Le langage de programmation Pascal

1997-1998

Professeur : Bertrand LIAUDET

Sommaire

Introduction

Ce polycopié présente les éléments de base de la programmation avec le langage Pascal. La connaissance des principes de base de l'algorithmique est nécessaire pour une compréhension immédiate de ce polycopié. Cependant ce polycopié pourra aussi permettre de retrouver les principes généraux de l'algorithmique.

Le Pascal a été créé en 1970 par Niklaus Wirth, professeur à l'École Polytechnique de Zurich.

Son nom a été choisi en l'honneur de Blaise Pascal, savant, penseur et écrivain français, 1623-1662, concepteur d'une machine arithmétique.

Le langage Pascal a été conçu à des fins pédagogiques. Le monde industriel s'en est rapidement emparé du fait de ses qualités intrinsèques et de l'existence du logiciel Turbo Pascal.

Le noyau du langage défini en 1977, la norme ISO 7185, ou la norme ANSI du début des années 80 (IEEE Pascal Standard X3J9/81-093) ne suffisent plus à décrire le Pascal du fait des extensions importantes qu'il a subies, particulièrement dans le logiciel Turbo Pascal.

Le Pascal a été conçu à partir de trois langages de programmation des années 60 : le Fortran (For-mule Tran-slation, langage dédié au calcul scientifique), le Cobol (langage dédié à l'informatique de gestion) et l'Algol (langage algorithmique).

Le Pascal est un langage compilé (par opposition aux langages interprétés). Le compilateur est un programme qui crée, à partir du texte du programme Pascal, un programme écrit dans le langage de la machine. Les compilateurs possèdent tous des spécificités. Du coup, tous les programmes Pascal ne sont pas tous compréhensibles par tous les compilateurs Pascal. C'est le problème de la portabilité.

Nous utilisons le compilateur Turbo Pascal, version 7.

Bibliographie

Ce polycopié n'aurait pas pu être écrit sans le précieux secours du polycopié de M. Jean-Christophe RIAT : Programmation en Pascal, ESTP, 1994-1995.

On pourra consulter les ouvrages suivants :

Delannoy Claude, Programmer en Turbo Pascal 7, Eyrolles, 1993, 1997. 350p. 160 Francs. C'est l'ouvrage le plus complet actuellement sur le marché. Un peu cher peut être...

Spoljar Philippe, Pascal, Sybex, 1995. 350p (format poche). Environ 30 Francs d'occasion. C'est le meilleur rapport qualité prix.

Frala Bernard, L'indispensable pour Turbo Pascal versions 3 à 6, Marabout, 1993. 309p (format poche). Environ 30 francs d'occasion. Son organisation est plus confuse que celle du précédent.

A. Présentation générale du langage Pascal

Le langage Pascal est un langage de programmation.

La programmation c'est l'élaboration et la codification de la suite des opérations formant un programme, c'est-à-dire l'ensemble ordonné et formalisé des opérations suffisantes (et si possible nécessaires) pour obtenir un résultat.

1. Composantes formelles du langage

1.1. L'alphabet

Dans tous les langages formels, on peut toujours ramener les symboles primitifs à une liste finie de symboles qui composent en quelque sorte l'alphabet du langage.

Pour pouvoir présenter clairement l'alphabet du langage de programmation Pascal avec notre langage habituel, le français, il faut pouvoir distinguer les deux langages. En logique, on appelle le langage dont on parle (ici le langage de programmation) le langage objet, et le langage avec lequel on parle (ici le français) le métalangage (ou langage naturel). Pour distinguer les éléments du langage objet, le Pascal, des éléments du métalangage, le français, on présente généralement les éléments du langage objet entre guillemets.

L'alphabet du langage Pascal comporte les caractères suivants :

- **les lettres majuscules** : "A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N", "O", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z".
- **les lettres minuscules** : "a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l", "m", "n", "o", "p", "q", "r", "s", "t", "u", "v", "w", "x", "y", "z".
- **les chiffres** : "0", "1", "2", "3", "4", "5", "6", "7", "8", "9".
- **le souligné** : "_" (spécifique au Turbo Pascal).
- **les opérateurs arithmétiques** : "+", "-", "*", "/".
- **les opérateurs relationnels** : "<", ">", "=".
- **les séparateurs simples** : " ", ".", ",", ";", ":" (soit, sans les guillemets : espace, point, virgule, point-virgule et deux points).
- **les bornes** : "(", ")", "{", "}", "[", "]", "'" (le dernier élément c'est l'apostrophe).
- **les pointeurs** : "^"
- **les caractères spéciaux** : "$", "#", "@" et les caractères de contrôle (codes ASCII de 0 à 31, spécifiques au Turbo Pascal).

Les genres des caractères de l'alphabet ici présentés (lettres, chiffres, etc.) ne sont qu'indicatifs. "=" est un opérateur relationnel mais aussi un séparateur dans son utilisation pour définir les types. Et ":=" est le symbole de l'affectation.

1.2. Les mots

Avec cet alphabet, on peut construire les mots du langage. Ces mots, soit appartiennent à une liste prédéfinie, soit sont construits selon des **règles de formation** qui permettent de savoir si un mot appartient au langage ou pas.

Les genres de mots du langage Pascal sont les suivants :

- les mots réservés, ou mots-clés
- les identificateurs
- les nombres
- les chaînes de caractères
- les commentaires

Dans les mots-clés et les identificateurs, le Pascal ne distingue pas les lettres minuscules des lettres majuscules : "Begin" c'est la même chose que "BEGIN", "Var MaVariable : Integer;" c'est la même chose que "VAR MAVARIABLE : INTEGER;".

Dans les chaînes de caractères, le Pascal distingue entre les minuscules et les majuscules : 'Bonjour' c'est différent de 'BONJOUR'.

Le Pascal ne s'intéresse pas au contenu des commentaires.

1.2.1. Les mots-clés

Il s'agit des mots prédéfinis par le langage Pascal et qui ont une signification particulière. C'est une liste finie.

Un mot-clé ne peut pas être utilisé comme identificateur.

On peut regrouper les mots-clés en sous-genre (qui ne sont qu'indicatifs) :

Usage général

"program", "procedure", "function", "uses"

"begin", "end"

Déclaration

"const", "var", "type", "label"

"array", "record", "set", "file", "packed", "of"

"string", "integer", "char", "real", "boolean"

Tests

"if", "then", "else"

"case", "of"

Boucles

"for", "to", "downto"

"while", "do"

"repeat", "until"

Débranchements

"goto", "break", "exit"

Opérateurs

"and", "or", "div", "mod", "in", "not", "xor"

Bibliothèque

"forward", "implementation", "interface", "unit".

Divers

"constructor", "fail", "nil", "object", "private", "shl", "shr", "virtual", "with".

1.2.2. Les identificateurs

Un identificateur est un mot défini par l'utilisateur.

Les **règles de formation** de ces identificateurs sont les suivantes :

- Un identificateur est composé de lettres, majuscules ou minuscules, de chiffres et de soulignés.
- Il ne peut pas commencer par un chiffre.
- Seuls les 63 premiers caractères d'un identificateur sont significatifs.
- Il n'y a pas de différence entre les minuscules et les majuscules.

Un identificateur étant composé à partir de 26 lettres, 10 chiffres et 1 souligné, soit 37 caractères, seuls les 63 premiers caractères étant significatifs, et le premier caractère ne pouvant pas être un chiffre, il y a : 27*(37 puissance 62) identificateurs possibles, soit environ 4.56E+98.

1.2.3. Les nombres

Ils peuvent avoir le format suivant :

- notation des entiers : "10", "-945"

- notation des réels : "3.1416", "-0.345"
- notation scientifique : "0.323E+1", "123.45E-6"

Le nombre de chiffres significatifs est déterminé par le type du nombre : integer, real, etc.

1.2.4. Les chaînes de caractères

Une chaîne de caractères est une suite de symboles de l'alphabet (un texte) figurant entre deux apostrophes.

1.2.5. Les commentaires

Un commentaire est une suite de symboles de l'alphabet (un texte) figurant entre deux accolades : "{" et "}" ou encore entre deux parenthèses avec étoiles : "(*" et "*)".

1.3. Les séparateurs

Les mots du langage doivent être séparés entre eux par des séparateurs. Les séparateurs sont formés avec les séparateurs simples, les opérateurs arithmétiques, les opérateurs relationnels et les bornes : " ", ".", ",", ";", ":","+", "-", "*", "/", "<", ">", "=", "(", ")", "{", "}", "[", "]", "'".

2. Règles de formation

Les mots du langage Pascal vont être agencés selon des **règles de formation** de façon à former un programme.

Les règles de formation d'un programme en langage Pascal concernent les éléments suivants :

- Le programme principal
- Les variables
- Les types
- Les constantes
- Les instructions
- Les procédures
- Les fonctions
- Les bibliothèques (ou unités)
- Les labels

2.1. Le Programme

Un programme est constitué de :

- un en-tête (ou en-tête du programme principal)
- des sous-programmes
- un corps (ou corps du programme principal)

2.1.1. L'en-tête du programme principal

L'en-tête est défini par :

- un identificateur de programme

compris entre le mot-clé "Program" et le séparateur ";".

puis, accessoirement :

- des identificateurs de bibliothèques, regroupés derrière le mot-clé "Uses";
- des identificateurs de labels, regroupés derrière le mot-clé "Label";
- des identificateurs de constantes, regroupés derrière le mot-clé "Const";
- des identificateurs de types, regroupés derrière le mot-clé "Type";
- des identificateurs de variables, regroupés derrière le mot-clé "Var";

2.1.2. Les sous-programmes : procédures et fonctions

Entre l'en-tête et le corps, le programme peut contenir des sous-programmes qui peuvent être des procédures ou des fonctions.

- chaque procédure débute par le mot-clé "Procedure" suivi de l'identificateur de la procédure.
- chaque fonction débute par le mot-clé "Function" suivi de l'identificateur de la fonction.

Cf. § A.2.5 pour plus de précisions.

2.1.3. Le corps du programme principal

Le corps du programme est constitué de :

- une suite d'instructions

comprises entre les mots-clés "Begin" et "end". Le programme se termine par le séparateur point : ".".

Exemples Pascal

Présentation de la structure type d'un programme Pascal :

```
Program NomDuProgramme;
Uses
     (* zone pour déclarer les appels aux bibliothèques *)
Label
     (* zone pour déclarer les labels : à ne pas utiliser !!! *)
Const
     (* zone pour déclarer les constantes *)
Type
     (* zone pour déclarer les types *)
Var
     (* zone pour déclarer les variables globales *)

(* Procédures et fonctions *)
     (* zone pour déclarer les procédures et les fonctions *)

(* Programme principal *)
Begin
     (* zone des instructions du programme principal *)
end.
```

Le programme le plus simple aura un en-tête réduit à son nom et un corps avec des instructions sans variable :

```
Program LePlusSimple;
begin
     writeln ('bonjour');
end.
```

L'exemple ci-dessous est celui d'un petit programme appelant une procédure qui calcule le périmètre et la surface d'un cercle à partir de son diamètre.

```
Program CalculsDeCercle;
Uses
     crt;
Const
     pi  = 3.1416;
Var
     diam, surf, peri : real;
     q : char;

(* Procédures et fonctions *)
```

```
    Procedure PeriSurfCerc (Diametre : real; var Perimetre, Surface : real)
    Begin
        Perimetre := Diametre * PI;
        Surface := PI * sqr(Diametre) / 4;
    end;

    (* Programme principal *)
    Begin
        clrscr;                                    (*effacement de l'écran *)
        repeat                          (*boucle de repetition du programme *)
            write ('entrez le diametre : ');
            readln (diam);
            PeriSurfCerc (diam, peri, surf);
            writeln ('perimetre : ':20, peri:5:2);
            writeln ('surface :  ':20, surf:5:2);
            write ('entrer q pour quitter le programme');
        readln (q);
        until q = 'q';
    end.
```

Dans les cas simples, le programme principal, c'est la partie du programme qui communique avec l'extérieur, le plus souvent avec l'utilisateur, parfois avec d'autres programmes, parfois avec des fichiers. Le programme principal récupère les données à traiter, les traite en appelant les procédures, puis affiche les résultats.

2.2. Les Variables

Une variable est un dispositif capable de recevoir, de conserver et de fournir une information, c'est-à-dire une valeur d'un certain type, autant de fois souhaitées.

Les variables ont :

- un nom : c'est l'identificateur de la variable.
- un type : il fixe les valeurs que peut prendre la variable.
- une valeur : c'est la traduction en fonction du type de l'état physique de la variable.
- une adresse : c'est la localisation de la variable dans la mémoire.

Remarque :

Une variable a toujours une valeur.

En effet, physiquement, une variable est une suite d'octets repérée par l'adresse du premier octet. A cette adresse, on associe un nom. Le type permet de savoir comment on va lire ces octets. (Si c'est un type "char", la traduction décimale de la valeur binaire de l'octet correspond au numéro du code ASCII du caractère et détermine ainsi la valeur du caractère. Si c'est un type "byte", c'est-à-dire un entier sur un octet, la traduction décimale de la valeur binaire de l'octet détermine la valeur de l'entier.) Quand on crée une variable, les bits des octets de la variable sont dans un état qui correspond à une valeur et qui est donc celle de la variable.

2.3. Les Types

Un type définit pour une variable :

- son ensemble de définition (toujours fini)
- les opérations et les fonctions qui peuvent lui être appliquées.

Le tableau ci-dessous classe tous les types du Pascal :

Types		
Simples		**Complexes**
Standards	**Définis par l'utilisateur**	
Entiers	Intervalles	Tableaux
Réels	Énumérés	Chaînes de caractères
Caractères		Ensembles
Booléens		Enregistrements
Pointeurs		Fichiers

Exemples Pascal

Voici quelques exemples de déclarations de types et de variables:

```
Type
    Table = array [1..10] of integer;
    Chaine10 = string [10];
    Notes = 0..20;
    Jours = (lun, mar, mer, jeu, ven, sam, dim);
    Agenda = array [7] of Jours;
Var
    I, J : integer;
    X, Y, Z : real;
    ch1 : string;
    ch2 : Chaine10;
    Tab : Table;
    Tab2 : array [1..20] of real;
    UnJour = Jours;
    UnAgenda = Agenda;
```

2.4. Les Constantes

Une constante c'est une variable constante, autrement dit un dispositif capable de ne recevoir qu'une seule fois une information, puis de la conserver et de la fournir autant de fois souhaitées.

Les constantes ont un nom et un type déterminé automatiquement selon la valeur donnée à la constante.

2.5. Les Procédures

Une procédure est un sous-programme qui comporte des variables en entrée et en sortie.

Une procédure a :

- un en-tête
- un corps

Une procédure est utilisée comme une affectation : c'est une instruction complète.

2.1.1. L'en-tête de la procédure

L'en-tête est défini par :

- un nom : c'est un identificateur

suivi de, entre parenthèses : "(" et ")"

- la liste des paramètres de définition de la procédure

le tout compris entre le mot-clé "Procedure" et le séparateur ";".

La liste des paramètres de définition est composée d'une liste de noms de variables dont on précise le type et le mode de passage : en entrée (par valeur) ou en sortie (par adresse).

Cf. § C. 3 pour plus de précisions.

Exemple Pascal

```
Procedure Equa1 (A, B: real; var X : real; var NbSol : integer);
```

Les paramètres de définition de la procédure sont séparés par des ";". Le mot-clé "var" signifie que la variable est en sortie de la procédure. Cet en-tête de procédure nous dit que "A" et "B" sont des réels en entrée, que "X" est un réel en sortie et que "NbSol" est un entier en sortie.

2.1.2. Le corps de la procédure

Le corps de la procédure est constitué de :

- éventuellement des identificateurs de variables dites locales

Regroupés derrière le mot-clé "Var";

- une suite d'instructions

comprises entre les mots-clés "Begin" et "end". Le procédure se termine par le séparateur point virgule : ";".

2.6. Les Fonctions

Une fonction est un sous-programme qui comporte des variables en entrée et dont la seule variable en sortie est le nom même de la fonction.

Une fonction a :

- un en-tête
- un corps

Une fonction est utilisée comme une variable, excepté qu'elle ne peut pas être affectée.

2.1.1. L'en-tête de la fonction

L'en-tête est défini par :

- un nom : c'est un identificateur

suivi de, entre parenthèses : "(" et ")"

- la liste des paramètres de définition de la fonction

le tout compris entre le mot-clé "Function" et le séparateur ";".

La liste des paramètres de définition est composée d'une liste de noms de variables dont on précise le type.

Exemple Pascal

```
Function Puissance (X: real; N : integer);
```

Les paramètres de définition de la fonction sont séparés par des ";". Cet en-tête de fonction nous dit qu'on calcule la "Puissance" à partir de "A" qui est un réel et de "N" qui est un entier.

2.1.2. Le corps de la fonction

Le corps de la fonction est constitué de :

- éventuellement des identificateurs de variables dites locales

regroupées derrière le mot-clé "Var";

- une suite d'instructions

comprises entre les mots-clés "Begin" et "end". La fonction se termine par le séparateur point virgule : ";".

2.7. Les Unités

Les unités sont des bibliothèques de procédures et de fonctions. On peut utiliser les unités prédéfinies dans le langage Pascal ou utiliser celles qu'on a créées.

2.8. Les Labels

Un label est un repère au milieu du programme qui permet d'utiliser les "goto".

A ne jamais utiliser !

2.9. Les Instructions

Le corps du programme principal, comme celui des procédures et des fonctions, est constitué d'instructions. Une instruction c'est soit :

- une affectation
- un appel de procédure
- un test
- une boucle
- un débranchement
- une entrée-sortie

Exemple :

```
Program Equation;
Var
    a, b, Solution : real;
    N : integer;
    q : char;

(* Procedures et fonctions *)
Procedure Equa1 (A, B: real; var X : real; var NbSol : integer);
```

```
Begin
    if A = 0 and B = 0 then begin                    (* test : si *)
        NbSol := -1;                                 (* affectation *)
        exit;                                        (* débranchement *)
    end else begin                                   (* test : sinon *)
    end;                                             (* test : fin de si *)
    if A = 0 and B <> 0 then                         (* test : si *)
        NbSol := 0;                                  (* affectation *)
        exit;                                        (* débranchement *)
    end else begin                                   (* test : sinon *)
    end;                                             (* test : fin de si *)
    NbSol := 1; (* affectation *)
    X := -B/A;                                       (* affectation *)
end;

(* Programme principal *)
Begin
    repeat                                           (* boucle *)
        writeln ('entrez l'équation, d'abord a, puis b');
        read (a);                                    (* entrée-sortie *)
        write (' X + ');                             (* entrée-sortie *)
        read (b);                                    (* entrée-sortie *)
        writeln (' = 0');                            (* entrée-sortie *)
        Equa1 (a, b, Solution, N);                   (* appel de procédure *)
        case N of                                    (* test : case *)
            -1   : writeln ('il y a une infinité de solutions');
            0    : writeln ('il n'y a pas de solution');
            1    : writeln ('la solution est : ', Solution);
        else
        end;                                         (* test : fin de case *)
        write ('tapez q pour quitter ');             (* entrée-sortie *)
        readln (q);                                  (* entrée-sortie *)
    until q = 'q';(* fin de boucle *)
end.
```

B. Précisions

1. Les Instructions

Un programme, une procédure ou une fonction, c'est une succession d'instructions. Une instruction c'est soit :

- une affectation
- un appel de procédure
- un test
- une boucle
- un débranchement
- une entrée-sortie

1.1. L'affectation

Une affectation permet de donner une valeur à une variable.

Exemples Pascal

```
Program AffectationEntier ;
var
    N : integer;
begin
    N:=1;
    Writeln (N);                                (* affiche "1" *)
    N := N+1;
    Writeln (N);                                (* affiche "2" *)
end.
```

```
Program AffectationChaine;
Var
    ESTP : string;
begin
    ESTP := 'ESTP est une bonne école d'ingénieurs';
    Writeln (ESTP);                    (* affiche "ESTP est une bonne
                                              école d'ingénieurs" *)
    ESTP := ESTP +' avec de bons professeurs';
    Writeln (ESTP);              (* affiche "ESTP est une bonne école
                                 d'ingénieurs avec de bons professeurs " *)
end.
```

Structure Pascal

```
var := expression;
```

"var" est un identificateur de variable.

On verra au § B. 3 la structure générale d'une expression.

Structure Algo

```
var <- expression;
```

1.2. L'appel de procédure

Un appel de procédure permet d'exécuter le programme correspondant à la procédure, les variables ou les expressions fournies au moment de l'appel venant prendre la place des variables de définition de la procédure, le remplacement se faisant dans l'ordre : la première variable d'appel remplace la première variable de définition, idem pour la seconde, la troisième, etc.

Lors de l'appel, les variables de définition de la procédure qui sont en entrée peuvent être remplacées par des variables, des constantes, des valeurs ou des expressions. Dans tous les cas, on évalue la valeur fournie en entrée et cette valeur est utilisée par la procédure. On dit aussi que ces variables, ou paramètres, sont passées par valeur.

Lors de l'appel, les variables de définition de la procédure qui sont en sortie sont forcément remplacées par des variables. On dit aussi que ces variables, ou paramètres, sont passées par adresse.

Exemples

```
A:=1;
B:=2;
Equa1 (A, B, N, Sol);          (* La procédure résout l'équation pour
                                  A = 1 et B = 2. A la sortie de la procédure
                                  Sol vaut -2 N vaut 1 *)
Equa1 (3, -6, N, Sol);         (* à la sortie, Sol vaut 2 et N vaut 1 *)
Equa1 (A+B, 3, N Sol);         (* à la sortie, Sol vaut -1 et N vaut 1 *)
```

Remarque

Une variable de définition est toujours différente d'une variable d'appel, même si les deux ont le même nom. Le nom d'une variable de définition est un nom générique tandis que le nom d'une variable d'appel est un nom effectif.

1.3. Le test

Il y a deux sortes de test : le test bivalent et le test plurivalent.

1.3.1. Le test bivalent : IF

Le test bivalent permet de choisir d'exécuter une série d'instructions plutôt qu'une autre.

Exemple Pascal

```
Program IfThenElse;
var
    N : integer;
begin
```

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

Structure Pascal

```
    if condition then begin
        instructions
    end else begin
        instructions
    end;
```

On verra au § B.3 la structure générale d'une condition.

Structure Algo

```
    condition ? instructions | instructions ¿
```

1.3.2. Le test plurivalent : CASE

Le test plurivalent permet de choisir les instructions à exécuter selon l'appartenance d'une variable à un ensemble de valeurs.

Exemples Pascal

```
    Program CaseOf;
    var
        C : char.
    begin
        readln (C);
        case C of
            'a'..'z' :
                writeln ('minuscule');
            'A'..'Z' :
                writeln ('majuscule');
        else
            writeln ('code ascii : ', ord(c) );
        end;
    end.
```

```
Program CaseOf;
var
    N : integer ;
begin
    readln (N);
    case N of
        0, 2, 4, 6, 8 :
            writeln ('chiffre pair');
        1, 3, 5, 7, 9 :
            writeln ('chiffre impair');
        10..100 :
            writeln ('entier compris entre 10 et 100);
    else
        writeln ('entier supérieur à 100');
    end;
end.
```

Structure Pascal

```
case expression of
    domaine :
        instructions;
    domaine :
        instructions;
    etc.
else
    instructions
end;
```

Un domaine est une suite de nombres ou de chaînes de caractères séparés par des virgules : ",".

Structure Algo

```
< expression ? domaine : instructions | domaine : instructions |
domaine : instructions | etc...  | instructions >
```

1.4. La boucle

La boucle permet de répéter plusieurs fois une même série d'instructions.

Il y a 3 sortes de boucle en Pascal:

- La boucle avec compteur
- La boucle tant que

- La boucle jusqu'à ce que

1.4.1. La boucle avec compteur : FOR

Exemples Pascal

```
Program BoucleFor;
(* calcul de la somme des N premiers nombres*)
Var
    i, N, Som : integer;
Begin
    write ('entrez un entier : ');
    readln (N);
    Som := 0;
    for i := 0 to N do begin
        Som := Som + i;
    end;
    writeln ('la somme des ', N,' premiers nombres vaut : ', Som );
end.
```

ou alors :

```
Program BoucleForDownto
(* calcul de la somme des N premiers nombres*)
Var
    i, N, Som : integer;
Begin
    write ('entrez un entier : ');
    readln (N);
    Som := 0;
    for i := N downto 0 do begin
        Som := Som + i;
    end;
    writeln ('la somme des ', N,' premiers nombres vaut : ', Som );
end.
```

Structure Pascal

```
for i := début (to ou downto) fin do begin
    instructions;
end;
```

"début" et "fin" sont des expressions contenant des valeurs entières.

Structure Algo

```
  fin
{i:début  instructions }
```

pour des raisons typographiques, on écrira aussi dans ce polycopié :

```
{(i:début à fin) instructions }
```

1.4.2. La boucle tant que : WHILE

Exemple Pascal

```
Program BoucleWhile;
(* calcul de la somme des N premiers nombres*)
Var
     i, N, Som : integer;
Begin
     write ('entrez un entier : ');
     readln (N);
     Som := 0;
     i:=0
     while i < N do begin
          i := i + 1;
          Som := Som + i;
     end;
     writeln ('la somme des ', N,' premiers nombres vaut : ', Som );
end.
```

Structure Pascal

```
while condition do begin
     instructions;
end;
```

Une condition est une expression dont la valeur est de type booléen. Le chapitre 3 décrit la structure générale d'une expression.

Structure Algo

```
{/condition instructions }
```

pour des raisons typographiques, on écrira aussi dans ce polycopié :

```
{(/condition) instructions }
```

1.4.3. La boucle jusqu'à ce que : REPEAT

Exemple Pascal

```
Program BoucleRepeat;
(* calcul de la somme des N premiers nombres*)
Var
    i, N, Som : integer;
Begin
    write ('entrez un entier : ');
    readln (N);
    Som := 0;
    i:=0
    repeat
        SOM := SOM + i;
        i := i + 1;
    until  i > N;
    writeln ('la somme des ', N,' premiers nombres vaut : ', Som );
end.
```

Structure Pascal

```
repeat
    instructions;
until condition ;
```

Structure Algo

```
{ instructions }/condition
```

pour des raisons typographiques, on écrira aussi dans ce polycopié :

```
{ instructions (/condition)}
```

1.4.4. La boucle sans fin

La boucle sans fin est une quatrième sorte de boucle qui n'existe pas directement en Pascal.

Sa structure algo est la suivante :

Structure Algo

```
{ instructions }
```

son meilleur équivalent Pascal est :

Structure Pascal

```
while TRUE do begin
    instructions;
end;
```

1.5. Les débranchements

Avec la boucle sans fin, apparaît la nécessité de la notion de débranchement. Il faut pouvoir sortir de la boucle.

On peut définir deux types de débranchement :

- les débranchements structurés
- les débranchements non structurés

1.5.1. Les débranchements structurés

Ils permettent de sortir d'un bloc d'instructions : soit une boucle, soit une procédure.

• "break" fait sortir d'une boucle

Un "break", utilisé dans une boucle, est forcément utilisé dans un test, sans quoi toutes les instructions qui le suivent seraient inutiles ainsi que la boucle elle-même.

Structure Pascal du break

```
for i := début (to ou downto) fin do begin

    ...
    if
        ...
        break;
        ...
    end;
    ...
end;
```

```
while TRUE do begin
```

```
        ...
        if
            ...
            break;
             ...
        end;
        ...
    end;
```

```
    repeat
        ...
        if
            ...
            break;
             ...
        end;
        ...
    until condition ;
```

Structure Algo du break

```
{ ... ?... ! ... ¿ ... }
```

• "exit" fait sortir d'une procédure et d'une fonction

Un "exit", utilisé dans une procédure, est forcément utilisé dans un test, sans quoi toutes les instructions qui le suivent seraient inutiles ainsi que le débranchement lui-même.

Structure Pascal de l'exit

```
    procedure NomProc (...)
    ...
    begin
        ...
        if
            ...
            exit;
             ...
        end;
        ...
    end;
```

```
function NomFonc (...)
...
begin
    ...
    if
        ...
        exit;
        ...
    end;
    ...
end;
```

Structure Algo de l'exit

```
[ ... ? ... | ... ¿ ... ]
              *
```

• next fait passer au suivant dans la boucle

Le next n'existe pas en Pascal. Pour le programmer, il faut utiliser les débranchements non structurés. Le next, comme les autres débranchements structurés, permet de désimbriquer les tests. On peut toujours réimbriquer les tests pour éliminer ces débranchements.

Exemple Pascal

Cet exemple reprend la procédure "Equa1" présentée au § A.2.9 et la réécrit en éliminant les débranchements. Remarquez comment les tests sont réimbriqués en conséquence.

```
Procedure Equa1 (A, B: real; var X : real; var NbSol : integer);
Begin
    if A = 0 and B = 0 then begin
        NbSol := -1;
    end else begin
        if A = 0 and B <> 0 then
            NbSol := 0;
        end else begin
            NbSol := 1;
            X := -B/A;
        end;                          (* test : fin de si A = 0 et B <> 0 *)
    end;                               (* test : fin de si A = 0 et B = 0 *)
end;
```

1.5.2. Le débranchement non structuré : goto n'importe où

Un débranchement non structuré permet de passer d'un point du programme à n'importe quel autre point du programme repéré par un marqueur.

Le "goto" en Pascal permet ce débranchement. Les "label" permettent de définir des marqueurs.

Le "goto" c'est "goto" n'importe où!

Le "goto" est un résidu des langages de programmation précédant le Pascal car les tests, les boucles et les procédures permettent de ne pas utiliser de "goto". C'est le principe même de la programmation structurée.

On peut et on doit toujours éviter d'utiliser un "goto" !

1.6. Les entrées-sorties

Les procédures d'entrées/sorties permettent au programme de communiquer avec l'extérieur, le plus souvent avec l'utilisateur c'est-à-dire avec le clavier et l'écran, parfois avec des fichiers c'est-à-dire avec le disque dur ou une disquette (on traitera ce cas dans le chapitre concernant les fichiers), parfois avec d'autres périphériques (l'imprimante, la souris, etc.), parfois avec d'autres programmes.

On ne traite dans ce paragraphe que des procédures d'entrées-sorties qui permettent de communiquer avec l'écran et le clavier.

1.6.1. Écrire à l'écran : Write et Writeln

La procédure :

- **Write (liste d'expression)**

permet d'écrire le contenu des expressions listées à l'écran à partir de la position courante du curseur de l'écran. Toutes les expressions sont des paramètres en entrée.

La procédure :

- **Writeln (liste d'expression)**

fonctionne comme Write mais le curseur de l'écran effectue en plus un retour à la ligne après avoir affiché le contenu des expressions.

Il est possible de formater l'affichage selon les principes présentés dans les exemples ci-dessous.

Exemples Pascal

```
Writeln ('essai');                          (* affiche : 'essai' *)
Writeln ('essai':13);                       (* affiche :       essai' *)
b := FALSE;
Writeln (b);                                (* affiche : 'FALSE' *)
Writeln (not(b):10);                        (* affiche :      TRUE' *)
Writeln (1,234567);                         (* affiche : 1.2E+0 *)
Writeln (1,234567:5:3);                     (* affiche :  1.234 *)
```

Structure Algo du Writeln

```
afficher (x);
```

ou encore :

```
écran <- (x);
```

1.6.2. Lire au clavier : Read, Readln

La procédure :

- **Read (liste de variables);**

permet de donner des valeurs à une liste de variables à partir du clavier. La valeur donnée au clavier est affichée à l'écran.

La procédure :

- **Readln (liste de variables);**

fonctionne comme Read mais effectue en plus un retour à la ligne du curseur à l'écran.

Exemples :

```
Readln (x);
Readln (x, c);
```

Structure Algo du Readln

```
lire (x);
```

ou encore :

```
x <- clavier;
```

Remarques :

La fonction

- **ReadKey ;**

renvoie le caractère lu au clavier mais ne l'affiche pas à l'écran.

La fonction booléenne

- **KeyPressed ;**

renvoie vrai quand une touche du clavier est enfoncée.

Ces fonctions sont utiles pour une gestion plus fine de l'interface utilisateur. Elles nécessitent de faire appel à la bibliothèque CRT.

2. Les types

Le type est une des caractéristiques d'une variable. Il est lui-même caractérisé par :

- son nom (un identificateur ou un mot-clé);
- son domaine de valeurs possibles pour les variables;
- son format, c'est-à-dire la taille de l'espace mémoire nécessaire pour coder les valeurs et la règle de lecture de cette mémoire;
- l'ensemble des opérations possibles pour les variables.

Le Pascal fournit des types standards et permet de construire de nouveaux types.

Les types du Pascal sont les suivants :

Types		
Simples		**Complexes**
Standards	**Définis par l'utilisateur**	
Entiers	Intervalles	Tableaux
Réels	Énumérés	Chaînes de caractères
Caractères		Ensembles
Booléens		Enregistrements
Pointeurs		Fichiers

2.1. Les entiers

2.1.1. Définition

Il existe plusieurs types d'entiers avec plusieurs domaines de valeurs. Ils sont présentés dans le tableau ci-dessous :

Nom du Type	**Format**	**Domaine de valeurs**
Integer	16 bits, signé (2 octets)	-32 768 à +32 767
Shortint	8 bits, signé (1 octet)	-128 à +127
Longint	32 bits, signé (4 octets)	- 2 147 483 648 à +2 147 483 647
Byte	8 bits, non signé	0 à 255
Word	16 bits, non signé	0 à 65 535

Seul le type "Integer" est un type du Pascal standard. Les autres sont des types du Turbo Pascal.

2.1.2. Algèbre

Affectations

Si la variable "A" est une variable de type entier, on ne peut affecter à "A" qu'une valeur entière :

- A := 3;
- A := B; avec B entier.
- A := expression; le résultat de l'évaluation de l'expression doit être de type entier dans tous les cas.

Comparaisons

On peut comparer les entiers et les réels avec les six opérateurs classiques de comparaison.

- A = B
- A <> B
- A <= X
- A < X
- A >= expression
- A > 5.3

Opérateurs

6 opérateurs sont applicables aux entiers :

- "+" : addition
- "-" : soustraction
- "-" : opposé
- "*" : multiplication
- div : division entière
- mod : reste de la division entière

Fonctions entières

7 fonctions produisent un résultat entier :

Nom	Rôle	Type de l'entrée	Type du résultat
ABS	valeur absolue	entier	entier
ROUND	arrondi à l'entier le plus proche	réel	entier
SQR	carré	entier	entier
TRUNC	partie entière	réel	entier
ORD	numéro d'ordre : c'est la valeur même de l'entier	entier	entier
PRED	valeur du précédent	entier	entier
SUCC	valeur du suivant	entier	entier

2.2. Les réels

2.2.1. Définition

Il existe plusieurs types de réels avec plusieurs domaines de valeurs. Ils sont présentés dans le tableau ci-dessous :

Nom du type	Signification	Format	Domaine de valeurs	Chiffres significatifs
Real	réel	6 octets	2.9×10^{-39} à $1.7 \times 10^{+38}$	11
Single	simple	4 octets	1.5×10^{-45} à $3.4 \times 10^{+38}$	7
Double	double	8 octets	5.0×10^{-324} à $1.7 \times 10^{+308}$	15
Extended	étendu	10 octets	1.9×10^{-4951} à $1.1 \times 10^{+4932}$	19

Comp	entier (!)	8 octets	$-9.2x10^{-18}$ à $9.2x10^{+18}$	19

Seul le type "Real" est un type du Pascal standard. Les autres sont des types du Turbo Pascal.

2.2.2. Algèbre

Affectations

Si la variable "X" est une variable de type réel, on peut affecter à "X" des valeurs entières ou réelles :

- X := 3;
- X := A; avec A entier.
- X := 3.2;
- X := Y; avec Y réel.
- X := expression; le résultat de l'évaluation de l'expression peut être de type entier ou réel.

Comparaisons

On peut comparer les entiers et les réels avec les six opérateurs classiques de comparaison.

- X = A
- X <> A
- X <= Y
- X < Y
- X >= expression
- X > 5.3

Opérateurs

4 opérateurs sont applicables aux réels :

- "+" : addition
- "-" : soustraction
- "-" : opposé
- "*" : multiplication

Fonctions réelles

10 fonctions produisent un résultat réel :

Nom	Rôle	Type de l'entrée	Type du résultat
ABS	valeur absolue	réel	réel
ARCTAN	arc tangente (en radians)	entier ou réel	réel
COS	cosinus	entier ou réel	réel
EXP	exponentielle	entier ou réel	réel
FRAC	partie fractionnaire	réel	réel
INT	partie entière	réel	réel
LN	logarithme népérien	entier ou réel	réel
SIN	sinus	entier ou réel	réel
SQR	carré	réel	réel

SQRT	racine carré	entier ou réel	réel

2.3. Les booléens

2.3.1. Définition

Type	Format	Domaine de valeurs
Boolean	1 octet	True ou false

On considère en général que "True" vaut 1 et que "False" vaut 0.

2.3.2. Algèbre

Affectations

Si la variable "Bool" est une variable de type booléen, on peut affecter à "Bool" les constantes "True" et "False", des valeurs booléennes, le résultat de comparaisons :

- Bool := True;
- Bool : = Bool2; avec Bool2 booléen.
- Bool := A = 3; Bool est affecté de la valeur logique (vraie ou fausse) de la comparaison de A et de 3.

Comparaisons

On peut comparer deux booléens entre eux, étant convenu que : "False < True" est vrai.

- Bool = Bool2
- Bool > Bool2

On peut aussi comparer des booléens avec des résultats de comparaisons, ou des résultats de comparaisons entre eux :

- Bool = (A = 3)

Opérateurs

4 opérateurs sont applicables aux booléens :

- not
- and
- or (ou inclusif : l'un ou l'autre ou les deux)
- xor (ou exclusif : fromage ou dessert !)

Bool1	Bool2	not (Bool1)	Bool1 and Bool2	Bool1 or Bool2	Bool1 xor Bool2
true	true	false	true	true	false
true	false	false	false	true	true
false	true	true	false	true	true
false	false	true	false	false	false

Fonctions ordinales et booléens

Les fonctions ordinales sont utilisables avec les booléens. Cependant, elles n'ont pas un grand intérêt.

Nom	Rôle	Type de l'entrée	Type du résultat
ORD	ressort 0 si l'entrée vaut False, 1 si elle vaut True	booléen	entier
PRED	fournit l'inverse de l'entrée	booléen	booléen
SUCC	fournit l'inverse de l'entrée	booléen	booléen

2.4. Les caractères

2.4.1. Définition et code ASCII

Nom du type	Format	Domaine de valeurs
Char	1 octet	Code ASCII étendu

Les variables de type Char peuvent prendre toutes les valeurs correspondant au Code ASCII étendu.

ASCII signifie : American Standard Code For Information Interchange.

- Les codes de 0 à 31 et le code 127 correspondent à des caractères de contrôle qui ne peuvent ni être affichés, ni être imprimés (le code 7 active une sonnerie!).
- Les lettres majuscules, de "A" à "Z" sont codées de 65 à 90.
- Les minuscules, de "a" à "z" de 97 à 122.
- Les chiffres, de "0" à "9" de 48 à 57.

2.4.2. Algèbre

Affectations

Si la variable "Cara" est une variable de type caractère, on peut affecter à "Cara" un caractère ou une chaîne de caractère :

- Cara := 'a';
- Cara := Cara2; avec "Cara2" de type caractère.

Si on affecte une chaîne de caractères à un caractère, celui-ci prend la valeur du premier caractère de la chaîne :

- Cara := 'bonjour';

Cara vaut : 'b'.

Comparaisons

On peut comparer les caractères et les chaînes de caractères. Le résultat de la comparaison c'est le résultat de la comparaison des valeurs ASCII correspondantes :

- 'A' < 'B' est vrai (Code ASCII de 'A' = 65, de 'B' = 66);
- 'A' < 'a' est vrai (Code ASCII de 'A' = 65, de 'a' = 97);

Selon le principe de classification alphabétique habituelle, on considère que :

- 'A' < 'Axx' est vrai.

Fonctions applicables aux caractères

Nom	Rôle	Type de l'entrée	Type du résultat
CHR	fournit le caractère correspondant au code ASCII de la valeur entière en entrée	entier	caractère
ORD	fournit la valeur entière du code ASCII du caractère en entrée	caractère	entier
PRED	fournit le caractère précédent dans l'ordre ASCII	caractère	caractère
SUCC	fournit le caractère suivant dans l'ordre ASCII	caractère	caractère

2.5. Les chaînes de caractères

2.5.1. Définition

Une chaîne de caractère est une suite ordonnée de caractères.

Exemple Pascal

```
Type
    Chaine10 : String[10];
Var
    ch1, mot : chaine10;
    ch2 : string;
```

Par défaut, une chaîne de caractères contient au maximum 255 caractères. Elle est codée sur 256 octets car un octet est utilisé pour garder la longueur de la chaîne.

Une chaîne déclarée String[10] contient 10 caractères maximum et est représentée sur 11 octets, le onzième octet étant utilisé pour garder la longueur de la chaîne.

2.5.2. Algèbre

Affectations

Si la variable "Mot" est une variable de type chaîne de caractères, on peut lui affecter une expression de type chaîne de caractères ou de type caractère :

- mot := 'bonjour';
- mot := mot1; avec "mot1" de type chaîne de caractères.
- mot := C; avec "C" de type caractère.
- mot := expression; avec une expression de type caractère ou chaîne de caractères.

Comparaisons

On peut comparer les caractères et les chaînes de caractères. Le résultat de la comparaison c'est le résultat de la comparaison des valeurs ASCII correspondantes :

- 'Arrive' < 'arrive' est vrai (Code ASCII de 'A' = 65, de 'a' = 97);
- 'ARRIVE' > 'ARRIERE' est vrai (Code ASCII de 'V' = 86, de 'E' = 69);

C'est une comparaison par ordre alphabétique : le type est ordonné.

Opérateurs

1 opérateur est applicable aux chaînes de caractères :

- "+" : concaténation

Exemple Pascal

```
Mot := 'voici';
Mot1 := 'un';
Mot2 := 'exemple';
Phrase := 'Mot' + ' ' + Mot1 + ' ' + Mot2 + ' de concatenation';
                (* Phrase contient : 'voici un exemple de concatenation' *)
```

Cet opérateur est équivalent à la fonction concat.

Fonctions applicables aux chaînes de caractères

Nom	Description	Type des entrées	Type du résultat
Concat	Concat (Ch1, Ch2, Ch3,...) concatène les chaînes en entrée	chaînes ou caractères	chaîne
Copy	Copy (Chaine, Position, Longueur) extrait une partie (sous-chaîne) d'une chaîne, d'une longueur donnée, à partir d'une position donnée.	une chaîne 2 entiers	chaîne
Delete	Delete (Chaine, Position, Longueur) supprime dans une chaîne un nombre donné de caractères à partir d'une position donnée dans la chaîne.	une chaîne 2 entiers	chaîne
Length	Length (Chaine) fournit la longueur de la chaîne.	chaîne	entier
Pos	Pos (Chaine1, Chaine2) situe une sous-chaîne dans une chaîne.	2 chaînes	entier
Str	Str (RéelOuEntier, Chaine) transforme un réel ou un entier en chaîne.	un entier ou un réel une chaîne	chaîne
Val	Val (Chaine, RéelOuEntier, OK) transforme une chaîne en entier ou en réel et ressort un paramètre qui vaut 0 si la transformation s'est bien passée.	une chaîne un entier ou un réel un entier	entier ou réel

Exemples Pascal

```
Phrase := concat (Mot, ' ', Mot1, ' ', Mot2, Mot3, ' de concatenation');
Mot := copy ('bonjour', 2, 4);                    ( * Mot vaut 'onjo' *)
```

```
Delete ('bonjour', 3, 4);                    (* Mot vaut 'bor' *)
N := Length ('bonjour');                     ( * N vaut 7 *)
N := Pos ('jo', 'bonjour');                  (* N vaut 4 *)
N := 255;
Str (N, Mot);                                (* Mot vaut '255' *)
X := 1234,5678;
Str (X:7:2, Mot);                            (* Mot vaut '1234,56' *)
Val (Mot, Y, ok);                  (* Y vaut 1234,56 et ok vaut 0 *)
```

2.6. Les tableaux

2.6.1. Définition

Un type tableau est un type regroupant plusieurs variables de mêmes types, ordonnées et accessibles individuellement par leurs indices.

Il existe des tableaux à un indice (on dit encore une dimension) et des tableaux à deux indices (deux dimensions).

Exemple Pascal

```
Type
    Vecteur = Array [1..10] of integer;
    Matrice = Array [1.10,1..10] of real;
Var
    UnVecteur : Vecteur;
    Une Matrice : Matrice;
Begin
    Vecteur[1] := 5;
    Matrice[1,1] := 4.3;
```

2.6.2. Algèbre

Les opérations applicables aux variables d'un tableau sont les mêmes que celles applicables au type de la variable.

Affectation globale

On peut affecter une variable de type tableau dans une autre variable de type tableau à condition que le type des deux tableaux soit le même et qu'il ait été déclaré par l'utilisateur. Cette affectation revient à recopier un tableau dans un autre tableau.

Exemple Pascal

```
Type
    Tableau = Array [1..10] of integer;
Var
    Tab1, Tab2 : Tableau ;
Begin
    ...
```

```
Tab1 : = Tab2;
```

2.7. Les enregistrements

2.7.1. Définition

Un type enregistrement est un type regroupant plusieurs types différents appelés sous-types.

- les noms du type enregistrement et des sous-types sont choisis par l'utilisateur.
- les sous-types peuvent être des types standards du Pascal ou des types créés par l'utilisateur.

Une variable de type enregistrement est une variable regroupant sous un même nom un ensemble de variables de types différents et dont le contenu est accessible par le nom de la variable enregistrement associé au nom du sous-type.

Exemple Pascal

```
Type
    Date = Record
        Année : integer;
        Mois : 1..12;
        Jour : 1..31;
    end;
Var
    UneDate : Date;
```

2.7.2. Algèbre

Les opérations applicables aux variables d'un enregistrement sont les mêmes que celles applicables au type de la variable.

Affectation et instruction WITH

Pour accéder au contenu des variables, on écrit :

```
UneDate.Année := 1998;
UneDate.Mois := 1;
UneDate.Jour := 12;
```

On peut utiliser l'instruction WITH pour éviter d'avoir à répéter 'UneDate' :

```
With UneDate Do Begin
    Année := 1998;
    Mois := 1;
    Jour := 12;
end;
```

2.8. Les énumérations

2.8.1. Définition

Un type énuméré est un type dont :

- le nom est choisi par l'utilisateur
- l'ensemble des valeurs possibles est défini par l'utilisateur.

Exemple Pascal

```
Type
    Semaine =
        (lundi, mardi, mercredi, jeudi, vendredi, samedi, dimanche);
Var
    Jour : Semaine;
```

2.8.2. Algèbre

Affectations

si "Jour" est une variable de type énuméré "Semaine", on peut affecter à "Jour" une des valeurs de l'énumération du type :

```
Jour := lundi;
Jour := Jour2;                          (* avec Jour2 de type Semaine *)
```

Mais, les types énumérés n'ont pas de représentation externe, ce qui veut dire qu'on ne peut pas écrire :

```
writeln (Jour);                         (* IMPOSSIBLE !!! *)
```

Il faudra passer par une procédure :

```
Procedure WritelnJour (Jour : Semaine)
Begin
    Case Jour of
        lundi :     writeln ('lundi');
        mardi : writeln ('mardi');
        mercredi :  writeln ('mercredi');
        jeudi :     writeln ('jeudi');
        vendredi :  writeln ('vendredi');
        samedi :    writeln ('samedi');
        dimanche :  writeln ('dimanche');
    else
    end;
end;
```

Cette procédure sera appelée ainsi :

```
WritelnJour (Jour);
```

Comparaisons

On peut comparer des variables de même type énuméré. L'ordre, c'est l'ordre des valeurs du type tel qu'il apparaît dans sa définition.

- Jour < mercredi est vrai si "Jour" vaut "lundi" ou "mardi";

Fonctions applicables aux énumérés

Nom	Rôle	Type de l'entrée	Type du résultat
type	fournit la valeur du type correspondant au numéro de rang passé (les numéros de rang démarrent à 0). "*type*", c'est le nom du type énuméré.	entier	énuméré
ORD	fournit le numéro de rang du type passé (les numéros de rang démarrent à 0).	énuméré	entier
PRED	fournit le numéro de rang du type passé précédent.	énuméré	énuméré
SUCC	fournit le numéro de rang du type passé suivant.	énuméré	énuméré

Exemple Pascal

```
Program TypeEnumere;
Var
    Jour, Demain, Hier : Semaine;

Begin
    Jour := mardi;
    N := ord (Jour);      (* N vaut 1 car l'énumération démarre à 0 *)
    Demain := succ (Jour);              (* Demain vaut mercredi *)
    Hier := pred (Jour);                        (* Hier vaut lundi *)
    Jour := Semaine (0);                        (* Jour vaut lundi *)
end.
```

2.9. Les intervalles

2.9.1. Définition

Un type intervalle est un type dont :

- le nom est choisi par l'utilisateur
- l'ensemble des valeurs possibles appartient à un sous-ensemble du domaine d'un type scalaire discret (Integer, Boolean, Char et Énumération) dont on précise les deux bornes d'un intervalle.

Exemple Pascal

```
Type
```

```
    Majuscule = 'A'..'Z';
    Note = 0..20;
    Semaine =                                   (* type énuméré *)
        (lundi, mardi, mercredi, jeudi, vendredi, samedi, dimanche);
    JourTravail = lundi .. vendredi;(* intervalle d'un type énuméré *)
Var
    Lettre : Majuscule;
    UneNote : Note;
```

2.9.2. Algèbre

Les opérations applicables aux variables de types intervalles sont les mêmes que celles applicables aux variables du type scalaire discret de l'intervalle.

2.10. Les ensembles

2.10.1. Définition

Un type ensemble est un type dont :

- le nom est choisi par l'utilisateur;
- l'ensemble des valeurs possibles est défini par l'utilisateur par un type ordinal dont les valeurs ordinales sont comprises entre 0 et 255, soit les types "char", "byte", énumération ou intervalle.

Exemple Pascal

```
Type
    Lettres = Set of 'A' ..'Z';
    Caracteres = Set of Char;
    Chiffres = Set of 0..9;
Var
    EnsLettres : Lettres;
    EnsChiffres : Chiffres;
    EnsChar : Caractères;
```

2.10.2. Algèbre

Affectations

Pour initialiser un ensemble, on écrit :

```
EnsLettres := [];(* ensemble vide *)
EnsChiffres := [0, 2, 4, 6, 8];          (* ensembles des chiffres pairs *)
EnsChar := ['a', 'e', 'i', 'o', 'u', 'y'];          (* ensemble des voyelles *)
```

Opérateurs

3 opérateurs sont applicables aux ensembles :

- "+" : **réunion** Ens1 + Ens2 (Ens1 union Ens2)

- "*" : **intersection** Ens1 * Ens2 (Ens1 inter Ens2)
- "-" : **différence** Ens1 - Ens2

Exemple Pascal

Pour insérer ou supprimer un élément dans un ensemble, on écrit :

```
EnsLettres := EnsLettres  + ['A'];
EnsLettres := EnsLettres  - ['Z'];
EnsChiffres := EnsChiffres + [3];
EnsChiffres := EnsChiffres - [7];
EnsChar := EnsChar + ['*'];
```

Comparaisons

On peut comparer des ensembles :

- Ens1 = Ens2 est vrai si les deux ensembles contiennent les mêmes éléments
- Ens1 >= Ens2 est vrai si Ens2 est inclus dans Ens1.
- Ens1 <= Ens2 est vrai si Ens1 est inclus dans Ens2.

2.11. Les fichiers

2.11.1. Définition

Un fichier est une variable différente de toutes les autres variables utilisées jusqu'à présent car elle est conservée durablement indépendamment du programme.

Du point de vue le plus général, c'est-à-dire du point de vue du système d'exploitation de l'ordinateur, un fichier est une suite d'octets regroupés sous un même nom et stockés durablement.

Pour le langage Pascal, un fichier est une suite de variables de mêmes types conservées durablement.

Exemples Pascal :

```
Type
    FichierDeReels = File of real;
    FichierDEntiers = File of integer;
    Date = Record
        Année : integer;
        Mois : 1..12;
        Jour : 1..31;
    end;
    FichierDeStructure = File of Date;
    FichierDeTexte = TEXT;
```

Le type TEXT est celui d'un fichier de chaînes de caractères de longueurs variables terminées par les deux caractères spéciaux : retour chariot (ASCII = 13) et saut de ligne (ASCII = 10) qui codent la fin de la chaîne et le passage à la ligne.

Le type TEXT est donc un cas particulier de File of Char.

Une variable de type fichier peut être comprise comme un enregistrement qui contient deux informations :

- le nom du fichier;
- l'adresse d'un élément du fichier;

Cette adresse permet de circuler dans le fichier, c'est pourquoi on choisit ici de préfixer les variables de type fichier par "Pt" pour signifier que ce sont des pointeurs.

Exemples Pascal :

```
Var
    PtFichTexte : FichierDeTexte ;
    PtFichReels : FichierDeReels;
    PtFichEntiers : FichierDEntiers;
    PtFichStruct : FichierDeStructure;
    PtFichTexte2 : TEXT;
    PtFichReels2 : File of real;
```

2.11.2. Algèbre

Ouverture d'un fichier existant déjà :

Pour accéder au contenu d'un fichier qui existe déjà, on utilise les deux procédures suivantes :

Assign (Var PtFich : File; NomEtCheminFichier : String);

Reset (Var PtFich : File);

- NomEtCheminFichier : contient le nom du fichier et éventuellement le répertoire du fichier (sous la forme DOS standard : C: \ ... \ chezmoi \ monfichier). Par défaut, le fichier est cherché dans le répertoire courant.
- PtFich : contient l'adresse d'un élément du fichier.

Assign assigne à la variable de type fichier la chaîne de caractères fournie et qui est le nom du fichier. La variable de type fichier est modifiée : elle est en sortie. Le nom du fichier est en entrée.

Reset donne à l'adresse fournie par la variable de type fichier l'adresse du premier élément du fichier dont le nom a été assigné précédemment dans la variable de type fichier. Autrement dit Reset fait pointer le pointeur de fichier sur le premier élément du fichier. La variable de type fichier est donc utilisée (le nom du fichier) et modifiée (le pointeur) : elle est en entrée et en sortie.

Création d'un fichier

Pour créer un nouveau fichier, on utilise les deux procédures suivantes :

Assign (Var PtFich : File; NomEtCheminFichier : String);

Rewrite (Var PtFich : File);

Rewrite crée un fichier dont le nom a été assigné précédemment dans la variable de type fichier. Rewrite donne à l'adresse fournie par la variable de type fichier l'adresse du début du fichier, vide à ce moment. La variable de type fichier est donc utilisée (le nom du fichier) et modifiée (le pointeur) : elle est en entrée et en sortie.

Attention : si le fichier existait déjà, alors il est détruit.

Lire un élément dans le fichier et passer au suivant

Pour lire un élément du fichier on utilise les procédures suivantes :

Read (Var PtFich : File; Var v1, v2...);

Readln (Var PtFich : File; Var Ligne : String);

La procédure Read est utilisée pour tous les types de fichiers à l'exception des fichiers de type TEXT.

La procédure Readln est utilisée uniquement pour les fichiers de type TEXT.

Read et Readln recopient à partir de l'adresse fournie par la variable de type fichier (PtFich) les octets du fichier dans les variables passées en paramètre (v1, v2, etc.). L'adresse fournie par la variable de type fichier est alors modifiée : c'est celle de l'élément du fichier suivant la dernière lecture (éventuellement la fin du fichier). L'adresse fournie par la variable de type fichier est donc utilisée et modifiée : la variable de type fichier est en entrée et en sortie.

Dans le cas d'un fichier TEXT, Readln lit dans le fichier : soit sur toute la longueur de la chaîne en sortie, soit jusqu'à trouver dans le fichier un retour chariot (ASCII = 13) et saut de ligne (ASCII = 10) , c'est-à-dire les deux caractères de fin de ligne.

Écrire un élément dans le fichier et passer au suivant

Pour écrire un élément dans un fichier on utilise les procédures suivantes :

Write (Var PtFich : File; v1, v2...);

Writeln (Var PtFich : File; Ligne : String);

La procédure Write est utilisée pour tous les types de fichiers à l'exception des fichiers de type TEXT.

La procédure Writeln est utilisée uniquement pour les fichiers de type TEXT.

Write et Writeln écrivent le contenu des variables passées en paramètre (v1, v2, etc.) dans le fichier à l'adresse fournie par la variable de type fichier (PtFich). L'adresse fournie par la variable de type fichier est alors modifiée : c'est celle de l'élément du fichier suivant la dernière écriture (éventuellement la fin du fichier). L'adresse fournie par la variable de type fichier est donc utilisée et modifiée : la variable de type fichier est en entrée et en sortie.

Dans le cas d'un fichier TEXT, le Writeln écrit dans le fichier toute la chaîne de caractères en entrée (Ligne) puis les deux caractères spéciaux de fin de ligne : retour chariot (ASCII = 13) et saut de ligne (ASCII = 10).

Pointer directement sur un élément du fichier

Il est possible de donner à l'adresse fournie par la variable de type fichier l'adresse de n'importe quel élément du fichier en donnant son numéro d'ordre dans le fichier (on démarre la comptabilité à 0). Autrement dit, il est possible de pointer directement sur un élément d'un fichier :

Seek (Var PtFich : File; v1, Rang : Longint);

Seek donne à l'adresse fournie par la variable de type fichier la valeur de l'adresse de l'élément dont le rang est passé en entrée (cette adresse est calculée à partir de l'adresse du début du fichier, du numéro de rang et de la taille des éléments du fichier définie par leur type). L'adresse fournie par la variable de type fichier est modifiée : la variable de type fichier est en sortie, tandis que le numéro de rang est en entrée.

La procédure Seek peut être utilisée pour tous les types de fichiers, sauf les fichiers de type TEXT.

Fermeture d'un fichier

Pour fermer un fichier on utilise la procédure suivante :

Close (Var PtFich : File);

À la sortie de la procédure, l'adresse fournie par la variable de type fichier est nulle ainsi que le nom du fichier de la variable de type fichier.

Après un Close, la variable de type fichier peut être réutilisée pour une nouvelle assignation.

Détection de la fin d'un fichier

EOF (PtFich : File) : boolean;

La fonction EOF est Vrai quand l'adresse fournie par la variable de type fichier est celle de la fin du fichier.

Exemples Pascal

```
Program AfficherFichierTexte;
Var
    PtFich : Text;
    NomFich , ligne : String;
Begin
    write ('entrer le nom du fichier à afficher');
    readln (NomFich);
    Assign (PtFich, NomFich);
    Reset (PtFich);
    While not EOF (PtFich) do begin
```

```
            readln (PtFich, Ligne);
            writeln (Ligne);
        end;
        Close (PtFich);
end.
```

```
Program EcrireFichier;
Type
    MaStructure = record
        nom : string [10];
        reel : real;
        entier : integer;
    end;
Var
    UneStructure : MaStructure;
    PtFich : File of MaStructure;
    q : char;
Begin
    write ('entrer le nom du fichier à créer : ');
    readln (NomFich);
    Assign (PtFich, NomFich);
    Rewrite (PtFich);
    Repeat
        With UneStructure do begin
            Write ('entrer un nom'); Readln (Nom);
            Write ('entrer un reel'); Readln (Reel);
            Write ('entrer un entier'); Readln (Entier);
        end;
        Write (PtFich, UneStructure);
        Write ('entrer q pour arrêter le remplissage du fichier : ');
        readln (q);
    Until q = 'q';
    Close (PtFich);
end.
```

3. Les expressions

3.1. Éléments constitutifs

Une expression est une formule qui définit le calcul d'une valeur.

Elle se compose :

- d'opérandes, c'est-à-dire :
    - de variables,
    - de constantes,
    - de valeurs, c'est-à-dire :
        - de nombres,
        - de chaînes de caractères,
    - de fonctions, c'est-à-dire :
        - un identificateur réunissant entre parenthèses un ou plusieurs opérandes;
- d'opérateurs, c'est-à-dire de symboles réunissant deux opérandes,
- de parenthèses.

Si le résultat est booléen, l'expression peut être appelée une comparaison.

3.2. Règles de formation

Pour qu'une expression soit bien formée, il faut que :

- sa construction suive les règles générales de formation;
- les types des variables et des évaluations soient concordants.

3.2.1. Règles générales

- Les variables, les constantes et les valeurs sont des expressions.
- Si Exp1 et Exp2 sont des expressions, alors (-Exp1) est une expression, (Exp1 opérateur Exp2) est une expression.
- Si Fonct est un identificateur de fonction, alors Fonct (liste d'expression) est une expression.
- On peut toujours retirer (ou ajouter) des parenthèses tant que cela ne change pas le résultat de l'évaluation de l'expression (c'est-à-dire en appliquant les règles de priorité des opérations)

Remarque :

On a toujours intérêt à ajouter des parenthèses pour rendre l'expression plus immédiatement compréhensible.

3.2.2. Concordance des types

- Le type des paramètres d'appel des fonctions doit correspondre aux types des paramètres de définition.
- Il doit y avoir autant de paramètres d'appel que de paramètres de définition.
- Les opérateurs binaires s'appliquent toujours à deux expressions de même type.
- Le résultat global de l'expression à droite d'une affectation doit être du type de la variable à gauche de l'affectation.
- Exceptions : on peut avoir un entier à la place d'un réel et un caractère à la place d'une chaîne de caractères.

3.3. Règles de l'évaluation : ordre de priorité des opérations

L'évaluation d'une expression se fait selon l'ordre de priorité suivant :

- 1 : les parenthèses et les fonctions;

  on évalue d'abord ce qui se trouve entre parenthèses et donc les fonctions dont la liste des paramètres se trouve entre parenthèses (remarquez que les paramètres d'une fonction peuvent être des expressions complexes, auquel cas il faut d'abord évaluer ces expressions avant d'évaluer la fonction).
- 2 : "-" : opérateur unaire de changement de signe (et pas opérateur binaire de soustraction)
- 3 : not
- 4 : *, /, div, mod, and
- 5 : +, -, or
- 6 : =, <>, <, >, <=, >=, in
- En cas d'égalité de priorité, on évalue de gauche à droite.

Cet ordre de priorité, c'est celui de l'algèbre mathématique habituel.

exemple Pascal

```
ch1 := 'bonjour'
ch2 := 'nj';
B := 3;
C := 9;
A := 5-pos (ch1, ch2)*sqr((B-C)/3);
                          (* A vaut : 5 - 3*sqr(-6/3) soit 5 -3*4 = -7 *)
```

C. Procédures, fonctions, programmes et sous-programmes

La notion de procédure est une notion fondamentale de la programmation en général. C'est aussi une notion fondamentale de la logique et de son application à toutes les sciences et toutes les techniques.

1. Généralités

1.1. Procédure et interface

Un algorithme, quelle que soit sa complexité ou sa simplicité, peut être considéré comme un système traitant des informations fournies en entrées et produisant des informations à la sortie. On appelle ce système "procédure" ou sous-programme. Une procédure d'un point de vue algorithmique c'est aussi bien un programme principal, une procédure ou une fonction.

Ce qu'on a appelé l'en-tête de la procédure, c'est l'interface de la procédure, c'est-à-dire ce qu'elle attend de l'extérieur et ce qu'elle fournit à l'extérieur.

Le corps c'est l'intérieur de la procédure.

Dans le cas d'une procédure standard (par exemple Equa1(a, b, x, NbSol)), l'interface est constituée des variables en entrée et en sortie.

Dans le cas d'une fonction (par exemple Sqr (x)), l'interface est constituée des variables en entrée et de la fonction elle-même en tant que variable en sortie.

Dans le cas d'un programme principal (par exemple le programme Equation) l'interface c'est le clavier qui permettra de donner la valeur de certaines variables et l'écran qui permettra d'afficher la valeur de certaines variables.

1.2. Le discours de la méthode

Les notions de procédure et de fonction, de programme et de sous-programme sont essentielles en programmation parce qu'elles sont liées à la méthode de programmation.

Une fois définies les notions de variables, d'affectation et de procédure, on peut parler de méthodologie : celle-ci va définir les principes qui vont nous permettre d'écrire des algorithmes.

La méthode utilisée pour écrire des algorithmes, on la retrouve dans le Discours de la méthode de Descartes[1] :

1) Un algorithme sera toujours considéré comme une procédure : autrement dit, il faut considérer l'algorithme comme une totalité. À partir de là, il faut commencer par lister toutes les informations traitées par l'algorithme puis toutes les informations produites par l'algorithme, donc

---

[1] René Descartes, Discours de la méthode, 2ème partie, 1637 : "Au lieu de ce grand nombre de préceptes dont la logique est composée, je crus que j'aurais assez des **quatre** suivants, pourvu que je prisse une ferme et constante résolution de ne manquer pas une seule fois à les observer. Le **premier** était de ne recevoir jamais aucune chose pour vraie, que je ne la connusse évidemment être telle [....] Le **second**, de diviser chacune des difficultés que j'examinerais, en autant de parcelles qu'il se pourrait et qu'il serait requis pour les mieux résoudre. Le **troisième**, de conduire par ordre mes pensées, en commençant par les objets les plus simples et les plus aisés à connaître, pour monter peu à peu, comme par degrés, jusqu'à la connaissance des plus composés [....] Et le **dernier**, de faire partout des dénombrements si entiers et des revues si générales, que je fusse assuré de ne rien omettre."

Le **premier** principe est un principe d'évidence, principe de la possibilité d'affirmer le vrai. Le **second** est le principe de la division du tout en partie, du problème en sous problèmes : c'est un principe d'analyse. Le **troisième** est le principe de la reconstruction du tout à partir des parties : c'est un principe de synthèse. Le **dernier** est un principe de systématicité : c'est un principe de totalité. Pour aller du tout aux parties, on va de 4 à 2 et à 1. Pour aller des parties au tout on va de 1 à 3 en passant par 4.

toutes les variables en entrée et en sortie de la procédure en précisant à chaque fois leur type : "*faire partout des dénombrements si entiers et des revues si générales, que je fusse assuré de ne rien omettre".*

2) Il ne faut jamais hésiter, dès que le problème semble un peu difficile, à le décomposer en sous-problèmes dont la solution est remise à plus tard : "*diviser chacune des difficultés que j'examinerais, en autant de parcelles qu'il se pourrait et qu'il serait requis pour les mieux résoudre*". Pour que cette décomposition soit rigoureusement possible, il faut que les sous-problèmes soient présentés comme des procédures ou des fonctions avec leurs listes de variables en entrée et en sortie.

3) Une fois des procédures simples définies, il faut les écrire et les vérifier indépendamment les une des autres, en partant des plus simples pour arriver aux plus complexes : "*ne recevoir jamais aucune chose pour vraie, que je ne la connusse évidemment être telle*".

1.3. La qualité

L'utilisation de procédures concourt à améliorer la qualité d'un programme :

- **lisibilité** : chaque procédure comporte quelques dizaines de lignes avec une action précise à réaliser.
- **fiabilité** : chaque module est développé, testé et mis au point séparément.
- **optimisation** : du fait qu'une procédure peut être utilisée par d'autres procédures, la taille du programme et la durée de développement diminuent.

Pour chaque procédure, on précisera en commentaire:

- son rôle
- la liste des variables en entrée et en sortie et leurs significations
- la liste des variables globales
- le fonctionnement général
- les particularités du fonctionnement

2. Types de procédures et types de variables.

On peut définir trois types de procédures

- les procédures standards
- les sous-programmes ou procédures globales
- les procédures mixtes

On peut définir trois types de variables :

- **les variables globales** : les variables utilisées dans tout le programme.
- **les paramètres** : les variables en entrée et en sortie définies dans l'en-tête des procédures et des fonctions.
- **les variables locales** : les variables uniquement utilisées dans le corps de la procédure.

2.1. les procédures standards

Les procédures et les fonctions qu'on a vues jusqu'à présent sont des procédures standards, c'est-à-dire des procédures ou des fonctions qui sont entièrement définies par leurs en-têtes et leurs corps.

On y trouve :

- des paramètres
- des variables locales
- pas de variables globales

Les procédures standards sont les procédures types des bibliothèques.

Les fonctions sont toujours des procédures standards.

2.2. les procédures sans paramètres

Ce sont des morceaux du programme principal qu'on a regroupé sous le nom d'une procédure. C'est utile pour la lisibilité du programme principale : pour éviter qu'il soit trop important.

On y trouve :

- des variables globales.
- éventuellement des variables locales.
- pas de paramètres.

2.3. les procédures mixtes

Ce sont des procédures standards qui utilisent des variables globales.

On y trouve donc:

- des variables globales.
- des paramètres.
- des variables locales.

exemples Pascal

Ce premier exemple présente un cas de décomposition : pour écrire Equa2, on fait appel à Equa1 dont on ne définit dans un premier temps que l'en-tête.

```
Procedure Equa2 (A, B, C: real; var X1, X2 : real; var NbSol : integer);
Begin
    if A = 0 then begin
        Equa1(B, C, X1, NbSol);
    end else begin
        Delta := sqr(B) - 4*A*C;
        if Delta < 0 then begin
            NbSol := 0;
        end else begin
            if Delta = 0 then begin
                NbSol := 1;
                X1 := -b / (2*a)
            end else begin
                NbSol := 2;
                X1 := (-b -Sqrt(Delta)) / (2*a);
                X2 := (-b +Sqrt(Delta)) / (2*a);
            end;                          (* test : fin de si Delta = 0 *)
        end;                              (* test : fin de si Delta < 0 *)
    end;                                  (* test : fin de si A = 0 *)
end;
```

Ce deuxième exemple montre une procédure standard et deux procédures sans paramètre.

```
Program Tri;
{   Programme de tri d'un tableau
-   variables en entrée :
-       constante : MaxTab : c'est le nombre d'élément du tableau
-       lue au clavier : tab[i] : c'est le tableau à trier
-   variables en sortie :
-       écrit à l'écran : tab[i] : le tableau trié
}
Const
    MaxTab = 10;
Type
    tableau = array [1..MaxTab] of integer;
Var
    tab : tableau;
    q : char;

{   procedure standard de tri d'un tableau
-   en entrée
-       N : le nombre d'éléments du tableau
-       tab : le tableau à trier
-   en sortie
-       tab : le tableau trié
-   fonctionnement
-       on parcourt tous les éléments du tableau. Pour chaque valeur
-       on recherche le numéro de la ligne du minimum des éléments
-        compris entre l'élément en cours et la fin et on inverse
-       la valeur de l'élément du tableau en cours et celle du
minimum
}
Procedure TriTableau (Var Tab: tableau; N : integer)
Var
    NLMin, aux, i, j : integer;
Begin
    for i := 1 to N-1 do begin
        NLMin := i;
        for j := i + 1 to N do begin
            if tab[j] < Tab[NLMin] then begin
                NLMin := j;
            end else begin
            end;
```

```
            end;
            aux := tab[i];
            tab[i] := tab[NLMin];
            tab[NLMin] := aux;
        end;
end;

{   Procédure globale de lecture des données
-   var globale en entrée
-       MaxTab : la taille du tableau
-   var globale en sortie
-       tab : le tableau à lire au clavier
-   fonctionnement
-       lecture d'un tableau au clavier
}
Procedure LectureDesDonnees;
Var
    i : integer;
begin
    writeln ('entrez les ', MaxTab, ' valeurs du tableau);
    for i := 1 to MaxTab do begin
        write ('tab[', i, '] : ');
        readln (tab[i]);
    end;
end;

{   Procédure globale d'affichage des résultats
-   var globale en entrée
-       MaxTab : la taille du tableau
-       tab : le tableau à afficher à l'écran
-   fonctionnement
-       affichage d'un tableau à l'écran
}
Procedure AfficheResultats;
Var
    i : integer;
begin
    writeln ('voici le tableau trie');
    for i := 1 to MaxTab do begin
        writeln ('tab[', i, '] = ', tab[i]);
    end;
```

```
end;

(* Programme principal *)
Begin
    repeat
        LectureDesDonnees;
        TriTableau (Tab, MaxTab);
        AfficheResultats;
        write ('tapez q pour quitter ');
        readln (q);
    until q = 'q';
end.
```

3. Précisions concernant les paramètres

3.1. Paramètres de définition et paramètres d'appel

Les paramètres de définition sont les paramètres définis dans l'en-tête de la procédure.

Les paramètres d'appel sont les paramètres utilisés au moment de l'utilisation (de l'appel) de la procédure. Les paramètres d'appel viennent remplacer dans l'ordre les paramètres de définition.

3.2. Entrée, sortie, valeur, adresse

En Pascal, on précise le mode de passage des paramètres dans l'en-tête des procédures.

Il existe des paramètres en entrée et des paramètres en sortie.

3.2.1. Les paramètres en entrée : par valeur

- La variable dans l'en-tête est définie sans le mot-clé "var"
- Lors de l'utilisation de la procédure ou de la fonction, le paramètre en entrée est remplacé (on dit aussi instancié) par n'importe quelle expression de type compatible (valeur, constante, variable, fonction, expression complexe). C'est la valeur de l'expression instanciée qui sera utilisée dans le corps de la procédure.
- Si un paramètre en entrée est une variable, sa valeur est toujours la même à la sortie de la procédure.
- Dans le corps de la procédure, il ne doit pas y avoir de modification d'une variable en entrée. En effet, celle-ci pouvant être une valeur, cela n'aurait pas de sens. Donc un paramètre en entrée n'est jamais à gauche d'une affectation ou en sortie d'une procédure. Un compilateur bien fait devrait détecter ce genre d'erreur.

3.2.2. Les paramètres en sortie : par adresse

- La variable dans l'en-tête est définie avec le mot-clé "var".
- Lors de l'utilisation d'une procédure (pas d'une fonction), le paramètre en sortie est remplacé (on dit aussi instancié) forcément par une variable de type compatible. C'est l'adresse de la variable instanciée qui sera utilisée dans le corps de la procédure permettant la modification de la valeur de la variable.
- Un paramètre en sortie est forcément une variable. Sa valeur n'est plus la même à la sortie de la procédure.
- Une fonction ne contient qu'une seule variable en sortie et c'est le nom même de la fonction qui joue ce rôle.
- Dans le corps de la procédure (ou de la fonction), il doit y avoir une modification des variables en sortie. Un paramètre en sortie est donc forcément soit à gauche d'une affectation, soit en sortie d'une procédure. Un compilateur bien fait devrait détecter les variables en sortie qui ne vérifient pas ces conditions.

exemples Pascal

```
(* exemples d'appel de Equa1 *)
    Equa1 (0, 0, X, N);                    (* N vaut -1  *)
    Equa1 (2, -4, X, N);                   (* N vaut 1, X vaut 2  *)
    Equa1 (-4, 2, X, N)                    (* N vaut 1, X vaut  0,5  *)
    Equa1 (X, N, X, N);                    (* N vaut 1, X vaut -2  *)
    Equa1 (2, N, 1, N);                    (* ERREUR  *)
end;
```

3.3. La portée des variables

On a vu qu'il y a trois types de variables :

- Les variables globales. Ce sont les variables définies dans l'en-tête du programme principal.
- Les paramètres. Ce sont les variables définies dans l'en-tête des procédures.
- Les variables locales. Ce sont les variables définies entre l'en-tête et le corps de la procédure.

Les variables globales sont accessibles dans le programme principal et dans les procédures, à condition qu'aucun paramètre et aucune variable locale ne portent le même nom.

Il faut toujours éviter au maximum d'utiliser des variables globales dans les procédures.

Les variables locales ne sont accessibles que dans le corps de la procédure où elles sont déclarées.

Les paramètres étant instanciés, ils correspondent selon les cas d'utilisation des procédures, soit à des variables locales, soit à des variables globales.

Exemple Pascal :

On pourra se reporter à l'exemple précédent (le programme Tri). Le programme Equation ci-dessous présente deux procédures standards. Remarquer le traitement des erreurs de saisie fait dans la procédure LireNombre.

```
Program Equation;
Var
    a, b, Solution : real;
    N : integer;

(* Procedures et fonctions *)
Procedure Equa1 (a, b: real; var X : real; var NbSol : integer);
Begin
    ...
end;

Procedure LireNombre (Var X : real; var ok : integer);
Var
    Chaine : string;
Begin
    repeat
        write ('entrer un reel : ');
        readln (Chaine);
        if length (Chaine) = 1 and copy (Chaine, 1, 1) = 'q' then begin
            ok := -1;
            exit;
        end else begin
        end;
        Val (Chaine, X, ok);
```

```
            if ok <> 0 then begin
                writeln ('erreur de saisie au ', ok, ' ème caractère');
                writeln ('recommencez ou tapez q pour quitter')
            end else begin
            end;
        until ok = 0;
end;

(* Programme principal *)
Begin
    repeat
        LireNombre (a, ok); if ok = -1 then exit;
        LireNombre (b, ok); if ok = -1 then exit;
        Equa1 (a, b, Solution, N);
        case N of
            -1  : writeln ('il y a une infinité de solutions');
            0   : writeln ('il n'y a pas de solution');
            1   : writeln ('la solution est : ', Solution);
        else
        end;
        write ('tapez q pour quitter ');
    until ReadKey = 'q';
end.
```

4. La récursivité

4.1. Définition

Une procédure est récursive quand le corps de la procédure contient un appel à la procédure elle-même.

Seules les procédures standards doivent être récursives.

Exemple :

la fonction factorielle peut être définie de deux manières :

n! = n * (n-1) * (n-2) * ... * 2 * 1

ou bien

n! = n * (n-1)!

0! = 1

On peut donc traduire l'algorithme de deux façons :

soit classiquement avec une boucle :

```
Fact (n) : [ Fact := 1; n > 0 ? {(i:=1 à N) Fact  <- Fact * i } | ¿ ]
```

soit sous forme récursive :

```
Fact (n) : [ n = 0 ? Fact <- 1 | Fact  <- n * Fact (n-1)  ¿ ]
```

exemple

si on veut évaluer Fact (3)

on arrive à : Fact <- 3 * Fact (2)	l'expression c'est : 3 * Fact (2)
il faut évaluer Fact (2) : 2 * Fact (1)	l'expression c'est : 3 * 2 * fact (1)
il faut évaluer Fact (1) : 1 * Fact (0)	l'expression c'est : 3 * 2 * 1 * fact (0)
il faut évaluer Fact (0) : 1	l'expression c'est : 3 * 2 * 1 * 1

À la dernière ligne, l'expression est totalement évaluée. Du coup on sort de l'appel de la procédure Fact (0), puis de l'appel de la procédure Fact (1), puis de l'appel de la procédure Fact (2), puis de l'appel de la procédure Fact (3).

Remarque : on peut toujours écrire une procédure récursive sous forme non récursive.

4.2. Récursivité croisée et Forward

On parle de récursivité croisée quand la procédure A appelle la procédure B et que la procédure B appelle la procédure A (en général, l'appel des procédures est arborescent et pas circulaire).

Ce cas pose un problème de compilation car si on déclare A avant B, le Pascal ne connaîtra pas B au moment de son appel par A et réciproquement.

Le mot-clé "forward" donne la solution.

On déclare l'en-tête de B avant A, suivi du mot-clé "Forward".

Ensuite on pourra se contenter de déclarer un en-tête réduit de B, c'est-à-dire seulement son identificateur.

Exemple Pascal :

```
:Program RecursiviteCroise
...
Procedure B (var x : ...; y : ...); Forward;
Procedure A (var x : ...; y : ...);
begin
      ...
      B (...);
      ...
end;

Procedure B;
begin
      ...
      A (...);
      ...
end;
...
Begin
      ...
```

```
end.
```

5. Les bibliothèques

5.1. Présentation

Une bibliothèque est un ensemble de procédures utilisables par un programme.

En Pascal, les bibliothèques sont appelées Unités.

Il existe des bibliothèques définies par le langage Pascal :

- System,
- Dos,
- Crt,
- etc

System est la bibliothèque standard utilisée par le Pascal sans qu'il soit nécessaire d'y faire référence. Exemple de fonctions standards :

- Random (nombre) : fournit un nombre réel entre 0 et 1.

Crt est une bibliothèque de gestion d'écran. Exemples de fonctions Crt :

- ClrScr : efface l'écran.
- Sound (integer) : émet un son en continu.
- NoSound : interrompt le son.
- HighVideo : affiche en surbrillance.
- TextBackground (numéro) : fixe la couleur du fond.
- Textcolor (numéro) : fixe la couleur du texte.

Dos est une bibliothèque de fonction Dos. Exemples :

- GetDate
- GetTime

5.2. Utilisation des unités Pascal

Pour utiliser des unités, qu'elles soient définies par le langage Pascal ou par l'utilisateur, on utilise le mot-clé "Uses" dans l'en-tête du programme principal.

Exemple Pascal :

```
Program Bibliothèque;
Uses
    Crt, Dos;
...
```

5.3. Création de ses propres unités

L'intérêt des unités c'est de pouvoir se constituer ses propres bibliothèques de procédures.

Pour cela, on crée un programme

- qui démarre par le mot-clé "Unit"

et qui est divisé en deux parties :

- une partie interface qui démarre par le mot-clé "Interface" et qui contient les en-têtes des procédures et fonctions que la bibliothèque propose;

- une partie implémentation qui démarre par le mot-clé "Implementation" et qui contient les en-têtes réduits des procédures de l'interface ainsi que leurs corps et éventuellement d'autres procédures utilisées uniquement par les procédures de l'interface.

```
Unit Equations
    Interface
        procedure Equa1 (A, B, X, NbSol);
        procedure Equa2 (A, B, C, X1, X2, NbSol);
    Implementation
        procedure Equa1;                    (* en-tête réduit *)
        begin
        ...
        end;
        procedure Equa2;                    (* en-tête réduit *)
        begin
        ...
        end;
```

Seules les procédures définies dans l'interface sont accessibles grâce à la bibliothèque.

Des procédures peuvent être définies dans l'implementation et pas dans l'interface. Elles ne seront pas utilisables par des programmes. Ces procédures servent à l'intérieur de procédures utilisées par le programme.

5.4. La compilation

Le programme source est un fichier en langage Pascal.

Le programme objet est un fichier en langage machine. C'est le résultat de la compilation.

Quand on compile les unités, il se crée un programme objet qui sera utilisé lors des compilations des programmes source appelant la bibliothèque correspondant au programme objet.

Compile/Compile permet de compiler la bibliothèque comme les programmes.

Compile/Destination permet de rediriger le résultat de la compilation dans un fichier.

Le fichier objet généré a le même nom que le fichier source mais .TPU au lieu de .PAS.

Remarque : si on redirige le résultat de la compilation d'un programme principal dans un fichier, le Pascal crée un fichier .EXE, fichier objet directement exécutable sous DOS.