Introduction au langage Pascal

Le PASCAL, créé par WIRTH au début des années 70, est un langage bien structuré et bien typé qui est utilisé pour illustrer ce cours d'algorithmique. Il possède une forte similitude avec notre pseudo-code.

1. Programme

Un programme PASCAL est au moins composé d'un entête PROGRAM et des instructions délimitées par BEGIN et END.

```
PROGRAM Exemple;
BEGIN
        writeln('Hello World');
END.
```

Notez : PASCAL n'est pas sensible à la casse.

```
program Exemple;
begin
        WRITELN('Hello World');
end.
```

2. Commentaires

Les commentaires servent au programmeur pour documenter son code. Ils seront ignorés par le compilateur lors de la compilation.

```
PROGRAM Exemple;
BEGIN
        (* ceci est
        un commentaire multi-ligne *)
        { ceci est
        un autre commentaire multi-ligne }
        writeln('Hello World');
END.
```

3. Types

Il existe quatre types de base pour les variables en PASCAL :

- INTEGER, opérations: +, -, *, div, mod
- REAL, opérations: +, -, *, /
- BOOLEAN, valeurs : TRUE/FALSE, opérations: AND, OR, NOT, XOR
  Opérations à valeur booléenne : >, <, >=, <=, =, <>
- CHAR , c := CHR(n) { ascii => char } ; n = ORD(c) {char => ascii }

Le type chaîne sera traité plus tard.

Il est possible de définir ses propres types à partir des quatre types de base via une déclaration en dessous de l'entête.

```
TYPE
      moninteger = INTEGER ; { renommer un type existant }
      jour = (lundi,mardi,mercredi,jeudi,vendredi,samedi,dimanche);
            { une suite de valeurs prédéfinies }
      byte = 0..255; { un sous-ensemble des entiers }
BEGIN
        …
END.
```

4. Déclaration, saisie et affichage des variables

Les déclarations de variables se trouvent au début du programme en dessous de l'entête, mot-clé VAR.
Saisie clavier : `readln`
Affichage à l'écran : `write` ou `writeln`.

```
PROGRAM Exemple;
VAR x: INTEGER;
BEGIN
       write('saisir x: ');
       readln(x);
       writeln('x = ', x);
END.
```

5. Constantes

On peut également introduire des constantes qui sont des valeurs fixes tout au long du programme. On déclare les constantes avant de déclarer les variables. Le type de la constante est déterminé automatiquement par le compilateur (entier si nombre sans point, réel si nombre avec point, caractères si entre apostrophes). Une constante prédéfinie particulière à retenir est MAXINT : le plus grand entier possible.

```
PROGRAM Exemple;
CONST   message = 'Salut';
       m = MAXINT;
BEGIN
       writeln(message);
       writeln('le plus grand entier représentable :', m);
END.
```

6. Affectation

L'affectation d'une valeur à une variable s'écrit variable := expression

```
PROGRAM Exemple;
VAR x, y, z: INTEGER;
BEGIN
       x := 5;
       y := 6;
       z := x + y;
       writeln(x,y,z);
END.
```

7. Structure de contrôle

7.1 Si – alors - sinon

IF condition THEN instruction1 { Attention : pas de « ; » }
ou : IF condition THEN instruction1 ELSE instruction2

```
IF cond1 THEN
        IF cond2 THEN inst1    {cond1 et cond2}
                 ELSE inst2    {cond1 et pas cond2}
ELSE
        IF cond3 THEN inst3    {pas cond1 mais cond3}
                 ELSE inst4    {ni cond1 ni cond3}
```

7.2 Selon

```
CASE expression OF
        liste_de_cas1 : instruction1;
        liste_de_cas2 : instruction2;
        .....
        liste_de_casN : instructionN ;
        ELSE instructionELSE
END
```

```
CASE a*b OF  { a et b déclarés entiers }
        0 : writeln('un des nombres est nul');
        1..9 : writeln('le produit n''a qu''un chiffre');
        10, 100, 1000: writeln('le produit est une puissance de 10');
        ELSE : writeln(a, ' x ', b, ' = ', a*b)
END
```

7.3 Tantque

WHILE expression booléenne DO instruction

```
PROGRAM Exemple;
VAR nombre, racine : REAL;
BEGIN
        writeln('entrez un réel entre 0 et 10');
        readln(nombre);
        racine := 0;
        WHILE racine*racine < nombre DO
                racine := racine + 0.1;
        writeln('la racine de ', nombre, 'vaut à peu près', racine)
END.
```

7.4 Répète jusqu'à ce que

```
REPEAT
        instruction1;
        instruction2;
        ...etc...
        instructionN
UNTIL condition ;
```

```
VAR a : INTEGER;
BEGIN
        writeln('entrez le nombre 482');
        REPEAT
                readln(a)
        UNTIL a = 482;
        writeln('c''est gentil.')
END.
```

7.5 Pour

FOR variable := début TO (ou DOWNTO) fin DO { Attention début et fin ne sont calculés qu'1 fois }
Instruction;

```
PROGRAM Exemple;
VAR letter : CHAR;
BEGIN
      FOR letter := 'Z' DOWNTO 'A' DO
            writeln(lettre);
END
```

8. Instructions composées

Plusieurs instructions peuvent être regroupées par BEGIN - END

```
PROGRAM Exemple;
VAR a, i : INTEGER;
BEGIN
      a := 1;
      FOR i := 1 TO 10 DO
      BEGIN
            writeln(a);
            a := a*i;
      END
END
```

9. Méthodes

Structure d'une entité de programme :
- Entête
- Déclaration des
  - Constantes;
  - Types ;
  - Variables;
- *Définition des méthodes (procédures et fonctions)*
- BEGIN
  - Instructions
- END

9.1 Procédures

```
PROGRAM Exemple;
VAR a,b,c,d : REAL;

PROCEDURE aff_somme(x, y : REAL);
VAR z : REAL;
BEGIN
               z := x + y;
               writeln(x ,' + ', y ,' = ', z)
END;
BEGIN   {programme principal}
       writeln('entrez 4 valeurs : ');
       readln(a,b,c,d);
       aff_somme(a,b);
       aff_somme(3,5);
       aff_somme(c+a,d)
END.
```

9.2 Fonctions

FUNCTION nom_fonction (liste_parametres) : type_de_la_fonction ;

```
PROGRAM Exemple;
VAR a, b, c : REAL;

FUNCTION MAX(x, y : REAL) : REAL;
BEGIN
       IF x >= y THEN
              MAX := x
       ELSE
              MAX := y
END;

BEGIN
       writeln('entrez deux valeurs : ');
       readln(a,b);
       c := max(a,b);
       writeln('le plus grand est ',c)
END.
```

9.3 Quelques fonctions standards

ABS : renvoie la valeur absolue
SQR : renvoie le carré
SQRT : racine carrée
SUCC : variable énumérée suivante
PRED : précédent
ROUND : arrondi à l'entier le plus proche
TRUNC : partie entière
EX : exponentielle
LN : log naturel
SIN : sinus
COS : cosinus
ARCTAN : arc tangente

10. Variables globales et locales

Les variables déclarées dans les méthodes sont locales et visibles uniquement au sein de la méthode. Les variables déclarées au niveau du programme même sont globales et visibles pour toutes les méthodes.

11. Passage par valeur et par variable

Il y a deux méthodes pour passer des variables en paramètre dans une fonction : le passage par valeur et le passage par variable. Ces méthodes sont décrites ci-dessous.

```
PROGRAM Exemple;
VAR global : Integer;

Procedure test()
VAR local : Integer;
Begin
   local  := 1;
   global := 1;
End;

Begin
   { local n'est pas visible ici. }
   global:= 5;  {Initialise global à
   5}
   test(); {Appelle la fonction. La variable globale est modifiée par la
           fonction test}
   writeln(global);   {Ici, global vaut 1 }
End.
```

11.1. Passage par valeur

La valeur de l'expression passée en paramètre est copiée dans une variable locale. C'est cette variable qui est utilisée pour faire les calculs dans la fonction appelée. Si l'expression passée en paramètre est une variable, son contenu est copié dans la variable locale. Aucune modification de la variable locale dans la fonction appelée ne modifie la variable passée en paramètre, parce que ces modifications ne s'appliquent qu'à une copie de cette dernière.

Ceci est le mode de passage par défaut (voir exemples 9.1 et 9.2).

11.2. Passage par variable

La deuxième technique consiste à passer non plus la valeur des variables comme paramètre, mais à passer les variables elles-mêmes. Il n'y a donc plus de copie, plus de variable locale. Toute modification du paramètre dans la fonction appelée entraîne la modification de la variable passée en paramètre.

Pour ce faire, ajouter le mot clé VAR avant le paramètre concerné.

```
PROGRAM Exemple ;
VAR i : integer;

Procedure test(VAR j : integer)
Begin
   {La variable j est strictement égale à la
    variable passée en paramètre.}
   j := 2;    {Ici, cette variable est modifiée.}
End;

Begin
```

```
   i := 3;  {Initialise i à 3}
   test(i); {Appelle la fonction. La variable i est passée en
             paramètres, pas sa valeur. Elle est modifiée par
             la fonction test.}
   writeln(i);   {Ici, i vaut 2.}
End.
```

12. Variable structurée de type enregistrement

Les enregistrements sont des variables structurées qui peuvent contenir plusieurs champs de type de base, structuré (enregistrement ou tableau), voir de type défini précédemment.
Il est conseillé de définir le type enregistrement dans les types au départ pour en déclarer la variable de ce type par la suite.

```
        RECORD
                champs1 : type1;
                champs2 : type2;
                .....
        END
```

```
TYPE
  TAddresse = RECORD
    nombre: Integer;
    rue: string;
    codePostal: Integer;
    ville: string;
  END;

VAR
  uneAddresse, uneAutreAddresse : TAddresse;
```

13. Tableaux

13.1 Tableaux unidimensionnels

```
        VAR nom_tableau : ARRAY [type_index] OF type_composantes
```

```
PROGRAM Exemple;
CONST dimension = 3 ;
VAR vecteur: ARRAY [1..dimension] OF REAL
      i : Integer ;
BEGIN
       FOR i := 1 TO dimension DO
              vecteur [i] := 0.0 ;
END.
```

```
PROGRAM Exemple;
TYPE jour = (lundi,mardi,mercredi,jeudi,vendredi,samedi,dimanche);
VAR
       j : jour;
       nb_heures_cours : ARRAY[jour] OF INTEGER;
BEGIN
       FOR j := lundi TO samedi DO
              nb_heures_cours[j] := 8 ;
END.
```

13.2 Matrices (tableaux de tableaux)

Trois déclarations équivalentes possibles:

1) TYPE ligne = array[1..10] of REAL;
   matrice = array[1..5] of ligne;
2) TYPE matrice = array[1..5] of array[1..10] of REAL
3) TYPE matrice = array[1..5,1..10] of REAL

```
VAR m1, m2 : matrice;
i, j : INTEGER;
```

On peut écrire :

```
m1[i,j]:= 10.0 ;   {affectation de la valeur 10 en ligne i, colonne j}
m1[i]:= m1[i+1] ;  {copie complète de la ligne i+1 sur la ligne i}
m2 := m1 ;         {copie de la matrice complète}
```

14. Les chaînes

Une chaine est gérée par le type `String`, un tableau de caractères dont la taille ne peut pas dépasser 255. Ceci est également la valeur par défaut.

```
var
  chaine1: string ;    // chaine de 255 caractères
  chaine2: string[50]; // chaine de 50 caractères
```

Les chaines doivent être délimitées par des apostrophes. Si la chaine contient elle même une apostrophe, il suffit de doubler l'apostrophe :

```
chaine1:='l''apostrophe';
```

Chaque caractère de la chaine est repéré par un index. Le premier caractère possède la position 1. La case 0 d'une chaine possèdent la longueur de celle-ci (ex: `ord(chaine1[0])` vaut 12).

L'opérateur + permet de concaténer deux chaines de caractères (type String) ou deux caractères (type Char) en une nouvelle chaine. Les opérateurs de comparaison = , <> , < , > , <= , >= peuvent également être employés avec des chaînes de caractères. Les chaines sont comparées d'après la valeur des codes ASCII qui la composent. On peut aussi comparer un type String avec un type Char, car dans ce cas ce dernier est automatiquement considéré comme une chaine de caractère dont la longueur vaut 1.

Voici quelques fonctions importantes :
`Length(S : String) : Integer;` la longueur d'une chaine de caractères, la même chose que `ord(s[0])`
`Copy(S : String; Index , Count : Integer) : String;` extraire d'une chaine une sous chaine d'une longueur donnée à partir d'une position donnée. Si l'on tente d'extraire plus de caractères qu'il n'est possible, la sous chaine est automatiquement tronquée. Si la position de l'index est supérieure à la longueur de la chaine, on obtient une chaine vide.
`Delete(S : String ; Index , Count : Integer);` suppression d'une sous chaine à partir d'une position et d'une longueur donnée
`UpperCase(S : String) : String;` mettre une chaine en majuscule
`LowerCase(S : String) : String;` mettre une chaine en minuscule

14. Tableaux dynamiques

Un tableau dynamique est déclaré sans taille. La taille sera ensuite spécifiée lors d'un appel de la function `SetLength` qui alloue la mémoire nécessaire pendant l'exécution du programme. Notez que les indices d'un tableau ainsi créé commencent toujours à 0. Il est aussi possible de remodifier la taille d'un tableau existant. Si la taille est diminuée, les derniers éléments seront

supprimés. *Attention*: Contrairement aux tableaux statiques, l'instruction B:=A entre deux tableaux dynamiques ne fait pas de copie : B pointe vers le même tableau que A.

```
Program Exemple;
Var
  A, B : Array of Integer;
Begin
  SetLength(A,10);
  A[0] := 33;
  B := A;
  A[0] := 31;
  writeln(B[0]) ; { affiche 31 ! }
End.
```