

# Pense bête de PASCAL

12 octobre 2009

## 1 Structure des programmes

La structure d'un programme en pascal est la suivante : (Rappelons que tous ce qui est écrit entre (\*) et (\*) est un commentaire, ce n'est pas pris en compte lors de l'exécution du programme).

De plus Pascal ne fait pas la distinction entre MAJUSCULES et minuscules, mais il peut être intéressant que vous choisissiez une convention qui vous permette de mieux mettre en valeur les mots clef importants. (ceci est laissé votre appréciation)

```
PROGRAM NomDuProgramme;
```

```
CONST
```

```
  (* declarations des constantes : pas à votre programme*)
```

```
TYPE
```

```
  (* declarations des types : pas à votre programme*)
```

```
VAR
```

```
  (* declarations des Variables, la syntaxe est : *)
```

```
  (* "NomDeLaVariable : Type", voir la section suivante*)
```

```
(* definitions des sous-programmes, expliqué dans le poly suivant*)
```

```
BEGIN
```

```
  (* Instructions qui forment le programme *)
```

```
END.
```

Exemple : Le programme qui ne fait rien :

```
program DoNothing;  
begin  
end.
```

## 2 Variables et structures de données

Les variables sont des " boîtes " qui peuvent contenir une valeur numérique.

## 2.1 Types

Une variable possède un type. Le type est en quelque sorte le domaine de définition de la variable. En pascal il y a 4 types :

- Integer : le types qui correspond aux entiers ( $\mathbb{Z}$ )
- Real : le types qui correspond aux réels ( $\mathbb{R}$ )
- Boolean : le type des booléens ( $\{True, False\}$ )
- Char : les caractères qui ne sont pas à votre programme

Attention : les réels peuvent être notés avec la notation dite scientifique :  $4.32e5 = 4.32 \times 10^5$ .

## 2.2 Déclaration

Toutes vos variables doivent être déclarées. Cela ce fait après le mot clef VAR dans la structure du programme en insérant des lignes de la forme :

```
NomDeLaVariable : TypesDeLaVariable ;
```

Exemple :

```
a : boolean;(*Création de la variable appelée a qui contient un booléen*)
nom_de_variable_super_long : integer ;(*Les noms de variables peuvent être à peu près n'i
b,c,d,e : real;(*Il est possible de déclarer plusieurs variables DU MÊME TYPE à la fois*)
```

## 2.3 Assignment

Une variable appelée dans une instruction sera en quelque sorte remplacée par la valeur qu'elle contient. La seule exception à cette règle est l'instruction d'affectation qui sert à modifier la valeur contenue dans la variable.

Cela ce fait avec l'instruction suivante :

```
NomDeLaVariable := Nouvelle_valeur ;
```

Exemple :

```
PROGRAM UtilisationDesVariables;
```

```
VAR
```

```
a : integer ;
```

```
b : boolean ;
```

```
c : real ;
```

```
BEGIN
```

```
a:= 3; (*a contient maintenant 3*)
```

```
b:=True; (*b contient maintenant la valeur "vrai"*)
```

```
c:= 213.7;(*c contient maintenant 213.7*)
```

```
a:= a+1; (*il est possible d'utiliser la valeur de la variable avant*)
          (*la modification pour déterminer la nouvelle valeur      *)
          (*dans le cas présent la nouvelle valeur de a est 4      *)
```

### 3 Fonctions standards

Quelques fonctions sont disponible en pascal.

#### 3.1 Les opérateurs

Opérateur	opération	type des arguments	type du résultat
+	addition	real or integer	real or integer
-	soustraction	real or integer	real or integer
*	multiplication	real or integer	real or integer
/	division réelle	real or integer	real
DIV	division entière	integer	integer
MOD	reste modulo	integer	integer
AND	le ET logique	boolean	boolean
OR	le OU logique	boolean	boolean
NOT	le NON logique	boolean	boolean
i	le test d'infériorité	real or integer	boolean
i=	le test d'infériorité ou d'égalité	real or integer	boolean
i	le test de supériorité	real or integer	boolean
i=	le test de supériorité ou d'égalité	real or integer	boolean
=	le test d'égalité	real or integer	boolean
i;	le test de différence	real or integer	boolean

#### 3.2 Les fonctions standard

Fonction	description	type de l'argument	type du résultat
abs	valeur absolue	real or integer	comme l'argument
arctan	arctan en radians	real or integer	real
cos	cosinus d'un angle en radians	real or integer	real
exp	exponentielle	real or integer	real
ln	logarithme népérien	real or integer	real
round	arrondi à l'entier le plus proche	real	integer
sin	sinus d'un angle en radians	real or integer	real
sqr	mise au carré	real or integer	comme l'argument
sqrt	racine carrée	real or integer	real
trunc	truncate (arrondi à l'entier inférieur)	real or integer	integer

### 4 Entrés - Sorties

Les entrés - sorties sont les instruction qui permettent au programme et à l'utilisateur du programme de communiquer. Rappelons qu'un programme ne fait ni plus ni moins que ce qu'on lui demande : si on ne lui demande pas d'afficher le résultat il ne le fait pas...

3 instructions qui prennent comme arguments une liste :

- write(.) - qui fait écrire au programme les élément de la liste donnée en arguments.
- writeln(.) - qui fait comme write mais en plus revient à la ligne (une fois, tout à la fin)

- `readln(.)` - qui demande au programme d'attendre que l'utilisateur lui fournisse de quoi remplir toutes les variables de la liste donnée en argument.

## 5 Tests et boucles

Ce sont les instructions qui permettent de contrôler le " flot" d'exécution.

### 5.1 le IF ... THEN ... ELSE

Permet d'effectuer un test (donc une expression à valeur booléenne) et d'effectuer des tâches différentes selon la valeur de l'expression booléenne. Il y a différentes syntaxes possibles :

- Simple :

```
if ExpressionBooléenne then
  InstructionSiVrai;
```
- avec plusieurs instructions :

```
if ExpressionBooléenne then
begin
  InstructionSiVrai1;
  InstructionSiVrai2;
  ...
  InstructionSiVraiN
end;
```
- Avec le traitement du cas où le test échoue :

```
if ExpressionBooléenne then
  InstructionSiVrai
else
  InstructionSiFaux;
```

Avec bien sûr les cas où l'on mélange le deuxième et le troisième cas. De façon générale il est toujours possible de remplacer une instruction par :

```
begin
  InstructionSiVrai1;
  InstructionSiVrai2;
  ...
  InstructionSiVraiN
end;
```

### 5.2 le FOR ... DO ...

Permet de répéter une instruction plusieurs fois en se servant d'un compteur : Si Compteur est une variable de type entière,

```
for Compteur := Début to Fin do
  instruction;
```

Cette série d'instructions va commencer par mettre la valeur de Compteur à Début, puis tant que  $Compteur \leq Fin$ , elle va effectuer instruction puis incrémenter (ajouter 1) Compteur, effectuer instruction puis incrémenter Compteur, effectuer instruction puis incrémenter Compteur...

**Remarque 1** *L'instruction peut dépendre du compteur. D'ailleurs très souvent c'est le cas en pratique.*

**Remarque 2** *On peut bien sûr remplacer l'instruction par plusieurs en utilisant "begin ... end;"*

**Remarque 3** *Attention, si l'instruction modifie la valeur contenue dans compteur cela peut avoir un comportement étrange. Notamment il est possible de créer une boucle infini (qui ne s'arrête jamais à part en faisant planter (a) le programme, (b) l'ordinateur, (c) l'ordre de l'univers (cas assez rare).*

### 5.3 le WHILE ... DO ...

Répète une instruction TANT QU'un test soit vrai. La syntaxe est la suivante :

```
while ExpressionBooléenne do
    InstructionTantQueVrai;
```

**Remarque 4** *L'instruction est effectuée APRÈS le test.*

**Remarque 5** *Attention, si l'instruction ne change rien dans les variables de l'expression booléenne on aura une boucle infinie ! De façon générale il faut être prudent lorsqu'on utilise un WHILE.*

### 5.4 le REPEAT ... UNTIL ...

Similaire au while. Répète une instruction JUSQU'À CE QU'une expression booléenne soit vraie. Syntaxe :

```
repeat
    statement1;
    statement2
until BooleanExpression;
```

**Remarque 6** *Dans ce cas pas besoin de mettre begin et end pour pouvoir utiliser plusieurs instructions.*

**Remarque 7** *Le test est effectué après la suite d'instructions.*

**Remarque 8** *Attention encore aux boucles infinies*