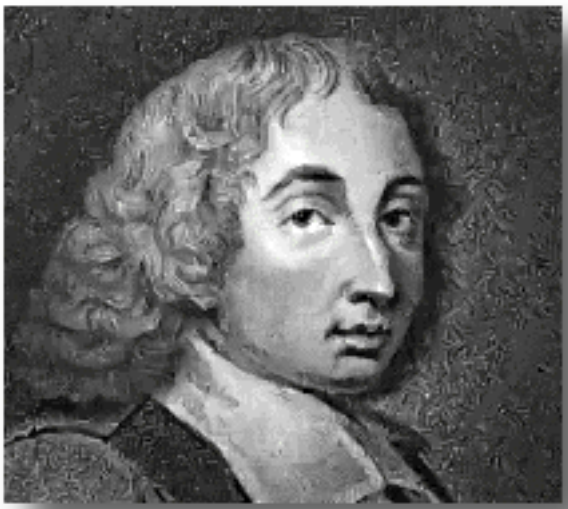


Introduction au Dec Pascal

Yann MORERE

3 juillet 2000

Résumé

Ce document contient une présentation du Dec Pascal et de ces principaux composants. Ce document est destiné aux Deug 1re année.

Table des matières

Liste des tableaux

1 Introduction

1.1 Pourquoi $\mathcal{PASCAL}$?

Beaucoup de noms de langages de programmation sont basés sur des acronymes (contraction des initiales d'un titre) par exemple :

- FORTRAN pour **FOR**mula **TRAN**slator
- COBOL pour **CO**mmon **B**usiness **O**riented **L**anguage
- C par rapport au langage qui le précédait : le B

$\mathcal{PASCAL}$ est la premier langage qui emprunte son nom à un homme de science : Blaise PASCAL, mathématicien, physicien, philosophe et écrivain français (Clermont-Ferrand, 1623 - Paris 1662) qui à dix-huit ans inventa une machine arithmétique. Il peut donc être considérer comme un des pionniers de l'informatique. C'est pour cette raison que le langage porte son nom.

1.2 Le langage

Il est le fruit de la réflexion de N. WIRTH (professeur à Zurich). En effet, il désirait un langage et assez rapide (langage compilé est plus rapide qu'un langage interprété : "BASIC"). Son développement s'est effectué dans la première moitié des années 70 et avait pour objectif

1. la création d'un langage de qualité (rigueur et facilité)
2. la gratuité du compilateur

Ce langage de programmation s'est développé dans les industries au cours de la seconde moitié des années 70.

2 Préliminaires au Dec $\mathcal{PASCAL}$

2.1 Le Jeu de Caractères Utilisable

Le jeu de caractères ASCII (American Standard Code for Information Interchange) est utilisable avec le Dec $\mathcal{PASCAL}$. [1..9],[a..z],[A..Z] et les autres caractères comme *,-,+, ?, !,etc...

Le compilateur ne fait pas la différence entre les majuscules et les minuscules. Mais il sera judicieux de prendre des noms en majuscules pour les constantes, des minuscules pour les variables et des identifiants en minuscules avec initiales en majuscules pour les fonctions et procédures.

2.2 Les symboles spécifiques

La liste des symboles spécifiques est contenue dans le tableau 1.

'	apostrophe	=	égal	.	période
:=	affectation	**	exponentiel	+	plus
[] ou (..)	crochets	>	supérieur	^ ou @	pointeur
:	2 points	≥	supérieur ou égal	;	point virgule
,	virgule	<	inférieur	..	opérateur intervalle
{} ou (**)	commentaires	≤	inférieur ou égal	%	pourcent
/	division	−	soustraction	*	multiplication
"	double cote	<>	différent	()	parenthèses

TAB. 1: Tableau des symboles spécifiques au Dec $\mathcal{PASCAL}$

2.3 Délimiteurs de Chaînes de Caractères

En Dec $\mathcal{PASCAL}$ on peut utiliser indifférement comme délimiteurs de chaînes de caractères les " et les ' :

`"chaîne de caractères"` et `'chaîne de caractères'` sont des écritures équivalentes.

2.4 Les mots reservés

Ce sont des identificateurs utilisés par le compilateur. Il s'agit des types de données, directives de compilation, séparateurs, opérateurs.

On ne doit en aucun cas redéfinir ces identificateurs.

2.5 Les identificateurs

Il s'agit des noms de variables créés à partir des lettres [a..z],[A..Z] et chiffres [0..9] ainsi que $ et _.
Il faut respecter certaines règles pour l'écriture d'un identificateur :

- ☑ il ne doit pas commencer par un chiffre
- ☑ il ne doit pas comporter d'espaces et de symboles spéciaux.
- ☑ il doit comporter au maximum 31 caractères significatifs
- ☑ il ne doit pas commencer par _
- ☑ Attention : pas de différence entre majuscule et minuscule

2.6 Les identificateurs prédéclarés

Il s'agit des fonctions et procédures intégrées au Dec $\mathcal{PASCAL}$. Celles-ci au contraire des mots reservés sont redéfinissables. Mais ce n'est pas conseillé. Dans ces identificateurs prédéclarés on retrouve, entre autres, les fonctions mathématiques de base :

- ☑ ABS
- ☑ SQRT()
- ☑ LN()
- ☑ ARCTAN()
- ☑ COS(), SIN()

Les fonctions de conversion de types :

- ☑ INT()
- ☑ CHR()
- ☑ ORD()

2.7 Les commentaires

Ils sont utilisés pour commenter les instructions d'un programme afin de les rendre plus compréhensibles à une personne qui n'a pas écrit le programme.

On utilise pour cela les caractères {} ou (**).

Attention il ne faut pas imbriquer les commentaires.

3 Structure Générale d'un Programme $\mathcal{PASCAL}$

3.1 L'entête

Un programme $\mathcal{PASCAL}$ commence toujours par la même ligne qui donne le nom du programme.

Syntaxe :

```
program toto(input, output);
```

La présentation peut être libre et on peut écrire :

Syntaxe :

```
program
    toto(input, output);
```

Mais pour facilité la relecture du programme il est préférable de laisser sur une même ligne cette entête.

3.2 La section Déclaration

Elle contient la définition des constantes, des labels, des types, des variables, des fonctions et des procédures définis par l'utilisateur.

Cette section apparaît après l'entête du programme et avant la section exécutable.

3.2.1 La section CONST

Elle sert à définir des constantes en leur associant des identificateurs :

Syntaxe :

```
CONST
    nom_constante = expression;
```

Un seul identificateur est associé à une expression.

Exemple :

```
CONST
    YEAR = 1984;
```

Dans le programme lorsqu'on utilise la constante "YEAR" elle est automatiquement remplacée par sa valeur "1984".

3.2.2 La section TYPE

Elle introduit les noms et les types de variables pour un type de données défini par l'utilisateur.

Syntaxe :

```
TYPE
    [identificateur de type] = [type pascal] VALUE [Valeur initiale du type]
```

Exemple :

```
TYPE
    jour_de_semaine = (Dim, Lun, Mar, Mer, Jeu, Ven, Sam);
VAR
    sem1, sem2 : jour_de_semaine;
```

On doit toujours définir tous les types utiles avant de les utiliser pour la définition d'autres types.(Sauf pour les pointeurs)

3.2.3 La section VALUE

On peut choisir d'utiliser **VALUE** comme une section.

Syntaxe :

```
VALUE
    identificateur := expression constante;
```

Attention **VALUE** ne peut pas être utilisée dans les fonctions et les procédures.

Exemple :

```
TYPE
    jour_de_semaine = (Dim, Lun, Mar, Mer, Jeu, Ven, Sam) VALUE Lun;
VAR
    sem1, sem2 : jour_de_semaine;
```

3.2.4 La section VAR

Elle déclare les variables et associe à chaque identificateur un type de données et une valeur initiale (en option).

Syntaxe :

```
VAR
    identificateur : type_de_variable VALUE valeur_initiale;
```

Exemple :

```
VAR
    reponse, test    : boolean;
    temp1, x1, x2    : integer;
    re1              : real VALUE 3.14;
```

Règles d'utilisation de la valeur initiale :

- ☑ elle doit être du même type ou compatible avec la variable
- ☑ pas d'initialisation de variable de type **fichier**
- ☑ la constante **NIL** est le seul moyen d'initialiser un pointeur.

3.3 Le bloc d'instructions

Un programme $\mathcal{PASCAL}$ comporte toujours un bloc d'instructions compris entre les marqueurs `"begin"` et `"end."` et il est très important de remarquer qu'il s'agit d'un point «.» et non d'un point-virgule «;» comme à la fin de l'entête.

Le squelette d'un programme $\mathcal{PASCAL}$ est donc le suivant :

Syntaxe :

```
program mon_pg (input, output);
    declaration (section CONST VAR ...);
begin
    ... Instructions pascal;
end.
```

4 Types de Donnée et Valeurs

Toute donnée, en $\mathcal{PASCAL}$, doit avoir un type. Le type détermine l'intervalle de valeurs, les opérateurs valides et la taille de l'allocation mémoire pour cette variable.

4.1 Type Ordinal

Il est soit prédéfini en $\mathcal{PASCAL}$, soit redéfini par l'utilisateur. Ce type demande un identificateur et des bornes de valeurs.

Dans un type ordinal, les valeurs sont ordonnées de telle sorte que chaque donnée possède une valeur ordinale unique.

Exemple :

```
TYPE
    jour_de_semaine = (Dim, Lun, Mar, Mer, Jeu, Ven, Sam);
VAR
    sem1, sem2 : jour_de_semaine;
```

4.2 Type Entier

4.2.1 les types Integer → Integer64

Dec $\mathcal{PASCAL}$ possède en standard différents types d'entier (Cf. tableau 2).

Type	Codage	Intervalle
Integer	8 bits	$-128 \rightarrow 128$
Integer16	16 bits	$-32768 \rightarrow 32767$
Integer32	32 bits	$-2147483648 \rightarrow 2147483647$
Integer64	64 bits	$-9223372036854775808 \rightarrow 9223372036854775807$

TAB. 2: Tableau des valeurs de types entier du Dec $\mathcal{PASCAL}$

La plus grande valeur de **Integer32** et **Integer64** est donnée respectivement par la valeur prédéclarée **MAXINT** et **MAXINT64**.

4.2.2 Les types Unsigned → Unsigned64

Dec $\mathcal{PASCAL}$ possède en standard différents types d'entier non signé (Cf. tableau 3).

Type	Codage	Intervalle
Unsigned8	8 bits	$0 \rightarrow 255$
Unsigned16 Cardinal16	16 bits	$0 \rightarrow 65535$
Unsigned32 Cardinal32	32 bits	$0 \rightarrow 4294967295$
Unsigned64	64 bits	$0 \rightarrow 18446744073709551615$

TAB. 3: Tableau des valeurs de types entier non signé du Dec $\mathcal{PASCAL}$

La plus grande valeur de **Unsigned** et **Unsigned64** est donnée respectivement par la valeur prédéclarée **MAXUNSIGNED** et **MAXUNSIGNED64**.

Pour pouvoir forcer un entier signé à sa valeur non signée on utilise la fonction **UINT**

4.2.3 Les Changements de bases

Le Dec $\mathcal{PASCAL}$ reconnaît en standard les bases de 2 à 36.

Pour écrire ou faire des calculs dans une autre base, il faut formater le nombre.

Syntaxe :

```
[+/-] base#'nombre ecrit dans sa base'
```

Exemple :

```
2#10000011
```

ou

Exemple :

```
2#'1000 0011'
32#1J
-16#'7FFF FFFF'
```

Le Dec $\mathcal{PASCAL}$ connaît aussi les bases les plus courantes (2,8,16). On utilise les lettres **b**→ binaire, **o**→octal et **x**→hexadécimal. On peut alors formater les nombres différement.

Exemple :

```
%b'1000 0011'
%o'7712'
%x'DEC'
```

4.2.4 Le type Char

Ce type permet de stocker un seul caractère de la table ASCII (American Standard Code for Information Interchange). La valeur maximale de **CHAR** est donnée par **MAXCHAR**.

Pour traiter un caractère il doit être entre ' : **'X'**.
Pour traiter le caractère ' : **''**.

On peut aussi utiliser en Dec $\mathcal{PASCAL}$ les doubles cotes ". **"X"** et **" " "** pour le caractère ".

La fonction **ORD()** retourne la valeur ordinale entière du caractère passé en paramètre. Il est possible de spécifier un caractère non imprimable de la table ASCII.

La fonction **CHR()** retourne le caractère de la valeur ordinale entière passée en paramètre.

Exemple :

```
VAR
    carac   :   char;
    entier  :   integer ;
    ...
    carac   :=  CHR(65);
    entier  :=  ORD('A');
```

4.2.5 Le type Boolean

La plupart du temps il s'agit du résultat d'un test de validité. Il peut prendre seulement deux valeurs : **TRUE** et **FALSE** deux identificateurs prédéclarés. On a **ORD(TRUE)=1** et **ORD(FALSE)=0**.

4.2.6 Les types énumérés

Il s'agit d'une liste ordonnée de constantes définies par l'utilisateur et spécifiée par un identificateur. Dec $\mathcal{PASCAL}$ peut accepter jusqu'à 65535 types énumérés.

La valeur d'un type énuméré commence avec la valeur 0 et augmente de gauche à droite.

Exemple :

```
TYPE
        saisons = (Printemps, Ete, Automne, Hiver) ;
                   0          1    2        3
VAR
        saison : saisons VALUE Hiver;
```

La fonction **ORD()** accepte les types énumérés. Il faut que les identificateurs d'énumération soient uniques.

4.2.7 Les types sous-intervalles

Ces types sont définis par l'utilisateur et spécifient les limites d'autres types (type de base $\mathcal{PASCAL}$ ou type définit par l'utilisateur).

Exemple :

```
TYPE
    jour        =   (Dim, Lun, Mar, Mer, Jeu, Ven, Sam);
    week_end    =   Sam..Dim;
    jour_sem    =   Lun..Ven;
```

4.3 Le type réel

Dec $\mathcal{PASCAL}$ définit les types **REAL**, **SINGLE**, **DOUBLE** et **QUADRUPLE**.

Type	Précision
Single	7 décimales
Double	15 décimales
Quadruple	33 décimales

TAB. 4: Tableau formats flottants supportés par le Dec $\mathcal{PASCAL}$

Le Type **REAL** est noté en forme décimale ou exponentielle. Par contre les types **DOUBLE** et **QUADRUPLE** sont notés en forme exponentielle.

4.4 Le type pointeur

Ce type fait référence à une variable dynamique. On peut ainsi la créer, la détruire, la modifier pendant le déroulement du programme.

On utilise deux procédures afin de gérer ces pointeurs :

- **NEW()** qui permet d'allouer la mémoire pour la nouvelle variable créée
- **DISPOSE()** qui permet de désallouer la mémoire

Exemple :

```
TYPE
     mon_enreg   =  RECORD
                       Nom :    STRING(30);
                       age :    INTEGER;
                    End
VAR
     ptr:^mon_enreg;
     ...
    NEW(ptr);
     ptr ^ := mon_enreg[nom : 'toto'; age : 10];
```

La valeur **NIL** indique que le pointeur ne fait référence à aucune variable.

Exemple :

```
VAR
     ptr:^ integer  VALUE NIL;
```

4.5 Les types structurés

Ces types sont définis par l'utilisateur et contiennent tous des types de données $\mathcal{PASCAL}$ de base.

4.5.1 Le type tableau

Un tableau est un groupe d'éléments qui ont le même type de données. Un élément est identifié par un index ordinal qui désigne la position de l'élément.

Exemple :

```
VAR
     compteur : array [1..10] of integer ;
```

Pour les tableaux multidimensionnels :

Exemple :

```
VAR
     morpion3d : array [1..3] of array [1..3] of char;
```

Dans le programme :

Exemple :

```
morpion3d[1,1]:='X';
```

ou

Exemple :

```
morpion3d[1][1]:='X';
```

Il est possible d'initialiser un tableau lors de sa définition.

Exemple :

```
VAR
    compteur : array [1..10] of integer ;
VAR
    nombre  : compteur VALUE [1..3,5:1;4,6:2;7..9:3;10:6];
    nombre2 : compteur VALUE [1..3,5:1;OTHERWISE 1];
```

4.5.2 Le type Record

C'est un groupe de variables, appelées champs, qui contient plusieurs types de données.

Exemple :

```
TYPE
    joueur  = RECORD
                gain       :   integer ;
                perte      :   integer ;
                pourcentage :  real ;
                end;
VAR
    joueur1, joueur2 : joueur;
```

dans le programme :

Exemple :

```
joueur1.gain := 18;
```

Enregistrement à champs variants : un champs contient alors plusieurs types de données.

Exemple :

```
TYPE
    enreg (a : integer) = RECORD
                            champ1 : real;
                            case a of
                            0 : (X : integer);
                            1 : (Y : real);
                            end;
```

4.5.3 Le type Set

Il s'agit d'une collection de données de type ordinal.

Exemple :

```
TYPE
    sousset : set of 0..255;
```

4.5.4 Le type Fichier

Il s'agit d'une séquence de variables de même type. Le nombre de composants n'est pas fixé.

Exemple :

```
VAR
    fic_int : FILE of integer;
TYPE
    fic_rec : FILE OF RECORD
                trial : integer;
                temp,pression : integer;
                purete : real;
                end;
```

4.5.5 Le type fichier Text

Il s'agit d'un sous type de fichier qui ne contient lui que des caractères.

Exemple :

```
fic_int : TEXT
```

4.5.6 Le type chaîne de caractère String

Il existe différents types de chaînes de caractères en Dec $\mathcal{PASCAL}$:

- ☑ PACKED ARRAY OF CHAR : spécifie un chaîne de caractères de longueur définie
- ☑ VARYING OF CHAR : spécifie une chaîne de caractères de longueur variable avec une longueur maximale
- ☑ STRING : voie standard pour la spécification de chaîne de caractères

Packed array of char

Exemple :

```
VAR
        machaine : packed array [1..20] of char ;
                                < 65535
```

Varying of char

Exemple :

```
VAR
        machaine : varying[10] of char value '' ;
```

String

Exemple :

```
VAR
        machaine : string(10) value '' ;
```

5 Expressions et Opérateurs

5.1 Les expressions

En $\mathcal{PASCAL}$ une expression consiste en un ou plusieurs opérandes dont le résultat est une seule valeur.

Si une expression contient plusieurs opérandes alors ils sont séparés par des opérateurs.

Les Opérandes : nombre, chaîne, constante, variable, fonction...

Les Opérateurs : opérateur arithmétique, relationnel, logique, opérateur de type...

Le $\mathcal{PASCAL}$ reconnaît deux types d'expressions :

- ☑ une expression constante est définie à la suite de la compilation
- ☑ une expression de déroulement renvoie le résultat quand vous exécuter le programme

Lorsque l'on écrit un programme les deux opérandes d'une expression doivent être de même type.

5.2 Les Opérateurs

Il y a plusieurs classes d'opérateurs et l'on peut créer des expressions très complexes en combinant opérateurs et identifiants.

5.2.1 Les opérateurs arithmétiques

Ils sont utilisés pour les formules mathématiques (Cf.tableau 5).

+	Somme	DIV	Division entière
−	Différence	REM	Reste de la division entière
*	Produit	MOD	Fonction Modulo
**	Puissance	/	Division

TAB. 5: Tableau des opérateurs arithmétiques du Dec $\mathcal{PASCAL}$

5.2.2 Les opérateurs relationnels

Un tel opérateur teste la relation entre deux ordinaux, réels, doubles et renvoie un résultat booléen (Cf. Tableau 6).

On peut utiliser cet opérateur sur les chaînes.

=	Égalité	$\leq$	Inférieur ou égal
<>	Différent	>	Supérieur
<	Inférieur	$\geq$	Supérieur ou égal

TAB. 6: Tableau des opérateurs relationnels du Dec $\mathcal{PASCAL}$

5.2.3 Les opérateurs logiques

Un opérateur logique évalue une ou plusieurs expressions booléennes et retourne un booléen (Cf. tableau 7).

AND	Et	OR	Ou
NOT	Non		

TAB. 7: Tableau des opérateurs logiques du Dec $\mathcal{PASCAL}$

5.2.4 Les opérateurs de chaînes de caractères

Un opérateur de chaîne de caractères concatène, compare des expressions chaînes. Le résultat est une chaîne ou un booléen (Cf. tableau 8).

+	Concatène 2 chaînes et renvoie le résultat dans une chaîne
=	Test d'égalité de 2 chaînes renvoie un booléen
<>	Test de différence de 2 chaînes renvoie un booléen
<	Test d'égalité de 2 chaînes renvoie un booléen
<	$A < B$ Test si la chaîne en valeur ASCII A est < B
<	$A < B$ Test si la chaîne en valeur ASCII A est < B
$\leq$	$A \leq B$ Test si la chaîne en valeur ASCII A est $\leq$ B
>	$A > B$ Test si la chaîne en valeur ASCII A est < B
$\geq$	$A \geq B$ Test si la chaîne en valeur ASCII A est $\geq$ B

TAB. 8: Tableau des opérateurs de chaînes du Dec $\mathcal{PASCAL}$

Les fonctions prédéfinies **EQ**, **NE**,**GE**, **LE**, **GT** et **LT** ont le même effet que les opérateurs du tableau 8 sur les comparaisons de chaînes mais il faut impérativement que les chaînes aient la même taille.

La fonction qui donne la taille d'un chaîne est **LENGTH()**.

Exemple :

Soit la chaîne de caractères suivante : chaine :='toto est en vacances' ;

```
VAR
    longueur : integer ;
```

Dans le programme la longueur de la chaîne est donnée par :

```
longueur := LENGTH( chaine ) ;
```

5.2.5 Précédence des Opérateurs

Il existe un ordre dans lequel Dec $\mathcal{PASCAL}$ combine les opérandes. Cf. tableau 9.

Haute Priorité ↓ Basse Priorité	NOT ** $*, /, DIV, REM, MOD, AND$ $+, -, OR$ $=, <>, >, \geq, <, \leq, In$

TAB. 9: Priorité des opérateurs du Dec $\mathcal{PASCAL}$

6 Les séparateurs et opérateurs instructions

Ils sont présents pour spécifier les actions, et apparaissent dans la partie exécutable du programme.

6.1 L'opérateur d'affectation

C'est l'opérateur " **:=**" qui affecte un valeur à une variable.

Syntaxe :

```
variable := valeur;
```

6.2 L'opérateur Break

L'opérateur **break** fait sortir des structures de type « for, while, repeat...» qui contiennent cet opérateur. Il s'utilise seul. C'est un opérateur dont l'utilisation n'est pas conseillée.

Exemple :

```
repeat
    i := i + 1;
    if i=20 then      begin
                       writeln('arrive');
                      break;
                      end
    else  writeln('pas arrive');
until (i = -1);
```

6.3 L'opérateur Case

L'opérateur **case** sélectionne une des instructions à effectuer. L'exécution dépend de la valeur ordinale appelé dans le sélecteur **case**.

Syntaxe :

```
case éslecteur of
    liste_de_label... : instructions;
    otherwise instructions;
end;
```

Exemple :

```
case age of
    1..4     :   ecole := 'maternelle';
    5..10    :   ecole := 'elementaire';
    11..15   :   ecole := 'college';
    16..18   :   begin
                 ecole := 'lycee';
                 end;
    19       :   ecole := 'bac';
    otherwise    ecole := 'diplome';
end;
```

6.4 L'instruction block

Un **block** regroupe une série d'instructions.

Exemple :

```
begin
    instructions ;
    instructions ;
end;
```

6.5 L'instruction vide

Avec l'instruction vide il n'y a pas de code à exécuter.

Syntaxe :

```
case alpha of
    'a', 'e', 'i'    : drap_alpha := 'voyelle';
    'u'              :; { Instruction vide }
end;
```

6.6 L'instruction For

C'est une instruction de boucle en regard d'une valeur de contrôle. Cette variable de contrôle évolue dans un intervalle.

Syntaxe :

for *variable_de_contrôle* := *valeur* { to / downto } *valeur_final* do

for *variable_de_contrôle* in *expression_set* do

La valeur de contrôle est de type ordinal.

Exemple :

```
for i := 1 to 20 do
    begin
        writeln ('i =', i);
    end;
```

6.7 L'instruction Repeat

C'est une instruction de boucle qui exécute les instructions qu'elle contient jusqu'à ce qu'une expression booléenne soit vérifiée.

Syntaxe :

```
repeat
    instructions ;
until (expression booleenne);
```

Ici nous n'avons pas besoin d'utiliser les instructions de bloc "**begin end**" car les instructions sont déjà entourées par le bloc "**repeat until**".

Exemple :

```
repeat
     i := i + 1;
     if i=20 then
          begin
          writeln ('arrive');
          i:=-1;
          end
     else  writeln ('pas arrive');
until (i = -1);
```

6.8 L'instruction While

C'est une instruction de boucle qui exécute les instructions tant qu'une expression booléenne est vérifiée.

Syntaxe :

```
while (expression booleenne) do
     begin
          instructions;
     end;
```

Exemple :

```
while (frequence < 1000.00) do
     begin
          xl := pi * 2 * frequence * inductance;
          writeln ('xl a ',frequence,:4:0, 'Hz=',xl:8:2);
          frequence :=  frequence + 100.00;
     end;
```

6.9 L'instruction If

Cette instruction teste une expression booléenne et fait une action spécifique si le résultat est "**TRUE**".

La clause "**else**" est faite seulement si le résultat est faux.

Syntaxe :

```
if (expression_booleenne) then
     expressions
else
     expressions;
```

Si l'on a une clause "**else**", les instructions après "**then**" ne doivent pas avoir de **;**.

Exemple :

```
if (x>10) then y := 4
else y := 5;
if (x>10) then
            begin
            y := 4;
            z:= 4;
            end
else y := 5;
```

Il faut faire très attention aux structures "**if ... then ... else**" imbriquées.

6.10 L'instruction Return

Cette instruction redonne le contrôle au programme qui a appelé la procédure ou la fonction en cours d'exécution.

Syntaxe :

```
return valeur;
```

Une fonction renvoie une valeur au programme appelant.

7 Fonctions et Procédures

Ce sont des sous programmes qui se différencient de la manière suivante :

- ☑ Une procédure contient une liste d'instructions à exécuter
- ☑ Une fonction contient une liste d'instructions à exécuter et renvoie une valeur unique

Elles sont souvent aussi appelées routines. Il est possible aussi d'accéder à des routines externes au Dec $\mathcal{PASCAL}$.

7.1 La Déclaration

Il faut impérativement déclarer les fonctions ou procédures avant de les utiliser.

Syntaxe :

```
[attributs] procedure nom_procedure [(liste de parametres)];é
dclarations des variables utiles};
    begin
        instructions de la procedure;
    end;
```

Syntaxe :

```
[attributs] function nom_fonction [(liste de parametres)] : [attribut] type;
    declarations des variables utiles;
begin
    instructions de la fonction;
    return nom_fonction;
end;
```

7.2 L'appel des fonctions et procédures

L'appel d'une fonction ou procédure dans le programme principal s'opère de la manière suivante :

Syntaxe :

```
nom_procedure [( liste de parametres)];
```

Syntaxe :

```
variable := nom_fonction [( liste de parametres)];
```

7.3 Les paramètres

Il est possible de passer des paramètres aux fonctions de différentes façons. La première consiste en une copie des valeurs passées à la fonction. La seconde permet de modifier les variables passées en paramètres.

7.3.1 Passage par Valeurs

Lors du passage par valeurs, les variables paramètres de la fonction ou procédure reçoivent des copies de valeurs des variables du programme principal. Ainsi les variables du programme principal ne sont pas modifiées par la fonction ou procédure.

Exemple :

```
 function  addition(op1,op2,op3 :  real ) :  real;
 procedure debut;
VAR
     a,b,c,z :  real;
 begin
     a := 2.5; b := 5.2;
     c := a + b;
     z := a * b;
     z := addition(a,b,c);
end;
 function  addition(op1,op2,op3 :  real ) :  real;
 begin
     addition := op1 + op2 + op3;
     op1 := op2 + op3;
end;
```

Ici la valeur de "**a**" dans le programme principal, donc "**op1**" dans la fonction, ne sera pas modifiée car la passage se fait par valeur.

7.3.2 Passage par paramètres

Lors du passage par paramètres, les variables paramètres de la fonction ou procédure reçoivent les adresses mémoires des variables du programme principal. Ainsi les variables du programme principal sont modifiées par la fonction ou procédure.

Exemple :

```
function addition(op1,op2,op3 : real) : real;
procedure debut;
VAR
    a,b,c,z : real;
begin
    a := 2.5; b := 5.2;
    c := a + b;
    z := a * b;
    z := addition(a,b,c);
end;
function addition(VAR op1,op2,op3 : real) : real;
begin
    addition := op1 + op2 + op3;
    op1 := op2 + op3;
end;
```

Ici la valeur de "**a**" dans le programme principal, donc "**op1**" dans la fonction, sera modifiée car la passage se fait par paramètres.

7.3.3 Association de paramètres

Il est possible de passer les paramètres en même temps par valeurs et par paramètres. Afin de passer les paramètres réels dans la fonction ou procédure, on les place dans l'ordre des paramètres formels.

Exemple :

```
function addition(VAR op1,op2 : real ; op3 : real) : real;
begin
    addition := op1 + op2 + op3;
end;
```

Dans le programme :

somme := addition($\underbrace{a}_{op1}$,$\underbrace{b}_{op2}$,$\underbrace{15}_{op3}$);

8 Les fichiers

Un fichier est une collection de données individuelles que l'on appelle composants.

On peut utiliser les fichiers en déclarant une variable de type **FILE**. Dec $\mathcal{PASCAL}$ crée automatiquement un fichier "buffer" du type des composants. On ne peut pas faire d'opération directement sur le fichier, il faut passer par le fichier "buffer".

Syntaxe :

```
VAR
    nom_variable_fichier : FILE OF type_de_donnees};
```

Le type de données peut être entier, réel ou encore des structures de données par l'intermédiaire des enregistrements (type "record").

Pour les fichiers texte la syntaxe est un peu différente.

Syntaxe :

```
VAR
    nom_variable_fichier : TEXT
```

Afin de gérer ces fichiers il existe différentes routines (fonctions) développées de façon standard en Dec $\mathcal{PASCAL}$.

- **CLOSE** qui permet de fermer un fichier
- **DELETE** qui permet d'effacer un fichier
- **EOF** qui permet de tester la fin d'un fichier
- **GET** acquiert le composant courant du fichier
- **EXTEND** qui permet d'ouvrir et d'écrire dans un fichier après le dernier composant
- **OPEN** qui permet d'ouvrir et d'écrire dans un fichier
- **PUT** ajoute le composant courant dans le fichier
- **READ** qui permet de lire un ou plusieurs composants du fichiers dans des variables
- **RESET/REWRITE** qui permet de mettre le pointeur de composant sur le premier composant du fichier

Pour les fichiers TEXT il existe quelques fonctions spécifiques

- **EOLN** qui permet de tester la fin d'une ligne
- **READLN** qui permet de lire une ligne dans un fichier
- **WRITELN** qui permet d'écrire dans un fichier

9 Format des sorties à l'écran

Par défaut lors de l'affichage des valeurs des variables à l'écran par la fonction **WRITELN()**, Dec $\mathcal{PASCAL}$ réserve une longueur d'affichage.
Le plus souvent les sorties écrans sont de la forme :

Le résultat est : 12

Cette longueur d'affichage est souvent plus longue que le nombre de caractères à afficher. On a alors recours à des options de formatage des sorties.

9.1 Format des entiers

Afin de réduire l'espace entre la chaîne de caractères et la variable à afficher, on spécifie le nombre minimum de caractères dans lesquels on va écrire la variable.

Exemple :

```
writeln ('Le resultat est :', x:2)
```

Avec x un entier. On obtient alors

Exemple :

```
Le resultat est : 5
```

ou encore

Exemple :

```
Le resultat est : 125
```

9.2 Format des réels

Afin de réduire l'espace entre la chaîne de caractères et la variable à afficher, on spécifie le nombre minimum de caractères dans lesquels on va écrire la variable.

Exemple :

```
writeln('Le resultat est :',x:5:2)
```

Avec x un réel. On obtient alors

Exemple :

```
Le resultat est : 5.25
```

ou encore

Exemple :

```
Le resultat est :125.25
```

9.3 Format en différentes bases

Il est aussi possible d'afficher les résultats des calculs dans des bases différentes de 10.
Par exemple pour écrire un résultat sous sa forme binaire codée sur 8 bits on aura dans le programme :

Exemple :

```
writeln('Le resultat est :',BIN(x):8)
```

Avec x un entier. On obtient alors

Exemple :

```
Le resultat est : 00101101
```

On peut aussi avoir la forme octale par **OCT**, la forme hexadécimale par **HEX**, la forme décimal par **DEC** et la forme décimale non signée par **UDEC**.

10 La Compilation

10.1 Pourquoi compiler un programme

Lorsque vous tapez votre programme $\mathcal{PASCAL}$ avec un éditeur de texte comme VI, Emacs ou encore Nedit, vous créez un fichier source. Les instructions en langage $\mathcal{PASCAL}$ qu'il contient ne sont pas

directement interprétables par l'ordinateur. Il faut donc transformer ce fichier texte (compréhensible par vous, normalement) en fichier exécutable qui lui sera compris par la machine. C'est ce que l'on appelle la phase de compilation.

Attention il est très important d'enregistrer votre fichier $\mathcal{PASCAL}$ avant de compiler. En effet le compilateur lit le fichier écrit sur le disque. Il ne prendra donc pas en compte les modifications de votre fichier faites en mémoire et non enregistrées.

10.2 Commandes de compilation

Il est important de respecter certaines règles pour que tout se passe bien :

1. Il faut impérativement nommer le fichier avec une extension "*.p" ou "*.pas" sinon le compilateur refusera de compiler le fichier
2. Vérifiez que le fichier à compiler se trouve bien dans le répertoire où vous vous trouvez (commande Unix "pwd" et "ls")
3. Ne pas nommer le fichier exécutable comme le fichier source. Votre fichier source se retrouverait écrasé.

Compiler sans nommer le fichier exécutable Dans la fenêtre Unix :

Exemple :

```
crabe > pc prog1.pas ↩
```

Si tout se passe bien vous avez :

Exemple :

```
crabe >
```

S'il y a des erreurs de compilation, reprenez votre programme source et corrigez les, enregistrez le puis recompilez le.

Lorsqu'il n'y a plus d'erreurs vous pouvez lancer votre programme par :

Exemple :

```
crabe > a.out ↩
```

Compiler en nommant le fichier exécutable Dans la fenêtre Unix :

Exemple :

```
crabe > pc -o prog prog1.pas ↩
```

Si tout se passe bien vous avez :

Exemple :

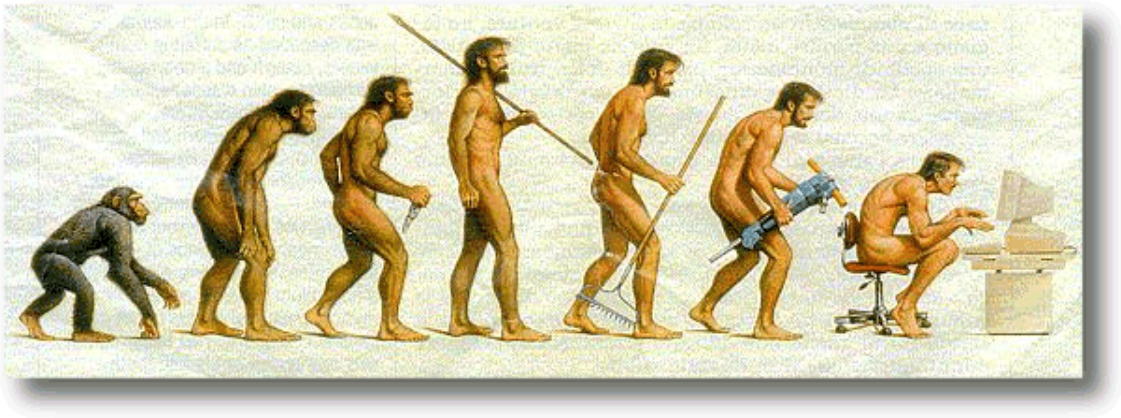
```
crabe >
```

S'il y a des erreurs de compilation, reprenez votre programme source et corrigez les, enregistrez le puis recompilez le.

Lorsqu'il n'y a plus d'erreurs vous pouvez lancer votre programme par :

Exemple :

```
crabe > prog ↩
```



Réflexion sur l'évolution humaine...