes traitements.

Par exemple, on peut considérer un ensemble de notes d'informatique reçues par les étudiants d'un niveau d'étude donné. Pour les représenter, il existe un type de données particulier appelé tableau de notes. Ceci permet de regrouper sous un même nom l'ensemble de notes d'informatique des étudiants.

La note d'informatique d'un étudiant X sera obtenue en utilisant un indice.

1- Définition d'un tableau

Un tableau est un ensemble d'éléments de même type désignés par un identificateur unique. Chaque élément est repéré par un indice précisant sa position dans l'ensemble (le premier élément est repéré par l'indice 0). Un tableau, c'est aussi une variable qui permet de rassembler sous un même nom (celui de la variable), un nombre fini d'éléments ayant tous le même type.

Un tableau est donc caractérisé par :

- ✓ le type des éléments qu'il contient ;
- ✓ les indices pour accéder à ces éléments.

Lorsqu'un élément particulier d'un tableau est désigné en précisant son indice, on parle alors de **tableau à une dimension**. Sinon, lorsqu'il est précisé par plusieurs indices, on parle de **tableau à plusieurs dimensions**.

2- Tableau à une dimension

Un tableau à une dimension est un tableau dont on accède aux éléments en utilisant un seul indice.

La syntaxe de déclaration d'un tableau à une dimension est la suivante :

```
Type
    nom_de_variable : array [ i1..i2 ] of nom_de_type;
```

- **Exemple** : Déclaration du tableau de notes d'informatique des n étudiants de Licence 1

```
Type
    NotesInfo = array[1...n] of real ;
```

Pour accéder à la 4eme note d'informatique dans le tableau « NotesInfo », il faut préciser le nom du tableau suivi de l'indice 4 entre les crochets comme suit :

```
NotesInfo[4] ;
```

- **Remarque** : Pour sélectionner un élément d'un tableau en fonction de son indice ; il faut préciser le nom du tableau suivi de son indice entre les crochets :

```
Nom_tableau[Indice] ;
```

3- Chaine de caractères (String)

Une chaîne de caractères est considérée comme un tableau de caractères avec des opérateurs supplémentaires tels que :

- « longueur », fonction qui permet d'obtenir le nombre de caractères de la chaîne de caractères.
- « Concaténation » noté « + » permettant de concaténer des chaines de caractères.

Sa syntaxe est la suivante :

```
string [m] ;
```

Par définition, les indices valides pour une chaîne de caractères sont des entiers compris entre 1 et la longueur de la chaîne. Chaque caractère peut être obtenu (en accès ou en modification) en utilisant son indice pour le sélectionner.

- **Exemple** :

```
VAR s : string[80];
BEGIN
    s := 'Le ciel est bleu.';
    writeln (s);
END.
```

4- Tableaux à plusieurs dimensions

Les tableaux peuvent avoir plusieurs dimensions. Il suffit de préciser les types qui correspondent aux nouvelles dimensions.

La syntaxe de déclaration d'un tableau à plusieurs dimensions est la suivante :

Type

Nom_tableau = array[lignes,colonnes] de Type_Élément ;

Pour accéder à un élément de ce tableau, il suffit de donner une valeur pour chaque indice, c'est-à-dire :

Tableau[Valeur Indice lignes, Valeur Indice colonnes]

Par exemple, on peut définir une matrice comme étant un tableau à deux dimensions, la première correspondant aux lignes et la seconde aux colonnes. Soit :

Type Tablo= array[1..M,1..N] of Real ;

Pour accéder à l'élément de la matrice m1, de type Tablo, se trouvant à l'intercession de la ligne 1 et la colonne 4, on écrit m1[1,4].

IV- Exercices

1. Ecrire un programme qui permet d'initialiser un tableau de 10 entiers pour qu'il contienne les valeurs suivantes :

1	2	3	4	5	6	7	8	9	10

2. Ecrire un programme qui lit 10 nombres entiers dans un tableau avant d'en rechercher le plus grand et le plus petit.

3. Ecrire un programme qui demande à l'utilisateur de lui fournir un nombre entier entre 1 et 7 et qui affiche le nom du jour de la semaine ayant le numéro indiqué (lundi pour 1, mardi pour 2, ... dimanche pour 7).
4. Ecrire un programme qui permet d'afficher le nom et prénoms, niveau d'étude, année d'inscription des étudiants d'une classe X.