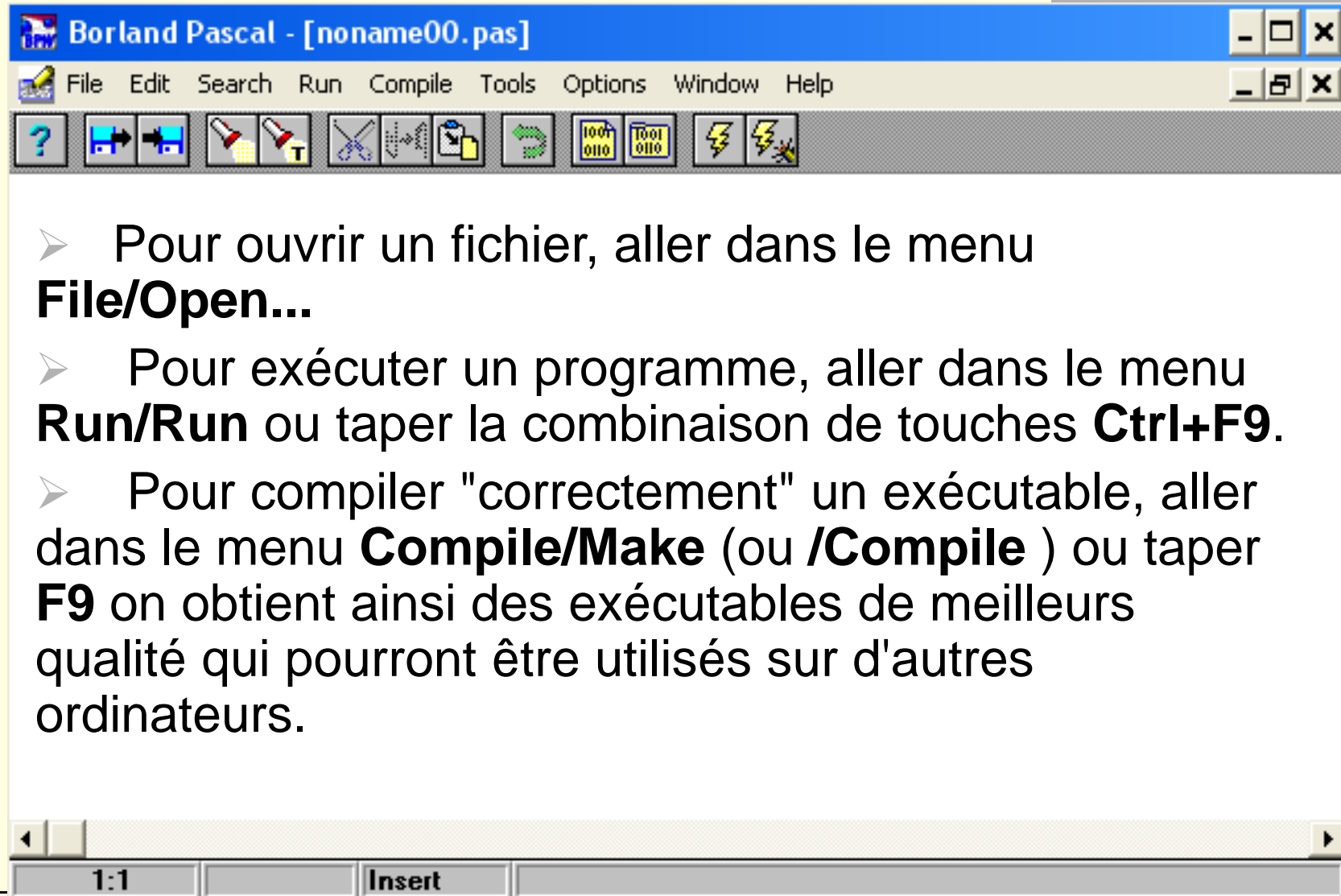


Cours du langage Pascal

Environnement



- Pour ouvrir un fichier, aller dans le menu **File/Open...**
- Pour exécuter un programme, aller dans le menu **Run/Run** ou taper la combinaison de touches **Ctrl+F9**.
- Pour compiler "correctement" un exécutable, aller dans le menu **Compile/Make** (ou **/Compile**) ou taper **F9** on obtient ainsi des exécutables de meilleurs qualité qui pourront être utilisés sur d'autres ordinateurs.

Architecture standard en pascal

{ les instructions facultatives pour compilation doivent être entre accolades }

Program *nom de programme ;*

Uses *unités utilisées ;*

Const *déclaration de constantes ;*

Type *déclaration de types ;*

Function *déclaration de fonction ;*

Procedure *déclaration de procédure paramétrée ;*

Var *déclaration de variables ;*

Procedure *déclaration de procédure simple ;*

BEGIN *{ programme principal }*

...

Commandes

...

END.

Grammaire du Pascal

- Un nom de programme respecte les règles liées aux identificateurs et **ne peut pas contenir le caractère point "."**
- Un programme principal débute toujours par **BEGIN et se termine par END.** (avec un point) .
- Un sous-programme (ou fonction, procédure, bloc conditionnel...) commence lui aussi par **BEGIN mais se termine par END ;** (un point-virgule).

Grammaire du Pascal

- Chaque **commande doit se terminer avec un point-virgule**. Il n'y a pas d'exception à la règle hormis **BEGIN** et **l'instruction précédent END ou ELSE**.
- Il est toléré de mettre plusieurs instructions les unes à la suite des autres sur une même ligne du fichier mais il est recommandé de n'en écrire qu'une par ligne.

Identificateurs

Les noms de constantes, variables, procédures, fonctions, tableaux, etc. (appelés **identificateurs**) doivent être des noms simples.

Les identificateurs doivent impérativement être différents de ceux d'unité utilisées, de mots réservés du langage Pascal:

AND, ARRAY, ASM, BEGIN, CASE, CONST,
CONSTRUCTOR, DESTRUCTOR, DIV, DO, DOWNT0,
ELSE, END, EXPORTS, FILE, FOR, FUNCTION, GOTO, IF,
IMPLEMENTATION, IN, INHERITED, INLINE, INTERFACE,
LABEL, LIBRARY, MOD, NIL, NOT, OBJECT, OF, OR,
PACKED, PROCEDURE, PROGRAM, RECORD, REPEAT,
SET, SHL, SHR, STRING, THEN, TO, TYPE, UNIT, UNTIL,
USES, VAR, WHILE, WITH, XOR.

Identificateurs

- ❑ Ne doivent pas excéder 127 caractères (1 lettre au minimum).
- ❑ Doivent exclusivement être composés des 26 lettres de l'alphabet, des 10 chiffres et du caractère de soulignement.
- ❑ Ne doivent pas contenir de caractères accentués, ni d'espace, ni de point et ni les caractères suivants : @, \$, &, #, +, -, *, /.
- ❑ Les chiffres sont acceptés hormis en première place.

Entrées et sorties à l'écran

La commande **WRITE** permet d'afficher du texte et de laisser le curseur à la fin du texte affiché.

- ❑ Cette commande permet d'afficher des chaînes de caractères (< 255) ainsi que des valeurs de variables, de constantes, de types...
- ❑ Le texte doit être entre apostrophe.
- ❑ Si le texte à afficher contient une apostrophe, il faut alors la doubler.
- ❑ Les différents noms de variables doivent être séparés par des virgules.

Syntaxe :

Write ('*Texte à afficher*', *variable1*, *variable2*, '*texte2*') ;

Write ('*L"apostrophe se double."*') ;

Entrées et sorties à l'écran

La commande **WRITELN** est semblable à la précédente à la différence près que le curseur est maintenant renvoyé à la ligne suivante.

Syntaxe :

WriteLn ('*Texte avec renvoi à la ligne*') ;

La commande **READ** permet à l'utilisateur de rentrer une valeur qui sera utilisée par le programme. Cette commande ne provoque pas de retour Chariot, c'est-à-dire que le curseur ne passe pas à la ligne.

Syntaxe :

Read (*variable*) ;

Entrées et sorties à l'écran

La commande **READLN** permet à l'utilisateur de rentrer une valeur qui sera utilisée par le programme. **Cette commande provoque le retour Chariot**, c'est-à-dire que le curseur passe à la ligne suivante.

Lorsque aucune variable n'est affectée à la commande, il suffit de presser sur <ENTREE>.

Syntaxe :

```
ReadLn (variable1, variable2) ;  
ReadLn ;
```

Opérateurs

Opérateurs mathématiques

- Addition (et union) $+$
- Soustraction (et complément) $-$
- Division $/$
- Multiplication (et intersection) $*$
- Égalité $=$
- MOD : renvoie le reste de la division $x \text{ MOD } y$
- DIV : renvoie le quotient de la division $x \text{ DIV } y$

Opérateurs prioritaires : $$, $/$, DIV et MOD.*

Opérateurs secondaires : $+$ et $-$.

Vous pouvez utiliser des parenthèses.

Opérateurs

Opérateurs relationnels

- Inférieur strict <
- Inférieur ou égale (et inclus) <=
- Supérieur strict >
- Supérieur ou égale (et contenant) >=
- Différent <>

Opérateur ultra prioritaire : NOT.

Opérateur semi prioritaire : AND.

Opérateur non prioritaires : OR et XOR.

Opérateurs

Opérateurs logiques :

- AND : le "et" logique des maths
- OR : le "ou"
- XOR : le "ou" exclusif
- NOT : le "non«

Priorité des opérateurs

- Niveau 1 : NOT.
- Niveau 2 : *, /, MOD, DIV, AND.
- Niveau 3 : +, -, OR, XOR.
- Niveau 4 : =, <, >, <=, >=, <>.

Variables - Déclaration

Déclaration

Toutes les variables doivent être préalablement déclarées avant d'être utilisées dans le programme, c'est-à-dire qu'on leur affecte un type .

On peut les déclarer de divers manières :

- ❑ Au tout début du programme avec la syntaxe

VAR nom de la variable : type ;

Dans ce cas, elles seront alors valables pour le programme dans son intégralité (sous-programmes, fonctions, procédures...).

- ❑ Au début d'une procédure avec la syntaxe précédente.

Elles ne seront valables que dans la procédure.

Variables - Les types

Le *type* de la variable est choisi en fonction du *type* de données qu'elle va recevoir.

TYPES DE VARIABLES				
Type	Déscription	Fourchette	Exemples	Mémoire requise
Shortint	Entiers courts	-128 à 127	-125; 0; 32	1 octet
Integer	Entiers "normaux"	-32 768 à 32 767	-30 000; 421;	2 octets
Longint	Entiers longs	-2147483648 à 2147489647	-12 545 454; 3 257	4 octets
Byte	Entiers sur 1 Bit (Byte ou Octet)	0 à 255	12; 157	1 octet
Word	Entiers sur 2 Bits (Word ou Mot)	0 à 65 535	27; 4 589	2 octets
Real	Nombres réels	$2.9E^{-39}$ à $1.7E^{38}$	3.1415; 789.457851	6 octets
Single	Nombres décimaux (simple précision)	$1.5E^{-45}$ à $3.4E^{38}$	3.1415926; 178 925.455678	4 octets
Double	Nombres décimaux (double précision)	$5E^{-324}$ à $1.7E^{308}$	54.5899; 9 897 669 651.45568959	8 octets
Extended	Nombres réel	$3.4E^{-4932}$ à $1.1E^{4932}$	3.14159265458; 9.81	10 octets
Comp	Entier	$-9.2E^{18}$ à $9.2E^{18}$	-271; 6 548	8 octets
Boolean	Boolean sur 1 octet	false ou true	false; true	1 octet
Wordbool	Boolean sur 1 mot	false ou true	false; true	2 octets
Longbool	Boolean sur 1 Double-Mot	false ou true	false; true	4 octets
Bytebool	Boolean sur 1 octet	false ou true	false; true	1 octet
String	Chaîne de caractères	256 caractères maximum	'Hello!'; 'Allez-vous bien ?'	256 octets
String[n]	Chaîne de n caractères	n caractères maximum	String[6]->'Hello!'	n octets
Char	1 caractère	1 caractère maximum	'R'	1 octet

Variables - Affectation

Affectation ou prise de valeurs

- ❑ Pour affecter une valeur à une variable, on utilise le commutateur " **:=** " (deux points et signe égale).
- ❑ Il faut que la valeur donnée soit compatible avec le type utilisé.

Exemple

VAR Nombre : integer; { Variable 'Nombre' du type entier }

BEGIN

Nombre **:=** 5; { On affecte 5 à la variable 'Nombre' }

Nombre **:=** Nombre + 1; { On ajoute 1 }

Nombre **:=** Nombre * Nombre; { Au carré }

WRITE(Nombre); { sortie : 36 }

END.

Affichage - Formats

Formater les sorties signifie qu'on désire leur imposer un **format** d'affichage.

Le format d'une variable de type réel :

- ❑ WriteLn (nombre : n) ; pour écrire le nombre sur n espaces.
- ❑ WriteLn (nombre : n : m) ; pour écrire le nombre sur n espaces avec m chiffres après la virgule. (un réel en possède bien plus).

Exemple: Soit x un réel; pour lequel on affecte à la valeur 1257

WriteLn(x) ⇒ 1.2570000000E+03

WriteLn(x:9) ⇒ 1.26E+03

WriteLn(x:12) ⇒ 1.25700E+03 (affichage sur 12 caractères)

WriteLn(x:12:2) ⇒ xxxx1257.00

WriteLn(x:12:5) ⇒ x1257.00000

Ce format peut être appliqué pour tous les autres types de variable .

Exemple : WriteLn ('ENCG' : 10) ;

Ici, la chaîne de caractères sera affichée sur 10 caractères: ENCG

Variables - Emplois

Emplois

Les variables peuvent être utilisées :

- Pour des comparaisons dans une structure conditionnelle
- Pour l'affichage de résultats.
- Pour le dialogue avec l'utilisateur du programme
- Pour exécuter des boucles

Structures alternatives - if...then...else

Cette commande est similaire au basic, elle se traduit par :

IF ... THEN ... ELSE

```
var Age : Integer;  
begin
```

SI ... ALORS ... SINON ...

```
    write('Entrez votre age : ');  
    read(Age);  
    if Age > 18 then
```

```
        begin
```

{ 1er bloc d'instructions }

```
        writeln('C'est bon, vous êtes majeur.');
```

```
        end
```

{ Pas de point virgule }

```
    else
```

```
        Begin
```

{ 2eme bloc d'instructions }

```
        writeln('Vous êtes mineur; ');
```

```
        end;
```

{ Ici, un point virgule }

```
end.
```

Notez que le END terminant le THEN ne possède pas de point virgule car s'il en possédait un, alors le bloc condition se stopperait avant le ELSE.

Structures alternatives - if...then...else

En Pascal, il n'y a pas de ELSEIF comme en Basic. On peut contourner l'obstacle en utilisant un **if...then...else** dans le **else** de l'instruction précédente et ainsi de suite :

Program exemple ;

var Age : Integer;

begin

write('Entrez votre age : ');

read(Age);

if Age < 18 then

{ SI Age < 18 ALORS...}

begin

writeln(' Vous êtes mineur);

end

{ Pas de point virgule }

else if Age > 65 then

{ SINON SI Age > 65 ALORS...}

begin

writeln('Vous me semblez bien vieux');

end

{ Pas de point virgule non plus}

else

{ SINON ...}

begin

writeln('C'est bon, vous êtes majeur.');

end;

{ Ici, un point virgule }

end.

Structures alternatives - Case ... Of ... End

Case ... Of ... End

L'utilisation des cas est parfois plus adaptée à certaines situations que la classique structure conditionnelle avec if...then.

Cette instruction compare la valeur d'une variable de type entier ou caractère à tout autres valeurs constantes.

Program exemple ;

Var *age*:integer ;

BEGIN

Write('Entrez votre âge : ') ;

Readln(*age*) ;

Case *age* of

0..17 : writeln('Encore mineur') ; { *intervalle de valeurs* }

18,19 : writeln('Vous êtes majeur') ; { *énumération* }

35: writeln(' l'âge d'or ') ; { *cas unitaires* }

60..99 : writeln('Vous êtes dans le troisième âge');

Else writeln(' Vous êtes d'un autre âge...') ; {aucun des cas spécifiés }

End ;

END.

Structures répétitives - For ... : = ... To ... Do ...

For ... : = ... To ... Do ...

Cette instruction permet d'incrémenter une variable à partir d'une valeur inférieur jusqu'à une valeur supérieur et d'exécuter une ou des instructions entre chaque incrémentation. Le pas de variation est l'unité et ne peut pas être changé.

Syntaxe :

```
For variable := borne inférieur To borne supérieur Do  
Begin  
  commandes ...  
End ;
```

For ... : = ... DownTo ... Do ...

Cette instruction permet de décrémenter une variable à partir d'une valeur supérieur jusqu'à une valeur inférieur et d'exécuter une ou des instructions entre chaque décrémentement.

Syntaxe :

```
For variable := borne supérieur DownTo borne inférieur Do  
Begin  
  Commandes  
End;
```

Structures répétitives - For ... : = ... To ... Do ...

```
program Table8;  
var i : integer;  
begin                                {la table de 8 }  
    For i := 1 To 10 Do  
        writeln(i, ' x 8 = ', i * 8);  
end.
```

```
program Table8;  
var i : integer;  
begin                                {table de 8 à rebours }  
    For i := 10 DownTo 1 Do  
        writeln(i, ' x 8 = ', i * 8);  
end.
```

Structures répétitives - Repeat ... Until ...

Repeat ... Until ...

Cette boucle effectue les instructions placées entre deux bornes (**repeat** et **until**) et évalue à chaque répétition une condition de type booléenne

- ❑ Il y a donc au moins une fois exécution des instructions.
- ❑ Il est nécessaire qu'au moins une variable intervenant lors de l'évaluation soit modifiée à l'intérieur de la boucle.

Syntaxe :

Repeat

...

commandes

...

Until *variable condition valeur ;*

Structures répétitives - Repeat ... Until ...

```
var Age : Integer;  
begin  
    Repeat  
        Writeln('Quel est votre age ?');  
        Readln(Age);  
    Until Age >= 18;  
    Writeln('C'est bon, vous êtes majeur.');
```

end.

Dans notre exemple il s'agit d'effectuer le **traitement** jusqu'à ce que Age soit **supérieur ou égal à 18**. La structure itérative repeat - until permettant de faire ce traitement et éventuellement **de le répéter** si la **condition de l'itération** n'est pas vérifiée.

Structures répétitives - While ... Do ...

While ... Do ...

Ce type de boucle, contrairement à la précédente, évalue une condition avant d'exécuter des instructions.

De plus, au moins une variable de l'expression d'évaluation doit être modifiée au sein de la structure de répétition pour qu'on puisse en sortir.

Syntaxe :

While *variable condition valeur* **Do**

Begin

... commandes

End ;

Structures répétitives - While ... Do ...

```
program Exemple;  
var Nombre : integer;  
begin  
    write( 'Entrez un nombre entier positif : ');  
    readln( Nombre);  
    while ( Nombre >= 0) do  
    begin  
        writeln ( Nombre);  
        dec ( Nombre)  
    end;  
end.
```

Le programme nous affiche un compte à rebours si la condition initiale est vrai ; en revanche il n'affiche rien si la condition était fausse c'est à dire si l'utilisateur a fourni un nombre négatif en entrée.

Program exemple1

Var *nom* : String ;

BEGIN

Write('Entrez votre nom : ') ;

ReadLn(nom) ;

WriteLn('Votre nom est ', nom) ;

ReadLn;

END.

Le programme *exemple1* déclare tout d'abord la variable nommée *nom* comme étant une chaîne de caractère (String). Ensuite, dans le bloc programme principal, il est demandé à l'utilisateur d'affecter une valeur à la variable *nom*. Ensuite, il y a affichage de la valeur de la variable et attente que la touche entrée soit validée (*ReadLn*).

L'équivalent de la commande *ReadLn* est *ReadKey* qui donne une valeur à une variable de type Char (caractère ASCII).

Branchement - GOTO

Un branchement peut être comparé à un pont. À chaque fois que le programme en rencontre un, il va faire un pont de ce branchement jusqu'au **label** qui lui est indiqué.

Un branchement est défini par un label qu'on choisit.

Tout label doit être déclaré préalablement en tête du programme, juste après program, à l'aide du mot-clef **label**.

Si il y en a plusieurs, il seront séparés par des **virgules**.

Dans le programme, le label est toujours suivi des doubles points (:)

Pour atteindre à ses labels, il faut utiliser l'instruction **goto** suivie du label choisi.

Branchement - GOTO

```
program Branchmt;  
  label Debut, Milieu, Fin;           { Déclaration de 3 labels }  
  var x : Integer;  
  begin  
    writeln('Où voulez-vous aller (1,2,3) ?');  
    read(x);  
    if x = 1 then goto Debut;  
    if x = 2 then goto Milieu;  
    if x = 3 then goto Fin;  
  
    Debut:                           { Label Debut }  
      writeln('Ceci est le début du programme');  
  
    Milieu:                          { Label Milieu }  
      writeln('Ceci est le milieu du programme');  
  
    Fin:                             { Label Fin }  
      writeln('Ceci est la fin du programme');  
  end.
```

Les procédures et fonctions

- ❑ Les procédures et fonctions sont des sortes de sous-programmes écrits avant le programme principal.
- ❑ Le nom d'une procédure ou d'une fonction ne doit pas excéder 127 caractères et ne pas contenir d'accent. Ce nom doit, en outre, être différent de celui d'une instruction en Pascal.
- ❑ L'appel d'une procédure peut dépendre d'une structure de boucle, de condition, etc.
- ❑ Les procédures et fonctions sont appelés depuis;
 - ❑ le programme principal,
 - ❑ d'une autre procédure
 - ❑ d'une autre fonction.
- ❑ Une procédure se distingue d'une fonction par le fait qu'elle ne renvoi pas de valeur, alors qu'une fonction en renvoi une.

Procédure simple

Une procédure peut voir ses variables définies par le programme principal.
La déclaration des variables se fait alors avant la déclaration de la procédure pour qu'elle puisse les utiliser.

Syntaxe :

Program *nom de programme* ;

Var variable : type ;

Procedure *nom de procédure* ;

Begin

...

commandes

...

End ;

BEGIN

nom de procédure ;

END.

Variables locales et sous-procédures

Une procédure peut avoir ses propres variables locales qui seront réinitialisées à chaque appel. Ces variables n'existent alors que dans la procédure.

Ainsi, une procédure peut utiliser les variables globales du programme (déclarées en tout début) mais aussi ses propres variables locales qui lui sont réservées.

Syntaxe :

Procédure nom de procédure ;

Var variable : type ;

 Procédure nom de sous-procédure ;

 Var variable : type ;

 Begin

 ...

 End ;

Begin

...

commandes

...

End ;

procédure

Sous-procédure

Procédure simple

```
program test;  
uses wincrt;  
var i,a, b:integer;  
Begin  
For i:=1 to 10 do  
begin  
Writeln('Introduire un nombre');  
read(a);  
carre;  
End;  
end.
```

```
procedure carre;  
begin  
b:=a*a;  
writeln(' le carré de ',a,' est : ' ,b);  
end;
```

Ce programme appelle une
procédure appelée carre

Procédure simple

Program exemple ;

Uses Wincrt ;

Var a, b, c : real ;

Procedure maths ;

Begin

a := a + 10 ;

b := sqrt(a) ;

c := sin(b) ;

End ;

BEGIN

Clrscr ;

Write('Entrez un nombre :) ;

Readln(a) ;

Repeat

maths ;

Writeln (c) ;

Until keypressed ;

END.

Ce programme appelle une **procédure appelée maths** qui effectue des calculs successifs. Cette procédure est appelée depuis une **boucle** qui ne se stoppe que lorsqu'une touche du clavier est pressée. (instruction keypressed).

Procédure paramétrée

On peut aussi créer des procédures paramétrées (dont les variables n'existent que dans la procédure). Ces procédures là ont l'intérêt de pouvoir être déclarée avant les variables globales du programme principal ; elles n'utiliseront que les variables passées en paramètres

Syntaxe :

```
Program nom de programme ;  
    Procedure nom de procédure( noms de variables : types ) ;  
    Begin  
        ...  
        commandes  
        ...  
    End ;  
BEGIN  
    nom de procédure ( noms d'autres variables ou leurs valeurs ) ;  
END.
```

Procédure paramétrée

Program exemple ;

Uses WinCrt ;

Procedure maths (param : Real) ;

Begin

param := Sqr(param) ;

WriteLn(param) ;

End ;

Var nombre : Real ;

BEGIN

ClrScr ;

Write('Entrez un nombre :') ;

ReadLn(nombre) ;

maths (nombre) ;

ReadLn ;

END.

La déclaration des variables se fait en même temps que la déclaration de la procédure, ces variables sont des paramètres formels car existant uniquement dans la procédure.

Procédure paramétrée (Var)

Il est quelquefois nécessaire d'appeler une procédure paramétrée sans pour autant avoir de valeur à lui affecter mais on souhaiterait que ce soit elle

- qui nous renvoie des valeurs
- qui modifie la valeur de la variable passée en paramètre.

Lors de la déclaration de variable au sein de la procédure paramétrée, la **syntaxe Var** (placée devant l'identificateur de la variable) **permet de déclarer des paramètres formels** dont la valeur à l'intérieur de la procédure ira remplacer la valeur, dans le programme principal, de la variable passée en paramètre.

Il faut savoir qu'une procédure paramétrée peut accepter plusieurs variables d'un même type et aussi plusieurs variables de types différents.

Procédure paramétrée (Var)

Ainsi, certaines variables pourront être associées au Var, et d'autres pas. Il faudra déclarer séparément (séparation par un point virgule;) les variables déclarées avec Var et les autres sans Var.

Syntaxes :

```
Procedure identifiant (Var var1, var2 : type1 ; var3 : type1) ;
```

```
Begin
```

```
...
```

```
End ;
```

```
Procedure identifiant (Var var1 : type1 ; Var var2 : type2) ;
```

```
Begin
```

```
...
```

```
End ;
```

Procédure paramétrée (var)

Uses WinCrt ;

```
Procedure maths ( var p1 : integer ; var p2,p3:real ) ;  
Begin  
p2 := Sqr(p1) ;  
p3:=p1*2;  
End ;
```

```
Var n2 ,n3: Real ;  
Var n1:integer;  
BEGIN  
ClrScr ;  
Write('Entrez un nombre :') ;  
ReadLn(n1) ;  
maths (n1,n2,n3) ;  
WriteLn(n1,' ',n2:5:2 , ' ',n2:5:2) ;  
ReadLn ;  
END.
```


Procédure paramétrée (var)

```
Program Exemple9c ;  
Uses winCrt ;
```

```
Procedure Saisie ( var nom : String ) ;  
Begin  
Write('Entrez votre nom : ' ) ;  
ReadLn(nom) ;  
End ;
```

```
Procedure Affichage ( info : String ) ;  
Begin  
WriteLn('Voici votre nom : ', info) ;  
End ;
```

```
Var chaine : String ;  
BEGIN  
ClrScr ;
```

```
Saisie(chaine) ;
```

```
Affichage(chaine) ;
```

```
END.
```

Fonctions

Elles sont appelées à partir du programme principal, d'une procédure ou d'une autre fonction. Le programme affecte des valeurs à leur variables (comme pour les procédures paramétrées, il faudra faire attention à l'ordre d'affectation des variables).

Syntaxes :

```
Function nom de fonction (variable : type ) : type ;  
    Var déclaration de variables locales ;  
    Begin  
        ...  
        commandes  
        ...  
        nom de fonction := une valeur ;  
    End ;
```

Fonctions

Program test;

Var a,b,r : real;

Function *rapport* (x,y: real) : real ;

Var z : real;

Begin

z := x/y;

Rapport := z;

End ;

Begin

Writeln('Entrer deux nombres');

Read(a,b);

r:= rapport(a,b);

Writeln(r);

End.

Fonctions

Program exemple ;

Uses Wincrt ;

Var resultat, x, n : integer ;

BEGIN .

Write ('Entrez un nombre : ') ;

Readln (x) ;

Write('Entrez un exposant : ') ;

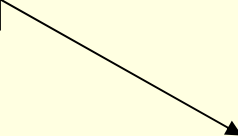
Readln (n) ;

resultat := **exposant (x , n)** ;

Writeln (resultat) ;

Readln ;

END.



Faut créer la fonction
exposant

Fonctions

Program exemple ;
Uses Wincrt ;

```
Function exposant ( i , j : integer ) : integer ;  
Var i2 , a : integer ;  
Begin  
    i2 := 1 ;  
    For a := i To j Do  
        i2 := i2 * i ;  
        exposant := i2 ;  
    End ;
```

```
Var resultat, x, n : integer ;  
BEGIN .  
    Write ('Entrez un nombre : ' ) ;  
    Readln (x) ;  
    Write('Entrez un exposant : ' ) ;  
    Readln (n) ;  
    resultat := exposant ( x , n ) ;  
    Writeln ( resultat ) ;  
    Readln ;
```

END.

Fonctions

Fonctions mathématiques

$\text{Sin}(a)$	sinus
$\text{Cos}(a)$	cosinus
$\text{ArcTan}(a)$	arctangeante
$\text{Abs}(a)$	valeur absolue
$\text{Sqr}(a)$	carré
$\text{Sqrt}(a)$	racine carré
$\text{Exp}(a)$	exponentielle
$\text{Ln}(a)$	logarithme népérien

L'argument des fonctions trigonométriques doit être exprimé en radian (Real),

Procédures et Fonctions

Inc(a) ;	Le nombre a est incrémenté de 1	Procédure
Inc(a,n) ;	Le nombre a est incrémenté de n	Procédure
Dec(a) ;	Le nombre a est décrémenté de 1	Procédure
Dec(a,n) ;	Le nombre a est décrémenté de n	Procédure
Trunc(a)	Prise de la partie entière du nombre a sans arrondis	Fonction
Int(a)	Prise de la partie entière du nombre a sans arrondis	Fonction
Frac(a)	Prise de la partie fractionnaire du nombre a	Fonction
Round(a)	Prise de la partie entière du nombre a avec arrondi à l'unité la plus proche	Fonction
Odd(a)	Renvoie true si le nombre a est impair et false si a est pair	Fonction
SizeOf(x)	Renvoie le nombre d'octets occupés par la variable x	Fonction

Architecture standard en pascal

{ les instructions facultatives pour compilation doivent être entre accolades }

Program *nom de programme ;*

Uses *unités utilisées ;*

Const *déclaration de constantes ;*

Type *déclaration de types ;*

Function *déclaration de fonction ;*

Procedure *déclaration de procédure paramétrée ;*

Var *déclaration de variables ;*

Procedure *déclaration de procédure simple ;*

BEGIN *{ programme principal }*

Commandes

END.

Types scalaires en Pascal

Les types énumérés

Si les types prédéfinis ne suffisent pas, vous pouvez créer d'autres types.

Il faut d'abord définir le type que vous désirez créer en donnant la liste ordonnée de toutes les valeurs possibles :

```
TYPE t_jour=(lun,mar,mer,jeu,vend,sam,dim);
```

Toutes les variables de ce type ne pourront pas avoir d'autres valeurs que celles que l'on a énumérées.

ensuite il faut déclarer les variables de ce type :

```
ex: VAR jour1, jour2 : t_jour;
```

Types scalaires en Pascal

On peut alors les utiliser dans le programme grâce à :

- ❑ des affectations :
JOUR1:=mer;
JOUR2:=JOUR1;
- ❑ des fonctions prédéfinies: PRED (précédent), SUCC (suivant),
ORD (numéro d'ordre (dans la déclaration), entre 0 et N-1)

pred(mar)=lun succ(mar)=mer ord(mar)=1

- ❑ des comparaisons : lun<mer mais attention dim>lun (suivant l'ordre donné dans la déclaration)

- ❑ des boucles : FOR jour1:=lun TO vend DO ...

- ❑ des sélections de cas :

CASE jour2 OF

lun : ...;

sam,dim : ...

END

Types scalaires en Pascal

Les types intervalles

Un intervalle est un ensemble de valeurs successives appartenant à un type déjà défini. Ce sont en quelques sortes des cas particuliers des énumérations.

Le type intervalle est défini par une borne inférieure et une borne supérieure séparés par un ..

- ❑ TYPE jour_travail=lun..vend; {si on a déjà défini t_jour }
- ❑ TYPE mois=1..12; {sous-ensemble du type INTEGER}
- ❑ TYPE byte=0..255; {prédéfini en TURBO}
- ❑ TYPE minuscules='a'..'z'; {sous-ensemble du type CHAR}

Les tableaux

Les tableaux sont très utiles pour stocker des données, lorsque ces dernières deviennent surabondantes.

Syntax :

Var *NomTab* : **Array**[*Min*..*Max*] **Of** *Type*;

- La ligne commence par **Var**. Il faudra donc déclarer les tableaux en même temps que les autres variables.
- *NomTab* représente un identificateur. Il doit respecter les règles énumérées précédemment pour les variables.
- **Array** permet de créer le tableau. Min définit la dimension inférieur et Max la dimension supérieur.
- **Type** définit le type du tableau : il peut être du type String, Boolean, Integer...

Les tableaux

A une dimension:

Var *NomTab* : **Array**[*Min..Max*] **Of** *Type*;



type *t_tableau* : **Array**[*Min..Max*] **Of** *Type*;

Var *NomTab* : *t_tableau*

A deux dimension

type *t_tableau* : **Array**[*Mmin..Mmax,Nmax..Nmax*] **Of**
Type;

Var *NomTab* : *t_tableau*

Les tableaux

Exemple: tableau de type Real de 10 cases

```
Var Tabl : Array[1..10] Of Real;
```

```
begin
```

```
Tabl[4] := 46.25;           {On affecte la valeur 46.25 à la case 4 du tableau}
```

```
writeln(Tabl[4]);          {On affiche la valeur de la case 4}
```

```
end.
```

1	2	3	4	5	6	7	8	9	10
			46.25						

On peut créer des tableaux:

à 2, 3, ... dimensions :

```
var Tabl : Array[1..3, 1..5] Of integer;
```

	1	2	3	4	5
1					
2					
3					

Lecture et écriture des éléments d'un tableau.

Pour avoir accès à un élément d'un tableau, il suffit d'écrire le nom de la variable de type tableau suivi de l'indice de l'élément visé entre crochets. Cet accès permet à la fois de lire et d'écrire cet élément.

Par exemple :

- Si Un Tableau est une variable du type Array of Integer, le nième élément de ce tableau sera MonTableau[n].
- Pour lui donner la valeur 5, on écrira simplement MonTableau[n]:=5.
- Pour transférer sa valeur dans la variable b, on écrira simplement b:=MonTableau[n].

Cas particulier du type String

Le type String représente les chaînes de caractères. C'est en fait un tableau qui aurait pu être déclaré comme `array[0..255] of Char`.

Le premier octet du tableau représente la longueur réelle de la chaîne considérée.

Par exemple :

Si on considère la chaîne `S:='EXEMPLE'`, on aura en mémoire un tableau de 256 caractères commençant par la suite `#7 E X E M P L E` suivie d'octets indéterminés.

On aura ainsi :

`S[1]='E', S[4]='M'`

De même on pourra modifier les caractères contenus dans la chaîne.

Caractères et chaînes de caractères

Syntaxe :

```
Var chaine : String ;  
telephone : String[10] ;
```

Le type String est en fait un tableau de caractères à une dimension dont l'élément d'indice zéro contient une variable de type Char et dont le rang dans la table ASCII correspond à la longueur de la chaîne.

Il est donc possible de modifier un seul caractère de la chaîne .

Caractères et chaînes de caractères

le type String est en fait un tableau de caractères à une dimension dont l'élément d'indice zéro contient une variable de type Char et dont le rang dans la table ASCII correspond à la longueur de la chaîne. Il est donc possible de modifier un seul caractère de la chaîne grâce à la syntaxe suivante :

chaine[index]:=lettre; Program Chaine;

```
Var nom:String;  
BEGIN
```

```
  nom:='EncG';
```

```
  nom[2]:='Z';
```

```
  nom[0]:=Chr(2);
```

```
  WriteLn(nom);
```

```
  nom[0]:=Chr(28);
```

```
  Write(nom,'-EL Jadida');
```

```
END.
```

Remplace n par Z

Longueur 2 caractères

EZ

EZcG – EL Jadida

Caractères et chaînes de caractères

Concat (s1, s2, s3, ..., sn) ;

Cette fonction concatène les chaînes de caractères spécifiées s1, s2, etc. en une seule et même chaîne.

Syntaxes :

s := Concat (s1, s2) ;

s := s1 + s2 ;

Insert (s1, s2, i) ;

Procédure qui insert la chaîne s1 dans la chaîne s2 à la position i.

Pos (s1, s2) ;

Fonction qui renvoie sous forme de variable byte la position de la chaîne s1 dans la chaîne-mère s2. Si la chaîne s1 en est absente, alors cette fonction renvoie 0 comme valeur.

Caractères et chaînes de caractères

Copy (s, i, j) ;

Fonction qui retourne de la chaîne de caractère s, un nombre j de caractères à partir de la position i (dans le sens de la lecture).

Rappelons que i et j sont des entiers (integer).

Delete (s, i, j) ;

Procédure qui supprime dans la chaîne nommée s, un nombre j de caractères à partir de la position i.

Str (x, s) ;

Procédure qui convertit le nombre (Integer ou Real) x en chaîne de caractère de nom s.

length(s);

Fonction qui renvoie sous forme de variable byte la longueur de la chaîne s

Les enregistrements

Il est parfois utile de regrouper des variables de types éventuellement différents grâce à la notion d'enregistrements. Un enregistrement contiendra donc plusieurs données qu'on appellera champs.

Déclaration d'un type enregistrement

Cette déclaration prend la forme suivante :

```
TypeEnregistrement = RECORD
```

```
Champ1 : Type1;
```

```
Champ2 : Type2;
```

```
.....
```

```
END;
```

TypeEnregistrement est le nom du type créé.

Champ1, Champ2, ... sont les noms des différents champs.

Type1, Type2, ... sont les types de ces champs.

record et end sont des mots réservés de Turbo Pascal.

Les enregistrements : déclaration du type

Par exemple :

Pour créer un type associant le nom d'une personne et sa date de naissance on pourra déclarer :

Naissance = record

Nom : String[30];

Jour : Byte;

Mois : Byte;

An : Word;

end;

Var *UnAmi* : **Naissance**;

si *UnAmi* est une variable du type **Naissance**, on aura accès au nom en écrivant UnAmi.Nom.

UnAmi.Nom:='Simo' permet d'écrire le nom.

b:=UnAmi.Jour permet de transférer le jour dans la variable b.

Architecture standard en pascal

{ les instructions facultatives pour compilation doivent être entre accolades }

Program *nom de programme ;*

Uses *unités utilisées ;*

Const *déclaration de constantes ;*

Type *déclaration de types ;*

Function *déclaration de fonction ;*

Procedure *déclaration de procédure paramétrée ;*

Var *déclaration de variables ;*

Procedure *déclaration de procédure simple ;*

BEGIN *{ programme principal }*

Commandes

END.

Lecture et écriture d'un champ.

Pour avoir accès à un champ d'une variable de type enregistrement on écrit le nom de la variable suivi d'un point et du nom du champ visé. Cet accès permet à la fois de lire et d'écrire ce champ.

L'instruction With

Cette instruction permet d'utiliser une variable de type enregistrement en ayant accès à ces champs sans avoir à répéter le nom de la variable.

Syntaxe :

```
with NomVar do
```

```
begin
```

```
(suite d'instructions où on utilise les champs de NomVar  
comme des variables )
```

```
end;
```


Lecture et écriture d'un champ.

Par exemple, avec la variable UnAmi définie précédemment, on pourra écrire :

```
Program test;
```

```
Uses WinCrt;
```

```
Type Naissance = record Nom : String[30]; Jour : Byte; Mois : Byte;  
An : Word; end;
```

```
Var UnAmi : Naissance;
```

```
BEGIN
```

```
with UnAmi do
```

```
begin
```

```
Nom:='Simo';
```

```
Jour:=17;
```

```
Mois:=5;
```

```
An:=1960;
```

```
end;
```

```
Writeln(UnAmi.nom);
```

```
END.
```

Manipulation de fichiers

Pour utiliser un ou des fichiers tout au long d'un programme, il faudra l'identifier par une variable dont le type est fonction de l'utilisation que l'on veut faire du fichier. Il existe :

- ❑ Les fichiers textes (Text), qui sont écrits au format texte (chaînes de caractères, nombres).
- ❑ Les fichiers typés (File Of), qui sont des fichiers écrits sur disque telles que les données se présentent en mémoire.
- ❑ Les fichiers tout court (File), qui sont des fichiers dont on ne connaît pas le contenu.

Syntaxe :

```
Var f : Text ;  
Var f : File Of type;  
Var f : File;
```

Assign (procedure)

Pour Turbo Pascal, toute opération sur un fichier se fait par l'intermédiaire d'une variable de type File.

Avant d'utiliser une telle variable il est nécessaire de l'initialiser en l'associant à un fichier DOS. Cette opération se réalise par l'intermédiaire de la procédure Assign.

Syntaxe :

Assign (variable d'appel , nom du fichier) ;

Exemple : pour utiliser le fichier monfic.dat contenu dans le répertoire \datas du lecteur c: on l'associera au fichier pascal f en écrivant

```
Assign(f,'c:\datas\monfic.dat');
```

Fichier type Text : Ouverture, Création

Avant de pouvoir effectuer des opérations de lecture ou d'écriture sur un fichier il est nécessaire de l'ouvrir. Cela peut se réaliser de 2 façons : si variable d'appel est f

➤ Création d'un nouveau fichier

On utilise la procédure **Rewrite(variable d'appel)**.

Un fichier Dos vide dont le nom est celui associé à f est créé. Si un fichier de même nom existe déjà, il est détruit.

➤ Ouverture d'un fichier déjà existant

On utilise la procédure **Reset(variable d'appel)**.

Le fichier Dos associé à f est ouvert et prêt pour des opérations de lecture et d'écriture.

Fichier type Text: Fermeture,

Lorsqu'on a terminé les opérations de lecture et d'écriture sur le fichier il faut le fermer. Un programme se doit de fermer tous les fichiers qu'il a ouverts.

- La **fermeture d'un fichier** s'effectue par l'intermédiaire de la procédure **Close(Variable d'appel)**.
- Pour **ajouter des lignes de texte**. La procédure **Append(var f:Text)** ouvre le fichier en fixant la position courante à la fin du fichier.

La procédure **Readln(var f:Text; Var S)** lit la ligne courante dans la chaîne S.

La procédure **Writeln(var f:Text; Var S)** écrit la chaîne S à la position courante et place les caractères #13#10.

Program fichier_ecriture ;

Uses wincrt ;

Var f : text ; ← Déclarer f comme fichier text

nom : string ;

BEGIN

Clrscr ;

Assign (f, 'c:\init.txt') ; ← Attribuer à f le nom du fichier init.txt

Rewrite (f) ; ← Créer et ouvrir un nouveau fichier

Write (' Entrez un nom d'utilisateur : ') ;

Readln (nom) ;

nom := ' Dernier utilisateur : ' + nom ;

Writeln (f, nom) ; ← Écrire le contenu de nom dans le fichier

Close (f) ;

END.


Lecture, écriture

```
Program fichier_lecture ;  
Uses wincrt ;  
Var f : text ;  
nom : string ;  
BEGIN  
Clrscr ;  
Assign (f, 'c:\init.txt') ;  
Reset (f) ;  
Readln (f, nom) ;  
Writeln (nom) ;  
Close (f) ;  
END.
```

Effacer l'écran



Attribuer à f le nom du fichier



Ouvre le fichier existant f




Lecture, écriture

```
Program fichier_ecriture ;  
Uses wincrt ;  
Var f : text ;  
n : integer;  
BEGIN  
  Clrscr ;  
  Assign (f, 'c:\init.txt') ;  
  Reset (f) ;  
  Repeat  
    Readln (f, n) ;  
    Writeln (n*2) ;  
  Until eof (f) ;  
  Close (f) ;  
END.
```

Effacer l'écran



Attribuer à f le nom du fichier



Ouvre le fichier existant f



Gestion des répertoires

Le DOS organise les fichiers d'un lecteur en répertoires. L'unité System fournit des procédures et fonctions permettant de réaliser certaines opérations globales sur les fichiers et les répertoires. Les opérations concernant les fichiers doivent être effectuées sur des fichiers fermés.

Procédures concernant les fichiers d'un répertoire

- La procédure **ERASE(f:File)** efface le fichier f du disque.
- La procédure **RENAME(f:File;NouveauNom:String)** change le nom du fichier f en NouveauNom.

Remarque : si NouveauNom contient un chemin d'accès sur le même lecteur cette procédure correspond à un déplacement du fichier d'un répertoire à l'autre.

Gestion des répertoires

Procédures concernant les répertoires

- La procédure **CHDIR(NouveauRepertoire:String)** permet de changer de répertoire courant.
- La procédure **MKDIR(NouveauRepertoire:String)** permet de créer un nouveau répertoire.
- La procédure **RMDIR(Repertoire:String)** détruit un sous-répertoire si celui-ci est vide.
- La procédure **GETDIR(Lecteur:Byte;Chemin:String)** renvoie le répertoire courant dans la variable chaîne Chemin; Lecteur est un entier pour lequel : 0 désigne le lecteur courant; 1 désigne le lecteur A; 2 désigne le lecteur B; etc...

Ouvrir d'un fichier type File

Avant de pouvoir effectuer des opérations de lecture ou d'écriture sur un fichier il est nécessaire de l'ouvrir. Cela peut se réaliser de 2 façons : si variable d'appel est f

➤ Création d'un nouveau fichier

On utilise la procédure **Rewrite(variable d'appel, taille)**.

Un fichier Dos vide dont le nom est celui associé à f est créé. Si un fichier de même nom existe déjà, il est détruit.

➤ Ouverture d'un fichier déjà existant

On utilise la procédure **Reset(variable d'appel, taille)**.

Le fichier Dos associé à f est ouvert et prêt pour des opérations de lecture et d'écriture.

Taille est facultative. Cependant il vaut 6 si on veut lire des nombres réels (Real) ou bien 256 pour des chaînes de caractères (String). Le fait que la variable taille soit de type Word implique que sa valeur doit être comprise entre 0 et 65535. Par défaut, taille=128 octets.

Variables - Les types

Le *type* de la variable est choisi en fonction du *type* de données qu'elle va recevoir.

TYPES DE VARIABLES				
Type	Description	Fourchette	Exemples	Mémoire requise
Shortint	Entiers courts	-128 à 127	-125; 0; 32	1 octet
Integer	Entiers "normaux"	-32 768 à 32 767	-30 000; 421;	2 octets
Longint	Entiers longs	-2147483648 à 2147489647	-12 545 454; 3 257	4 octets
Byte	Entiers sur 1 Bit (Byte ou Octet)	0 à 255	12; 157	1 octet
Word	Entiers sur 2 Bits (Word ou Mot)	0 à 65 535	27; 4 589	2 octets
Real	Nombres réels	$2.9E^{-39}$ à $1.7E^{38}$	3.1415; 789.457851	6 octets
Single	Nombres décimaux (simple précision)	$1.5E^{-45}$ à $3.4E^{38}$	3.1415926; 178 925.455678	4 octets
Double	Nombres décimaux (double précision)	$5E^{-324}$ à $1.7E^{308}$	54.5899; 9 897 669 651.45568959	8 octets
Extended	Nombres réel	$3.4E^{-4932}$ à $1.1E^{4932}$	3.14159265458; 9.81	10 octets
Comp	Entier	$-9.2E^{18}$ à $9.2E^{18}$	-271; 6 548	8 octets
Boolean	Boolean sur 1 octet	false ou true	false; true	1 octet
Wordbool	Boolean sur 1 mot	false ou true	false; true	2 octets
Longbool	Boolean sur 1 Double-Mot	false ou true	false; true	4 octets
Bytebool	Boolean sur 1 octet	false ou true	false; true	1 octet
String	Chaîne de caractères	256 caractères maximum	'Hello!'; 'Allez-vous bien ?'	256 octets
String[n]	Chaîne de n caractères	n caractères maximum	String[6]->'Hello!'	n octets
Char	1 caractère	1 caractère maximum	'R'	1 octet

Fermer un fichier

Lorsqu'on a terminé les opérations de lecture et d'écriture sur le fichier il faut le fermer. Un programme se doit de fermer tous les fichiers qu'il a ouverts.

- La fermeture d'un fichier s'effectue par l'intermédiaire de la procédure **Close(Variable d'appel)**.
- Le fichier associé à f est fermé. Il n'est plus possible d'effectuer des opérations de lecture et d'écriture à moins de le rouvrir avec **Reset(variable d'appel , taille)**.

La position courante dans un fichier ouvert

A chaque fichier est associée une position courante (entier de type `LongInt`) qui indique où les opérations de lecture et d'écriture vont être effectuées.

Au départ la position courante est 0.

Chaque opération de lecture ou d'écriture d'un bloc élémentaire (dont la taille est déterminée à l'ouverture) augmente la position courante d'une unité. Turbo Pascal fournit une procédure et une fonction gérant la position courante dans un fichier.

La procédure `Seek(Var f:File; n:LongInt)` fixe la position courante à n.

La fonction `FilePos(Var f:File):LongInt` renvoie la position courante.

Lecture dans un fichier

La lecture dans un fichier se fait avec la procédure

```
BlockRead(var f:File; Var Ident; NombreALire : Word[;  
Var NombreLus:Word]);
```

f est le fichier visé; Ident est le nom d'une variable de type quelconque qui recevra les informations contenues dans le fichier; NombreALire est le nombre de bloc élémentaires à lire.

Le paramètre facultatif NombreLus permet de récupérer le nombre de blocs effectivement lus.

La lecture se fait à la position courante du fichier qui est ensuite augmentée de NombreALire.

Écriture dans un fichier

L'écriture dans un fichier se fait avec la procédure

```
BlockWrite(var f:File; Var Ident; NombreAEcrire : Word[;  
Var NombreEcrits:Word]);
```

f est le fichier visé; Ident est le nom d'une variable de type quelconque qui sera écrite dans le fichier; NombreAEcrire est le nombre de bloc élémentaires à écrire.

Le paramètre facultatif NombreEcrits permet de récupérer le nombre de blocs effectivement écrits.

Evidemment, l'écriture se fait à la position courante du fichier qui est ensuite augmentée de NombreAEcrire.