# Mémo de Turbo Pascal

Lancelot Pecquet*

8 septembre 2000

**Résumé**

Ce document *ne prétend pas se substituer à un cours.* Il n'en a pas les qualités: il ne présente pas une philosophie d'une bonne programmation impérative, ne contient pas de notions algorithmiques et ne couvre même pas l'ensemble du langage Pascal ... Il constitue néanmoins un aide-mémoire, rappelant la syntaxe de Pascal au travers de tableaux récapitulatifs et d'exemples. Les notions ne sont pas introduites de façon très pédagogique et l'auteur prie ses lecteurs de l'excuser si la lecture en est rendue plus difficile.

Comme seule — et mauvaise — excuse à la piètre qualité de ce document, l'auteur tient à mentionner qu'il n'aime pas du tout le Pascal.

*Projet CODES — INRIA-Rocquencourt. B.P.105, 78153 Le Chesnay Cedex. e-mail:Lancelot.Pecquet@inria.fr

# Table des matières

# 1 Introduction

## 1.1 Pour commencer: un programme simple

Exemple de programme:

**Exemple 1**

```
PROGRAM un_programme_simple;

USES crt; {crt contient la fonction CLRSCR qui efface l'ecran}

VAR i:INTEGER; {Soit i un entier}

{Debut du programme proprement dit:}
BEGIN
  clrscr; {Efface l'ecran}
  writeln('Bonjour, entrez la valeur dont vous voulez tester la parite:');
  readln(i);  {On met la valeur lue au clavier "dans" l'entier i}
  IF ((i MOD 2) = 0) THEN
    writeln('i est pair')
  ELSE
    writeln('i est impair');
  END;

  writeln('Merci, appuyez sur Entree pour continuer...')
  readln; {Attendre l'appui de la touche Entree pour finir le programme}
END.
{Fin du programme (noter le point final).}
```

## 1.2 Mots réservés de Turbo Pascal

Les mots suivants ont un sens particulier pour Pascal:

| | | | | |
|---|---|---|---|---|
| ABSOLUTE | DOUBLE | IMPLEMENTATION | PROCEDURE | THEN |
| AND | DOWNTO | IN | PROGRAM | TO |
| ARRAY | ELSE | INLINE | REAL | TYPE |
| BEGIN | END | INTEGER | RECORD | UNIT |
| BOOLEAN | EXTENDED | LABEL | REPEAT | UNTIL |
| BYTE | EXTERNAL | LONGINT | SET | USES |
| CASE | FILE | MOD | SHL | VAR |
| CHAR | FOR | NIL | SHR | WHILE |
| COMP | FORWARD | NOT | SHORTINT | WORD |
| CONST | FUNCTION | OF | SINGLE | XOR |
| DIV | GOTO | OR | STRING | |
| DO | IF | PACKED | SUBRANGE | |

À cette liste, il faut aussi ajouter la liste des noms de fonctions — *e.g.* `CLRSCR` — qui sont définies dans les bibliothèques standard — *e.g.* `crt` —.

### 1.3 Opérations arithmétiques et booléennes, et comparaisons

**Opérateurs arithmétiques**

| Pascal | Signification Mathématique |
|---|---|
| `a + b` | $a + b$ |
| `-a` | $-a$ |
| `a - b` | $a - b$ |
| `a * b` | $ab$ |
| `a / b` | $a/b$ |
| `a DIV b` | quotient de la division euclidienne de $a$ par $b$ |
| `a MOD b` | reste de la division euclidienne de $a$ par $b$ |

**Comparateurs**

| Pascal | vrai ssi |
|---|---|
| `a = b` | $a = b$ |
| `a < b` | $a < b$ |
| `a > b` | $a > b$ |
| `a <= b` | $a \leq b$ |
| `a >= b` | $a \geq b$ |
| `a <> b` | $a \neq b$ |

**Opérateurs booléens**

| Pascal | vrai ssi |
|---|---|
| `NOT(a)` | $a$ est faux |
| `a AND b` | $a$ est vrai et $b$ est vrai |
| `a OR b` | $a$ est vrai ou $b$ est vrai |
| `a XOR b` | ($a$ `OR` $b$) `AND NOT`($a$ `AND` $b$) est vrai |

**Exemple 2** On peut combiner tout cela et avoir des expressions booléennes du genre: `(x*x + y*y <= 1) AND (((x <= 1) AND (x >= 0) AND (y <= 1) AND (y >= 0)) OR (((x >= -1) AND (x <= 0) AND (y >= -1) AND (y <= 0))))` pour décrire l'intersection du disque de rayon 1 avec l'union du premier et du troisième quadrant.

### 1.4 Les fonctions standard

**Fonctions de variable réelle**

| | |
|---|---|
| `abs(x)` | $\lvert x \rvert$ |
| `arctan(x)` | $\arctan x$ |
| `cos(x)` | $\cos x$ |
| `exp(x)` | $\exp x$ |
| `log(x)` | $\log x$ |
| `ln(x)` | $\ln x$ |
| `sin(x)` | $\sin x$ |
| `sqr(x)` | $x^2$ |
| `sqrt(x)` | $\sqrt{x}$ |
| `tan(x)` | $\tan x$ |

**Autres fonctions**

| | |
|---|---|
| `chr(n)` | le `CHAR` de code ASCII $n$ |
| `ord(c)` | le code ASCII du `CHAR` $c$ |
| `round(x)` | le plus proche entier du `REAL` $x$ |
| `trunc(x)` | la partie non-décimale du `REAL` $x$ |
| `frac(x)` | la partie décimale du `REAL` $x$ |

## 2 Instructions conditionnelles

### 2.1 `IF...THEN...`

L'instruction conditionnelle `IF` va déterminer le déroulement du programme en fonction de la valeur d'une condition booléenne.

Si l'on veut exécuter *une seule* instruction lorsque la condition booléenne $b$ est vraie, on écrit:

`IF` $b$ `THEN` Instruction à exécuter si $b$ est vraie;

Si l'on a une alternative avec *une seule* instruction à exécuter dans chaque cas, on écrit:

```
IF b THEN
   Instruction à exécuter si b est vraie {Pas de ";" ici}
ELSE
   Instruction à exécuter si b est fausse;
```

Si l'on a un bloc d'instructions à effectuer dans chacun des cas, il faut les grouper entre un `BEGIN` et `END`:

```
IF b THEN
BEGIN
   Liste d'instructions à exécuter si b est vraie;
END;
```

ou encore, avec une alternative:

```
IF b THEN
BEGIN
   Liste d'instructions à exécuter si b est vraie;
END {Pas de ";" ici}
ELSE
BEGIN
   Liste d'instructions à exécuter si b est fausse;
END;
```

## 2.2 CASE...OF...

Cette instruction permet d'exécuter une instruction en fonction du test d'une valeur *entière*[1]:

```
CASE i OF
   n1 : Instruction à faire si i = n1;
      ⋮
   np : Instruction à faire si i = np;
   ELSE Instruction à exécuter si i ∉ {n1, ... , np};
END;
```

la partie `ELSE` est optionnelle, on peut écrire aussi:

```
CASE i OF
   n1 : Instruction à faire si i = n1;
      ⋮
   np : Instruction à faire si i = np;
END;
```

1. On peut très bien se passer de `CASE`.

# 3 Boucles

Pour répéter une action plusieurs fois, on dispose de trois types de boucles. Chacune d'elles est exécutée autant de fois qu'on le souhaite. Une fois la boucle exécutée, le programme continue après la fin de celle-ci.

## 3.1 FOR...DO...

Cette boucle s'utilise lorsqu'on sait à l'avance combien de fois on va exécuter la boucle, et qu'on peut l'indexer par un entier $i$ variant dans un intervalle entier $\{a, \ldots, b\}$.

Si l'on veut que l'indice croisse de $a$ à $b$, on écrit:

```
FOR i:= a TO b DO
BEGIN
    Liste des instructions à faire au i-ième passage dans la boucle;
END;
```

si l'on veut qu'il décroisse de $b$ à $a$:

```
FOR i:= b DOWNTO a DO
BEGIN
    Liste des instructions à faire au i-ième passage dans la boucle;
END;
```

Comme pour `IF`, les marques de début — `BEGIN` — et de fin de bloc — `END` — sont optionnelles dans le cas d'*une seule* instruction.

Si l'on doit imbriquer plusieurs `FOR`, les `BEGIN...END` peuvent ne se mettre que dans la boucle la plus intérieure, comme par exemple dans:

**Exemple 3**

```
...
VAR i,j,n:INTEGER;
...
  FOR i := 1 TO n DO
    FOR j := 1 to n DO
    BEGIN
      Liste des instructions à effectuer lors du $(i,j)$-ième passage;
    END;
...
```

## 3.2 WHILE...DO...

Cette instruction s'utilise pour boucler tant qu'une condition booléenne $b$ est vraie:

```
WHILE b DO
BEGIN
    Liste des instructions à faire tant que b est vraie;
END;
```

Comme pour `FOR`, les marques de début — `BEGIN` — et de fin de bloc — `END` — sont optionnelles dans le cas d'*une seule* instruction.

### 3.3 REPEAT...UNTIL...

Cette instruction s'utilise pour boucler jusqu'à ce qu'une condition booléenne $b$ soit vraie[2]:

```
REPEAT
    Liste des instructions à faire;
UNTIL b;
```

# 4 Types Pascal

Notez qu'ici, on n'a pas besoin de BEGIN et de END pour marquer un bloc puisque les instructions sont confinées entre REPEAT et UNTIL.

## 4.1 Types simples

| Pascal | Signification | Exemples |
|---|---|---|
| BYTE | octet (entre 0 et 255) | x := 255; |
| INTEGER | entier machine (entre -32768 et 32767) | x := 15038;, x := -576 |
| REAL | flottant (entre $1 \cdot 10^{-38}$ et $1 \cdot 10^{38}$) | x := 167.3487;, x := -1.6887E12; |
| BOOLEAN | booléen | x := TRUE;, x := FALSE; |
| CHAR | caractère | x := 'u';, x := '6'; |

## 4.2 Types composés

### 4.2.1 Les ARRAYs

Un ARRAY est un tableau, unidimensionnel de taille fixée contenant des valeurs, toutes de même type. Le $i$-ième élément d'un tableau A est notée A[i].

**Exemple 4**

```
PROGRAM tablo;
USES crt; {crt contient la fonction CLRSCR qui efface l'ecran}
CONST N = 10; {Taille du tableau}
      P = 50; {On va tirer au hasard des nombres entre 0 et P}

TYPE tableau = ARRAY[1..N] OF INTEGER; {Declaration du tableau}

VAR A:tableau;
    i:INTEGER;

{Debut du programme principal:}
BEGIN
  clrscr; {Efface l'ecran}
  randomize; {Initialise le generateur pseudo-aleatoire}

  {Remplissons le tableau avec nombre au hasard:}
  FOR i := 1 TO N DO A[i] := random(P); {Entier aleatoire entre 0 et P}
```

2. C'est un peu le "dual" de WHILE.

```
  {Affichons le tableau:}
  write('['); {On ouvre un crochet}
  FOR i := 1 TO N - 1 DO write(A[i],', '); {N est > 1 donc N - 1 >= 1}
  writeln(A[N],']'); {Dernier element, suivi d'un crochet fermant}

  writeln('Merci, appuyez sur Entree pour continuer...')
  readln; {Attendre l'appui de la touche Entree pour finir le programme}
END.
{Fin du programme principal.}
```

Pour créer des tableaux multidimensionnels, on crée des tableaux de tableaux:

**Exemple 5**

```
PROGRAM matrices;
USES crt; {crt contient la fonction CLRSCR qui efface l'ecran}
CONST N = 10; {Taille des matrices}
      P = 50; {On va tirer au hasard des nombres entre 0 et P}

TYPE matrix = ARRAY[1..N] OF ARRAY[1..N] OF INTEGER; {Type matrice}

VAR A:matrix;
    i,j:INTEGER;

{Debut du programme principal:}
BEGIN
  clrscr; {Efface l'ecran}
  randomize; {Initialise le generateur pseudo-aleatoire}

  {Remplissons le tableau avec nombre au hasard:}
  FOR i := 1 TO N DO
    FOR j := 1 TO N DO
      A[i][j] := random(P); {Entier aleatoire entre 0 et P}

  {Affichons la matrice:}
  FOR i := 1 TO N DO
  BEGIN
    FOR j := 1 TO N DO writeln(A[i][j],'\t'); {\t est une tabulation}
    writeln('');                              {On revient a la ligne}
  END;

  writeln('Merci, appuyez sur Entree pour continuer...')
  readln; {Attendre l'appui de la touche Entree pour finir le programme}
END.
{Fin du programme principal.}
```

### 4.2.2 Les STRINGs

Un objet STRING est une chaîne de caractères de longueur maximale fixée (En fait c'est un ARRAY dont les éléments sont toujours de type CHAR.). La $i$-ième lettre de la chaîne S est notée S[i]. Attention, la longueur maximale que peut avoir une chaîne de caractères est $N = 255$ en Pascal.

Cela est dû au fait que Pascal stocke la longueur des chaînes dans un octet qui est un nombre de 8 bits. Il varie par conséquent de 0 à $255=2^8-1$.

**Exemple 6**

```
PROGRAM chaines;
USES crt; {crt contient la fonction CLRSCR qui efface l'ecran}
CONST N = 40; {Longueur maximale des chaines a manipuler}
VAR x,y,z:STRING[N];
    c:CHAR;
    l:INTEGER;

BEGIN
  clrscr; {Efface l'ecran}
  x := 'Hello';
  y := ', enjoy Pascal!';
  z := x + y; {z contient la concatenation de x et y: 'Hello, enjoy Pascal!'}
  c := z[2];  {c contient le caractere 'e'}
  l := length(z); {Longueur de la chaine z}

  writeln('La variable z contient la chaine:');
  writeln(z);
  writeln('Sa deuxième lettre est',c,'et sa longueur est',l);

  writeln('Merci, appuyez sur Entree pour continuer...')
  readln; {Attendre l'appui de la touche Entree pour finir le programme}
END.
```

On dispose également des instructions suivantes:

- `l := length(x)` $l$ est la longueur du mot $x$;
- `delete(x,i,n)` efface $n$ caractères à partir du $i$-ième dans la chaîne $x$;
- `i := pos(m,x)` $i$ est la position dans $x$ à laquelle la chaîne $m$ apparaît dans la chaîne $x$ pour la première fois ou 0 si elle n'apparaît pas;
- `y := copy(x,i,n)` $y$ contient la sous-chaîne de $x$ qui commence à l'indice $i$ et contient $n$ caractères.
- `insert(m,x,i)` insère la chaîne $m$ dans la chaîne $x$ à la place $i$;
- `str(n,x)` met dans la chaîne $x$ la suite de chiffres décimaux constituant le nombre $n$;
- `val(x,n,i)` convertit la chaîne $x$ — qui ne doit comporter que des chiffres — en le nombre $n$. Au cas où l'une des lettres n'est pas un chiffre, la variable $i$ contient la position du premier caractère non-numérique, sinon $i$ vaut 0.

### 4.2.3 Les `RECORD`s

Un `RECORD` est une fiche contenant une liste de rubriques, appelées *champs*. Chaque champ a un nom et un type. D'un champ à l'autre, les types peuvent varier. Le contenu du champ $x$ d'une fiche

$F$ est noté `F.x` . Lorsqu'on manipule la même fiche $F$ pour travailler sur plusieurs champs, on peut considérer que le nom de la fiche est implicite en utilisant le mot-clé `WITH`:

```
WITH F DO
BEGIN
   Liste des instructions à faire avec les champs de la fiche F;
END;
```

**Exemple 7**

```
PROGRAM fiches;

CONST N = 255; {Longueur maximale des chaines a manipuler}

{Definissons un nouveau type de fiche:}
TYPE
  personne = RECORD
    nom,prenom,adresse:STRING[N];
    sexe:CHAR;
    age:INTEGER;
  END;

VAR vous:personne;

{Debut du programme principal:}
BEGIN
  clrscr; {Efface l'ecran}

  {Construction d'une fiche:}
  writeln('Entrez votre nom SVP:');
  readln(vous.nom);
  writeln('Entrez votre prenom:');
  readln(vous.prenom);
  writeln('Entrez votre adresse:');
  readln(vous.adresse);
  writeln('Etes-vous de sexe masculin ou feminin (M/F):');
  readln(vous.sexe);
  writeln('Entrez votre age:');
  readln(vous.age);

  {Impression d'une fiche:}
  IF (vous.sexe = 'M') THEN writeln('Monsieur,') ELSE writeln('Madame,');
  WITH vous DO
  BEGIN
    writeln('Vous vous appelez',prenom,' ',nom);
    writeln('Vous habitez ',adresse);
    writeln('et vous avez ',age,' ans.');
  END;

  writeln('Merci, appuyez sur Entree pour continuer...')
  readln; {Attendre l'appui de la touche Entree pour finir le programme}
END.
{Fin du programme principal.}
```

# 5 Fonctions et procédures

## 5.1 Les PROCEDUREs

Une PROCEDURE est une partie de programme, réutilisable, qu'on invoque pour effectuer une action. Par exemple, pour effacer l'écran, on utilise la procédure CLRSCR; Cette procédure n'est pas définie dans notre programme, mais dans la bibliothèque appelée crt, qu'on incorpore si on le souhaite en plaçant en début de programme la commande USES crt; .

Dans un programme, nous pouvons également définir de nouvelles procédures. Leur descriptif est fait dans le préambule du programme, après la déclaration des variables avec VAR et avant le BEGIN du programme principal. Leur utilisation se fait dans le programme principal par invocation du nom de la procédure.

La syntaxe générale pour la déclaration d'une procédure $p$ est:

PROCEDURE $p(x_1 : t_1 \; ; \; \dots \; ; \; x_n : t_n)$;
VAR *déclaration de variables locales*;
BEGIN
    Liste des instructions à effectuer;
END;

L'action de la procédure est parametrée par les valeurs de ses arguments $x_i$, de type respectif $t_i$. Si $a, b, c$ sont du type $t_i$ on peut écrire: $\dots ; a, b, c : t_i ; \dots$. Il peut ne pas y avoir d'argument (comme avec CLRSCR).

On a la possibilité d'utiliser des variables dites *locales*, dont l'existence est limitée à l'espace entre BEGIN et END pour stocker des résultats intermédiaires,*etc* ... Ces variables locales sont déclarées comme pour un programme avec le mot-clé VAR[3].

Dans le cas précédent, il n'est pas possible de modifier les valeurs des arguments de la procédure dans le corps de celle-ci. Si l'on souhaite pouvoir modifier la variable $x_i$ qui est du type $t_i$, alors il faut remplacer "$x_i : t_i$" par "VAR $x_i : t_i$" dans la déclaration de la procédure.

**Exemple 8** Comme exemple, nous allons écrire une procédure qui écrit un tableau $T$ de taille $l$. Nous reprenons le tableau de l'exemple 4.

```
PROGRAM affichage;
USES crt; {crt contient la fonction CLRSCR qui efface l'ecran}
CONST N = 10; {Taille du tableau}
      P = 50; {On va tirer au hasard des nombres entre 0 et P}

TYPE tableau = ARRAY[1..N] OF INTEGER; {Declaration du tableau}

VAR A:tableau;
    i:INTEGER;

{Procedure triant un tableau T de taille l:}
PROCEDURE affiche(T:tableau ; l:INTEGER);
VAR j:INTEGER;
BEGIN
```

3. Il est impératif, pour la bonne compréhension du programme, que les variables dont il est question dans la définition d'une procédure soient limitées aux arguments et aux variables locales. *Il ne faut pas manipuler de variables globales dans une procédure.*

```
15    IF (l = 1) THEN
16      write('[',T[1],']') {Pas de ";" avant ELSE}
17    ELSE
18    BEGIN
19      write('['); {Debut du tableau}
20      FOR j := 1 TO l-1 DO write(T[j],',');
21      writeln(T[l],']');
22    END;
23  END;
24  {Fin de la procedure affiche.}
25
26
27  {Debut du programme principal:}
28  BEGIN
29    clrscr; {Efface l'ecran}
30    randomize; {Initialise le generateur pseudo-aleatoire}
31    FOR i := 1 TO N DO A[i] := random(P); {Entier aleatoire entre 0 et P}
32
33    affiche(A,N); {Affiche le tableau A}
34  END.
35  {Fin du programme principal.}
```

À la ligne 30, on affiche le tableau $A$, de longueur $N$. La procédure `affiche` est définie pour un tableau quelconque $T$, de longueur $l$ entre les lignes 10 et 23.

### 5.2 Les `FUNCTION`s

Une `FUNCTION` est une partie de programme, réutilisable, qu'on invoque pour avoir une valeur.

Par exemple, pour avoir la racine carrée d'un réel $x$, on utilise la fonction `sqrt(x);` (pour *square root*)

Cette fonction n'est pas définie dans notre programme mais fait partie des fonctions standard à disposition du programmeur Pascal.

Dans un programme, nous pouvons également définir de nouvelles fonctions. Leur descriptif est fait dans le préambule du programme, après la déclaration des variables avec `VAR` et avant le `BEGIN` du programme principal. Leur utilisation se fait dans le programme principal par invocation du nom de la fonction. La syntaxe générale pour la déclaration d'une fonction $f$ est:

`FUNCTION` $f(x_1 : t_1\ ;\ \dots\ ;\ x_n : t_n)$:t
`VAR` *déclaration de variables locales*
`BEGIN`
    Liste des instructions à effectuer;
    $f :=$ *Valeur de la fonction*;
`END`;

La fonction prend pour arguments des variables $x_i$ de type respectif $t_i$ et renvoie une valeur de type $t$.

Si $a, b, c$ sont du type $t_i$ on peut écrire: $\dots; a, b, c : t_i; \dots$. Il peut ne pas y avoir d'argument.

On a la possibilité d'utiliser des variables dites *locales*, dont l'existence est limitée à l'espace entre `BEGIN` et `END` pour stocker des résultats intermédiaires,*etc* ... Ces variables locales sont déclarées comme pour un programme avec le mot-clé `VAR`[4].

Par principe, une fonction *ne modifie pas ses arguments.*

**Exemple 9** Comme exemple, nous allons écrire une fonction qui calcule le plus grand de deux entiers $x$ et $y$.

```
1  PROGRAM max;
2  CONST P = 50;
3  USES crt; {crt contient la fonction CLRSCR qui efface l'ecran}
4  VAR a,b,c:INTEGER;
5
6  {Fonction calculant le plusgrand de a et b:}
7  FUNCTION plusgrand(x,y:INTEGER):INTEGER;
8  VAR z:INTEGER;
9  BEGIN
10   IF (x > y) THEN z:= x ELSE z := y;
11   plusgrand := z;
12 END;
13 {Fin de la fonction plusgrand.}
14
15
16 {Debut du programme principal:}
17 BEGIN
18   clrscr; {Efface l'ecran}
19   randomize; {Initialise le generateur pseudo-aleatoire}
20   a := random(P); {Entier aleatoire entre 0 et P}
21   b := random(P); {Entier aleatoire entre 0 et P}
22   c := plusgrand(a,b); {Le plus grand entre a et b}
23   writeln('Le plus grand entre',a,' et ',b,' est ',c);
24
25   writeln('Merci, appuyez sur Entree pour continuer...')
26   readln; {Attendre l'appui de la touche Entree pour finir le programme}
27 END.
28 {Fin du programme principal.}
```

À la ligne 20, on calcule le plus grand nombre entre $a$ et $b$. La fonction `plusgrand` est définie pour deux entiers quelconque $x$ et $y$, entre les lignes 6 et 12.

# 6 Entrées/Sorties

## 6.1 Écran et clavier

Essentiellement on utilisera:

- `clrscr;` efface l'écran.
- `gotoxy(c,l);` place le curseur à la colonne $c$ et à la ligne $l$. Le prochain `write` commencera à cet endroit;

4. Il est impératif, pour la bonne compréhension du programme que les variables dont il est question dans la définition d'une fonction soient limitées aux arguments et aux variables locales. *Il ne faut pas manipuler de variables globales dans une fonction.*

- **write**($x_1, \dots, x_n$); Écrit les variables $x_1 \dots, x_n$ les unes à la suite des autres. Elles peuvent être de tout type simple.
- **writeln**($x_1, \dots, x_n$); Écrit les variables $x_1 \dots, x_n$ les unes à la suite des autres — elles peuvent être de tout type simple — et ajoute un retour à la ligne.
- **readln**($x$); Lit ce que l'utilisateur tape avant d'appuyer sur la touche Entrée et le met dans la variable $x$ (qui doit être du bon type!)

Certaines fonctions comme **clrscr** sont définies dans une bibliothèque standard de gestion de l'écran et du clavier appelée **crt**. On l'utilisera avec la commande "**USES crt;**" en début de programme.

**Exemple 10**

```
PROGRAM E_S;

USES crt;

VAR x:INTEGER;
BEGIN
  clrscr; {Efface l'écran}
  readln(x); {Lit la valeur x au clavier}
  writeln('La valeur de x est',x);
END.
```

## 6.2 Les fichiers

On stocke des données (résultats de calculs, fichiers clients, répertoires téléphoniques, *etc* ... ) dans des *fichiers* (FILEs), sur un disque dur, une disquette, *etc* ... afin de les réutiliser.

Il y a deux types de fichiers: les fichiers de données et les fichiers de texte. Les fichiers de données contiennent des fiches, toutes de même type. Typiquement, on aura une fiche dont le format est un RECORD[5] particulier défini par le créateur du fichier. Dans les deux cas, une tête de lecture/écriture se déplace dans le fichier. À l'ouverture du fichier, sa position est 0.

### 6.2.1 Les fichiers de données

Voici comment on utilise un fichier:

**Exemple 11**

```
PROGRAM fichier_donnees;
USES crt; {crt contient la fonction CLRSCR qui efface l'ecran}
CONST N = 255; {Longeur max des chaines de caracteres}

TYPE
type_tel = STRING[10]; {Il n'y a que 10 chiffres pour un telephone}

personne = RECORD
```

5. Cela peut être un type simple ou composé mais tous les éléments du fichier seront du même type.

```
  nom,prenom:STRING[N];
  tel:type_tel;
END;

VAR p:personne;
    annuaire:FILE OF personne;
    taille,i:INTEGER;

{Début du programme principal.}
BEGIN
  clrscr; {Efface l'ecran}
  assign(annuaire,'C:\ANNU.DAT'); {Association avec le support physique}
  {$I-} {Relache le mecanisme de sortie du programme en cas de probleme d'E/S}
  reset(annuaire); {Tente d'ouvrir le fichier comme s'il existait deja}
  {$I+} {Enclenche le mecanisme de sortie du programme en cas de probleme d'E/S}
  IF IOResult <> 0 THEN  {Si une erreur d'E/S s'est produite...}
    rewrite(annuaire); {Ouvrir le fichier comme un nouveau fichier (il n'existait pas)}
  taille := filesize(annuaire); {Taille du fichier}
  seek(annuaire,taille);        {Placons nous a la fin du fichier}

  {Ici on va ajouter la premiere fiche:}
  WITH p DO
  BEGIN
    nom := 'Bidochon';
    prenom := 'Robert'
    tel := '0187220100';
  END;

  {Puis ecrire la fiche p dans le fichier ``annuaire'':}
  write(annuaire,p);

  {Faisons de même avec Madame:}
  WITH p DO
  BEGIN
    nom := 'Bidochon';
    prenom := 'Raymonde'
    tel := '0187220100';
  END;

  {Puis ecrire a la suite la fiche p dans le fichier ``annuaire'':}
  write(annuaire,p);

  {Et le chien:}
  WITH p DO
  BEGIN
    nom := 'Kador';
    prenom := 'Le chien'
    tel := '0187220101'; {Il a sa ligne privee :) }
  END;

  {Remplacons Robert (1er de la liste) par Kador:}
  seek(annuaire,0); {On se met en position 0}
  write(annuaire,p);  {Et on ecrit la fiche p}
```

```
62
63   {Maintenant pour resumer, on ecrit le contenu sur l'ecran:}
64   writeln('Le fichier C:\ANNU.DAT contient ',filesize(annuaire),' fiches,');
65   writeln('c''''est a dire',filesize(annuaire)*sizeof(personne),' octets:');
66   seek(annuaire,0); {On rembobine}
67   FOR i := 1 TO filesize(annuaire) DO
68   BEGIN
69     writeln('La fiche numero ',i,' est:');
70     read(annuaire,p); {Cela met dans p la fiche courante}
71     WITH p DO
72     BEGIN
73       writeln(prenom + ' ' + nom + ':' + tel);
74     END;
75   END;
76
77   close(annuaire); {Ferme le fichier}
78   writeln('Merci, appuyez sur Entree pour continuer...')
79   readln; {Attendre l'appui de la touche Entree pour finir le programme}
80 END.
81 {Fin du programme principal.}
```

Notez qu'à la ligne 23, on associe avec `assign` la variable `annuaire` avec un objet physique sur le disque dur qui s'appellera `C:\ANNU.DAT`.

Un point important est ce qui se passe entre les lignes 21 et 25: l'instruction `reset` ouvre un fichier *existant*. Si ce fichier n'existe pas, il résulte une erreur, ce qui, normalement, fait s'arrêter le programme. Or ici, en faisant précéder cette commande de la directive[6] `{$I-}`, on empêche volontairement que le programme finisse tout de suite: une variable — toujours connue de Pascal et appelée `IOResult` — est positionnée lors de l'exécution du programme à une valeur non-nulle si une erreur d'entrée/sortie (E/S[7]) s'est produite. On interprète la valeur de `IOResult` dans la suite du programme (ligne 24). Si elle est non-nulle, c'est qu'une erreur d'E/S s'est produite. Or, comme à la ligne 23 on a rétabli le mécanisme d'arrêt du programme en cas d'erreur, cette dernière s'est forcément produite lors du `reset`: le fichier n'existait en fait pas. Le cas échéant, on réouvre le fichier, mais cette fois-ci avec la commande pour les fichiers qui n'existent pas (`rewrite`).

Attention: Si on faisait un `rewrite` directement sur un fichier existant, on perdrait toutes les données qu'il contient!

### 6.2.2 Les fichiers de texte

Un fichier texte est un type particulier: `TEXT`. Sa manipulation est plus simple — on s'en sert comme de l'écran, avec des `readln` et des `writeln` — mais offre moins de possibilités: on ne peut accéder aux données que séquentiellement, c'est à dire que pour accéder à la ligne $n$, il faut faire une boucle de $n$ `readln`. En d'autres termes, pas d'instruction `seek`. Les données sont également stockées d'une façon moins efficace.

Le plus simple est de donner un exemple:

**Exemple 12**

```
PROGRAM fichier_texte;
```

6. Cette syntaxe, utilisant des accolades — caractères réservés aux commentaires — et ce mécanisme d'ouverture de fichiers est tout simplement exécrable!

7. I/O en anglais pour *Input/Output*.

```
USES crt; {crt contient la fonction CLRSCR qui efface l'ecran}
CONST N = 255; {Longeur max des chaines de caracteres}

VAR gloubi:TEXT;
    chaine:STRING[N];

{Debut du programme principal:}
BEGIN
  clrscr; {Efface l'écran}
  assign(gloubi,'C:\GLOUBI.TXT'); {Association avec le support physique}
  {$I-} {Relache le mecanisme de sortie du programme en cas de probleme d'E/S}
  reset(gloubi); {Tente d'ouvrir le fichier comme s'il existait deja}
  {$I+} {Enclenche le mecanisme de sortie du programme en cas de probleme d'E/S}
  IF IOResult <> 0 THEN  {Si une erreur d'E/S s'est produite...}
    rewrite(gloubi); {Ouvrir le fichier comme un nouveau fichier (il n'existait pas)}

  writeln(gloubi,'Confiture de fraises');
  writeln(gloubi,'Chocolat rape');
  writeln(gloubi,'Saucisses de Francfort');
  writeln(gloubi,'Bananes bien mures');
  writeln(gloubi,'Moutarde de Dijon');
  writeln(gloubi,'Anchois');
  writeln(gloubi,'Creme Chantilly');

  {On ecrit la liste:}
  reset(gloubi); {Rembobine le fichier}
  writeln('Ingredients du Gloubiboulga:');
  WHILE NOT(EOF(gloubi)) DO {Tant que la fin du fichier n'est pas atteinte}
  BEGIN
    readln(gloubi,chaine);
    writeln('- ',chaine);
  END;

  close(gloubi); {Ferme le fichier}
  writeln('Merci, appuyez sur Entree pour continuer...')
  readln; {Attendre l'appui de la touche Entree pour finir le programme}
END.
{Fin du programme principal.}
```

Pour résumer, on a les instructions suivantes (D="Données", T="Texte", TL = "tête de lecture", TL++ = "déplacement de la TL d'un cran"):

| Pascal | Type | Description |
|---|---|---|
| `assign`($f$,$F$) | D, T | Associe la variable $f$ au fichier “réel” dont le nom MS-DOS est $F$ |
| `EOF`($f$) | D, T | Renvoie `true` si la TL est en fin de fichier, `false` sinon |
| `filesize`($f$) | D | Renvoie le nombre de fiches du fichier (mis à jour en temps réel) |
| `read`($f, x$) | D, T | Lit la fiche $x$ à la position courante de la TL dans $f$, TL++ |
| `readln`($f, x$) | T | Lit la chaîne $x$ dans le fichier $f$ et passe à la ligne suivante |
| `reset`($f$) | D, T | (Ré-)Ouvre le fichier *existant* $f$ et place la TL en position 0 |
| `rewrite`($f$) | D, T | Crée le fichier $f$ et place la TL en position 0 |
| `seek`($f, i$) | D | Met la TL en position $i$ dans le fichier $f$ |
| `write`($f, x$) | D, T | Écrit la fiche $x$ à la position courante de la TL dans $f$, TL++ |
| `writeln`($f, x$) | T | Écrit la chaîne $x$ dans le fichier $f$ et passe à la ligne suivante |

L'instruction `sizeof`($t$) donne le nombre d'octets que prend en mémoire un objet de type $t$.

# 7 Un exemple complet

**Exemple 13** Comme exemple, nous allons écrire une procédure qui trie le tableau d'entiers défini dans l'exemple 4 dans l'ordre croissant, avec l'algorithme dit de “tri à bulles”. Comme nous serons très fiers des résultats, nous les écrirons dans un fichier de données à chaque exécution du programme.

```
PROGRAM tri_bulles;
USES crt; {Importe CLRSCR}
CONST N = 10; {Taille du tableau}
      P = 50; {On va tirer au hasard des nombres entre 0 et P}
TYPE tableau = ARRAY[1..N] OF INTEGER; {Declaration du tableau}
VAR A:tableau;
    i:INTEGER;
    resultats := FILE OF tableau;

{Fonction transformant un tableau de taille l en chaine affichable:}
FUNCTION string_tableau(T:tableau ; l:INTEGER);
VAR j,r:INTEGER;
    s,v:STRING[N];
BEGIN
  IF (l = 1) THEN
    val(v,T[1],r);  {Copie la valeur T[1] dans la chaine v, r=0 si tout est OK}
    s := '[' + v + ']' ; {Pas de ";" avant ELSE}
  ELSE
  BEGIN
    s[1] := '['; {Debut du tableau}
    FOR j := 1 TO l-1 DO
    BEGIN
      val(v,T[j],r);  {Copie la valeur T[j] dans la chaine v, r=0 si tout est OK}
      s := s + ',' + v;
    END;
    val(v,T[l],r);  {Copie la valeur T[l] dans la chaine v, r=0 si tout est OK}
    s := s + ',' + v + ']';
  END;
  affiche := s;
END;
{Fin de la fonction string_tableau.}
```

```
{Procedure triant sur place un tableau T de taille l}
PROCEDURE tri(VAR T:tableau ; l:INTEGER)
VAR i,j,u:INTEGER; {Variables locales}
BEGIN
  IF (l>1) THEN {Si T a plus d'un seul element:}
  BEGIN
    {Debut de la double boucle}
    FOR i := l DOWNTO 1 DO {Pour i descendant de l a 1:}
      FOR j := 2 TO i DO
      BEGIN
        IF (T[j-1] > T[j]) THEN {Si le couple n'est pas dans l'ordre:}
        BEGIN
          u := T[j-1];    {Stocker temporairement la valeur en j-1 dans u...}
          T[j-1] := T[j]; {...puis on inverse...}
          T[j] := u;      {...les positions i et j}
        END;
      END;
    {Double boucle finie}
  END;
END;
{Fin de la procedure tri.}


{Debut du programme principal:}
BEGIN
  randomize; {Initialise le generateur pseudo-aleatoire}
  clrscr; {Efface l'ecran}
  assign(resultats,'C:\RES_TRI.DAT');

  {Remplissons le tableau avec nombre au hasard:}
  FOR i := 1 TO N DO A[i] := random(P); {Entier aleatoire entre 0 et P}

  {Affichage et tri du tableau:}
  write('A a ete initialise a:');
  writeln(string_tableau(A,N)); {Affiche A avant le tri}
  tri(A,N); {Tri du tableau A}
  write('une fois trie, A vaut:');
  writeln(string_tableau(A,N)); {Affiche A après le tri}

  {$I-} {Relache le mecanisme de sortie du programme en cas d'erreur E/S}
  reset(resultats); {Tente d'ouvrir le fichier comme s'il existait deja}
  {$I+} {Enclenche le mecanisme d'arret de programme en cas d'erreur E/S}
  IF IOResult <> 0 THEN {si une erreur E/S s'est produite...}
    rewrite(resultats);  {ouvrir le fichier comme un nouveau fichier}
  taille := filesize(resultats);
  seek(resultats,taille); {Placement a la fin du fichier}
  write(resultats,A);     {Ecriture du tableau A}
  close(resultats);       {Fermeture du fichier}

  writeln('Merci, appuyez sur Entree pour continuer...')
  readln; {Attendre qu'on appuie sur Entree avant de sortir}
```

```
END.
{Fin du programme principal.}
```