Cours de Turbo Pascal 7

Par Hugo Etievant

Date de publication : 12 février 2001

Dernière mise à jour : 5 avril 2013

DÉBUTANT

Durée :

Ce cours présente tous les éléments de base de la programmation en Pascal.

Commentez

Préface 5
Note de l'équipe Pascal 5
Introduction 6
Utiliser l'éditeur et le compilateur Borland Turbo Pascal 7.0 7
Erreur 200 : division par 0 8
Mais d'où vient cette erreur ? 8
Comment y remédier ? 8
Où se procurer un patch ? 8
Chapitre 0 - Généralités 9
Architecture standard d'un source en Pascal 9
Grammaire du Pascal 9
Mots réservés du langage Pascal 10
Chapitre 1 - Entrées et sorties à l'écran 11
Write et WriteLn 11
Read et ReadLn 11
ReadKey et KeyPressed 12
Chapitre 2 - Opérateurs 13
Opérateurs mathématiques 13
Opérateurs relationnels 13
Opérateurs logiques 13
Priorité des opérateurs 13
Ensembles 13
Chapitre 3 - Variables, formats et maths 14
Déclaration 14
Prise de valeurs 14
Fonctions 15
Emplois 15
Opérations 15
Format 16
Chapitre 4 - Différents types de variables 19
Chapitre 5 - Structures conditionnelles 20
If .. then ... else 20
Case ... of ... end 20
Chapitre 6 - Structures répétitives 22
For ... := ... to ... do 22
For ... := ... downto ... do 22
Repeat ... until 23
While ... do 23
Break 24
Continue 24
Chapitre 7 - Procédures 26
Introduction 26
Procédures simples 26
Variables locales et sous-procédures 27
Procédures paramétrées 27
Le mot-clé Var 28
Chapitre 8 - Fonctions 31
Chapitre 9 - Audio 32
Sound ... Delay ... NoSound 32
Chr(7) 32
Chapitre 10 - Manipulation des fichiers 33
Déclaration 33
Les fichiers textes (Text) 33
Les fichiers typés (File of ...) 33
Les fichiers simples (File) 33
Lecture et écriture 34
Fermeture 36
Fonctions supplémentaires 37

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

Gestion du curseur....37
Fin de ligne, fin de fichier....38
Effacer un fichier....38
Renommer un fichier....38
Tronquer un fichier....39
Gestion des erreurs....39
Chapitre 11 - Graphismes....41
Chapitre 12 - Affichage en mode texte....47
Chapitre 13 - Caractères et chaînes de caractères....49
Caractères....49
Chaînes de caractères....50
Chapitre 14 - Créer ses propres unités....53
Chapitre 15 - Booléens et tables de vérité....55
Tables de vérité des opérateurs logiques....56
NOT....56
AND....56
OR....56
XOR....57
Chapitre 16 - Gestion des dates et heures....58
GetDate....58
SetDate....58
GetTime....58
SetTime....58
Exemple récapulatif sur la date et l'heure système....59
GetFTime....59
SetFTime....59
Unpacktime....60
Packtime....60
Exemple récapitulatif sur les dates et heures de fichiers....60
Chapitre 17 - Commandes système....61
Répertoires et lecteurs....61
MkDir....61
RmDir....61
ChDir....61
GetDir....61
Exemple récapitulatif....61
DiskFree....62
DiskSize....62
Environnement MS-DOS....63
DosVersion....63
DosExitCode....63
EnvCount et EnvStr....63
GetCBreak et SetCBreak....64
Fichiers....64
DosError....64
SetFAttr et GetFAttr....65
FExpand....65
FSplit....66
FileSize....66
Recherche de fichiers et répertoires....66
Mémoire vîve....67
MemAvail....67
MaxAvail....67
Chapitre 18 - Pseudo-hasard....68
Chapitre 19 - Paramètres de ligne de commande....70
Chapitre 20 - Types....71
Types simples....71
Types intervalles....71

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

Types énumérés....72
Types structurés (enregistrements)....73
Enregistrements conditionnels....75
Chapitre 21 - Tableaux....78
Chapitre 22 - Une bonne interface DOS....82
Chapitre 23 - Gestion de la mémoire par l'exécutable....85
Limite virtuelle de la mémoire....85
Passage de paramètre à un sous-programme....85
Chapitre 24 - Pointeurs....87
Déclaration....88
Accès à la variable pointée....88
New et Dispose....88
GetMem et FreeMem....91
Nil....92
Chapitre 25 - Ensembles....93
Déclarations....93
Affectations et opérations....94
Comparaisons....95
Chapitre 26 - Constantes....96
Tests d'évaluation....98

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

Préface

Le langage Pascal offre une très bonne approche de la programmation. Très utilisé dans le milieu scolaire, il permet d'acquérir des notions solides que l'on retrouve dans tous les autres langages. Le CyberZoïde est l'un des très rares site web à proposer un véritable cours de programmation en Pascal avec de très nombreux exemples et programmes annotés en libre téléchargement.
Les éléments de base de la programmation tels que : pointeurs, types, tableaux, procédures, fonctions, graphismes... et bien d'autres vous sont expliqués avec le maximum de pertinence, de simplicité et d'efficacité, puisque vous êtes déjà très nombreux (étudiants comme professeurs d'Université) à vous fier à ce cours.
De plus, vous disposez également de plusieurs tests d'évaluation qui vous permettent d'évaluer vos connaissances en Pascal. Enfin, les travaux pratiques de filière 3 de l'Université Claude Bernard (Lyon 1 (69), FRANCE) sont régulièrement corrigés et mis en téléchargement sur ce site.

Note de l'équipe Pascal

Le présent cours, dont la dernière version date de 2004, a été remis en page et corrigé en 2009 par l'équipe Pascal, avec l'accord de l'auteur.
Nous tenons à remercier tout particulièrement **Droggo** pour ses nombreuses corrections.

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

Introduction

Cette aide électronique sur la programmation en Turbo Pascal 7.0 est destinée en premier lieu aux non-initiés, à tous ceux qui débutent dans la programmation. Que ce soit dans le cadre de l'enseignement à l'Université ou pour votre intérêt personnel, vous avez décidé d'apprendre ce langage qui a le mérite de former à la logique informatique. Le langage Pascal est très structuré et constitue en lui-même une très bonne approche de la programmation.
Vous découvrirez, dans les pages qui vont suivre, les bases de la programmation en général : les structures de boucle et de contrôle, l'utilisation de la logique booléenne, la chronologie d'exécution du code... Ces notions de base vous serviront si vous décidez de changer de langage de programmation, car les principes de base (et même certaines instructions de base) sont les mêmes.
Dans la vie courante, nous n'avons pas pour habitude de nous limiter au strict minimum lorsqu'on communique; ici, ce principe est bafoué, puisque d'une langue vivante complexe vous allez passer à un langage strict, rigide et pauvre. Issue des mathématiques, cette langue exacte est par essence optimisée et simplifiée. Par delà, l'apprentissage d'un langage informatique forme à la systémique mathématico-informatique. Vous apprendrez à dominer le comportement de la machine et à être plus clair et précis dans votre manière de construire vos idées.

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

Utiliser l'éditeur et le compilateur Borland Turbo Pascal 7.0

Pour ouvrir un fichier, allez dans le menu **File/Open...** ou tapez la touche de fonction **F3**.
Pour exécuter un programme, allez dans le menu **Run/Run** ou tapez la combinaison de touches **Ctrl+F9**.
Pour compiler "correctement" un exécutable, allez dans le menu **Compile/Make** (ou **Compile/Compile**) ou tapez **F9** : on obtient ainsi des exécutables de meilleure qualité qui pourront être utilisés sur d'autres ordinateurs.
Si vous avez omis de mettre une pause à la fin d'un programme, ou si vous désirez tout simplement avoir sous les yeux la dernière page d'écran, il vous suffit d'aller dans le menu **Debug/User Screen** ou de taper **Alt+F5**.
Pour une aide, allez dans le menu **Help/Index** ou tapez **Shift+F1**. Pour obtenir de l'aide sur une instruction qui apparaît dans le source, placez le curseur du clavier dessus et allez dans le menu **Help/Topic Search** : une fenêtre apparaîtra alors.
Si un problème a lieu lors de l'exécution d'un programme, utilisez le débugger : **Debug/Watch**. Une fenêtre apparait en bas de page. Cliquez sur **Add** et tapez le nom de la variable dont vous désirez connaître la dernière valeur.

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

Erreur 200 : division par 0

Nombreux sont ceux d'entre vous qui ont eu un grave pépin avec le compilateur Turbo Pascal. En effet, l'exécution d'un programme utilisant l'unité **Crt** provoque un bug sur les ordinateurs récents. L'erreur observée est la suivante : ***Error 200 : division by zero***.

Mais d'où vient cette erreur ?

Sur les ordinateurs dont la vitesse du processeur dépasse 200 MHz (donc, toutes les machines actuelles), une erreur d'exécution se produit lors de l'initialisation de l'unité **Crt**. Lorsqu'il s'agit de déterminer le nombre d'itérations d'une boucle simple pour un nombre donné de cycles processeur (phase d'étalonnage), une erreur de débordement se produit et déclenche une exception interprétée par Turbo Pascal comme une division par zéro : la fameuse erreur 200. La solution est toutefois très simple, puisqu'il suffit de remplacer le compteur 16 bits par un compteur 32 bits.

Comment y remédier ?

Pour pouvoir utiliser de nouveau l'unité Crt dans vos programmes, il vous faut soit changer quelques fichiers propres au compilateur, soit appliquer un patch à chacun de vos programmes compilés avant de pouvoir les exécuter normalement. Notez que la compilation du programme ne provoque aucune erreur, c'est seulement son exécution qui provoque cette erreur de division par zéro.

Où se procurer un patch ?

Plusieurs patches sont disponibles sur le net mais le plus simple est de télécharger **la version de Turbo Pascal de Developpez.com** !

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

Chapitre 0 - Généralités

Architecture standard d'un source en Pascal

```
{ Les commentaires peuvent êtres entre accolades... }
(* ...ou encadrés comme ceci *)

Program { nom de programme } ;   { Voir note (1) }

Uses { unités utilisées } ;

Const { déclaration de constantes } ;

Type { déclaration de types } ;

Var { déclaration de variables } ;

Function { déclaration de fonction } ;

Procedure { déclaration de procédure } ;

BEGIN { début du programme principal }

  { Le corps du programme principal }

END.
```

Voir note (1) au bas de la page

Grammaire du Pascal

- Un nom de programme respecte les règles liées aux identificateurs (cf. plus bas) et ne peut pas contenir le caractère point ".".
- Un programme principal débute toujours par BEGIN et se termine par END. (avec un point). Alors qu'un sous-programme (ou fonction, procédure, bloc conditionnel...) commence lui aussi par Begin mais se termine par End; (sans point mais avec un point-virgule).
- Chaque instruction doit se terminer avec un point-virgule. Il n'y a pas d'exception à la règle hormis Begin et l'instruction précédant End ou Else.
- Il est permis de mettre plusieurs instructions les unes à la suite des autres sur une même ligne du fichier source mais il est recommandé de n'en écrire qu'une par ligne : c'est plus clair et en cas de bogue, on s'y retrouve plus aisément. De plus, s'il vous arrive d'écrire une ligne trop longue, le compilateur vous le signifiera avec l'erreur ***Error 11 : Line too long***. Il vous faudra alors effectuer des retours à la ligne comme le montre l'exemple suivant :

---

(1) La forme canonique de l'en-tête d'un programme est :
Programtartenpion (paramètres);
dans lequel *paramètres* est la liste des paramètres du programme.
Ces paramètres sont les noms des fichiers de type TXT externes au programme, qui peuvent lui être passés par la ligne de commande.
Deux de ces fichiers sont prédéfinis : **Input** et **Output**, pour respectivement la lecture des données et leur sortie, qui sont par défaut le clavier et l'écran, et dans ce cas, on peut s'abstenir de les déclarer dans l'en-tête.
Au lieu de faire
Programtartenpion (input,output);
on peut donc se permettre de faire
Programtartenpion;
mais attention, les anciens compilateurs n'acceptent pas cette écriture.

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

```
WriteLn('Fichier: ', file,
' Date de création:', datecrea,
' Utilisateur courant:', nom,
' Numéro de code:', Round(ArcTan(x_enter) * y_old):0:10);
```

- Les noms de constantes, variables, procédures, fonctions, tableaux, etc (appelés **identificateurs**) devraient être des noms simples. Par exemple, évitez d'appeler une variable comme ceci : x4v_t3la78yugh456b2dfgt mais appelez-la plutôt comme cela : discriminant (pour un programme sur les éq du 2nd degré) ou i (pour une variable de boucle).

- Les identificateurs doivent impérativement être différents de ceux d'unités utilisées, de mots réservés du langage Pascal et ne doivent pas excéder 127 signes (1 lettre au minimum). Les identificateurs ne peuvent contenir que les caractères **a** à **z**, **A** à **Z**, les chiffres **0** à **9**, et le caractère de soulignement **_**. Ils doivent obligatoirement commencer par une lettre.

- N'hésitez pas à insérer des commentaires dans votre code, cela vous permettra de comprendre vos programmes un an après les avoir écrit, et ainsi d'autres personnes n'auront aucun mal à réutiliser vos procédures, fonctions... Procédez ainsi :

```
{ Ici votre commentaire entre accolades }
(* Ici votre commentaire entre parenthèses et étoiles *)
```

- Vos commentaires peuvent tenir sur une seule ligne comme sur plusieurs. Vous pouvez aussi mettre temporairement en commentaire une partie de votre programme que vous souhaitez ne pas compiler.

- Un identificateur ne peut être égal à un mot réservé du langage Pascal !

Mots réservés du langage Pascal

- AND - ARRAY - ASM
- BEGIN
- CASE - CONST - CONSTRUCTOR
- DESTRUCTOR - DIV - DO - DOWNTO
- 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

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

Chapitre 1 - Entrées et sorties à l'écran

Write et WriteLn

La procédure Write permet d'afficher du texte et de laisser le curseur à la fin du texte affiché. Cette instruction permet d'afficher des chaînes de caractères n'excédant pas 255 signes ainsi que des valeurs de variables, de constantes, de types... Le texte doit être entre apostrophes. Si le texte à afficher contient une apostrophe, il faut alors la doubler. Les différents noms de variables doivent êtres séparés par des virgules.

Toute instruction doit être suivie d'un point virgule, à l'exception de la dernière instruction avant la fermeture d'un bloc d'instructions par end.

Syntaxe :

```
Write ('Texte à afficher', variable1, variable2, 'texte2');
Write ('L''apostrophe se double.');
```

La procédure WriteLn est semblable à la précédente à la différence près que le curseur est maintenant renvoyé au début de la ligne suivante.
Syntaxe :

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

Read et ReadLn

La procédure Read permet à l'utilisateur de rentrer une valeur qui sera utilisée par le programme. Cette instruction ne provoque pas de retour chariot, c'est-à-dire que le curseur ne passe pas à la ligne.
Syntaxe :

```
Read (variable);
```

La procédure ReadLn permet à l'utilisateur d'entrer une valeur qui sera utilisée par le programme. Cette instruction provoque le retour chariot, c'est-à-dire que le curseur passe au début de la ligne suivante. Lorsqu'aucune variable n'est affectée à la commande, il suffit de presser sur ***Entrée***.
Syntaxe :

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

Exemple :

```
Program Exemple1;
Var nom : String;
BEGIN
  Write('Entrez votre nom : ');
  ReadLn(nom);
  WriteLn('Votre nom est ', nom);
  ReadLn;
END.
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

Ce programme *Exemple1* déclare tout d'abord la variable nommée nom comme étant une chaîne de caractères (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 pressée (ReadLn).

ReadKey et KeyPressed

Pour les entrées au clavier, il existe également la fonction ReadKey, qui donne une valeur à une variable de type Char (caractère ASCII).
Syntaxe :

```
x := ReadKey;
```

Pour terminer, une fonction booléenne (qui renvoie *vrai* ou *faux*) indique simplement si un caractère a été entré au clavier : ReadKey. Elle peut être très utile pour sortir d'une boucle.
Syntaxe :

```
repeat
  {...
  commandes
  ...}
until KeyPressed ;
```

Exemple :

```
Program Exemple2;
Uses Crt;
Var i : Integer;
Const bornesup = 10000 ;
BEGIN
  i := 0;
  repeat
    WriteLn(sqrt(i));
 Inc(i);
  until (i = bornesup) or KeyPressed;
END.
```

Ce programme *Exemple2* répète une boucle jusqu'à ce qu'une valeur soit atteinte (bornesup) mais s'arrête si on appuie sur une touche, grâce à KeyPressed. L'instruction Inc(a) incrémente (augmente) la variable a de 1 (cette dernière étant de type Integer).

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

Chapitre 2 - Opérateurs

Opérateurs mathématiques

Opérate	Signification
+	Addition - Union
-	Soustraction - Complément
*	Multiplication - Intersection
/	Division
div	Quotient de la division entière
mod	Modulus : c'est le reste de la division entière
=	Egalité

Opérateurs prioritaires : *, /, div et mod.
Opérateurs secondaires : + et -.
Vous pouvez utiliser des parenthèses.

Opérateurs relationnels

Opérate	Signification
<	Inférieur strict
<=	Inférieur ou égal - Inclus
>	Supérieur strict
>=	Supérieur ou égal - Contient
<>	Différent

Opérateurs logiques

Opérate	Signification
AND	Le "et" logique des maths (voir **chapitre 15** sur les booléens et tables de vérité)
OR	Le "ou"
XOR	Le "ou" exclusif
NOT	Le "non"

Opérateur ultra-prioritaire : NOT.
Opérateur semi-prioritaire : AND.
Opérateurs non prioritaires : OR et XOR.

Priorité des opérateurs

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

Ensembles

Les opérateurs union, complément, intersection, inclus et contient s'appliquent aux ensembles (voir **chapitre 25**).

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

Chapitre 3 - Variables, formats et maths

Déclaration

Toutes les variables doivent êtres préalablement déclarées avant d'être utilisées dans le programme, c'est-à-dire qu'on leur affecte un type (voir les **types de variables**).
Elles peuvent être déclarées :

- Au tout début du programme, avec la syntaxe `VAR nom_de_la_variable : type;`. Elles seront alors valables pour le programme dans son intégralité (sous-programmes, fonctions, procédures...).
  On les appellera **variables globales**.
- Au début d'une procédure ou fonction, avec la syntaxe précédente. Elles ne seront valables que dans la procédure.
  On les appellera **variables locales**.

Prise de valeurs

Les variables sont faites pour varier, il faut donc pouvoir leur donner différentes valeurs au moyen de l'opérateur d'affectation `:=` (deux points égale) ou de certaines fonctions. Il faut bien sûr que la valeur donnée soit compatible avec le type utilisé. Ainsi, on ne peut donner la valeur '*bonjour*' à un nombre entier (`Integer`).
Exemples :

```
Y := 2009;
```

On donne ainsi la valeur 2009 à la variable *Y* (déclarée préalablement en `Integer`).

```
Lettre := 'a';
```

On affecte la valeur '*a*' à la variable *Lettre* (déclarée préalablement en `Char`).

```
Continuer := true;
```

On donne la valeur `true` (vrai) à la variable *Continuer* (déclarée préalablement en `Boolean`).

```
Nombre := Y + 103;
```

Il est également possible d'utiliser les valeurs d'autres variables, du moment qu'elles sont de même type; sinon, il faut faire des conversions au préalable.

```
Delta := Sqr(b) - 4 * (a * c);
```

On peut donc également utiliser une expression littérale mathématique dans l'affectation de variables. Mais attention à la priorité des opérateurs (voir **opérateurs**).

```
Phrase := 'Bonjour' + Chr(32) + Nom;
```

On peut aussi concaténer (ajouter) des variables `String` (voir **chapitre 13** sur les chaînes de caractères).

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

Fonctions

Fonctions mathématiques Pascal de base :

Syntaxe	Fonction
Sin	sinus
Cos	cosinus
Arctan	arctangente
Abs	valeur absolue
Sqr	carré
Sqrt	racine carrée
Exp	exponentielle
Ln	logarithme népérien

L'argument des fonctions trigonométriques doit être exprimé en radians (Real), à vous donc de faire une conversion si nécessaire. De plus, on peut voir que les fonctions tangente, factorielle n'existent pas : il faudra donc créer de toutes pièces les fonctions désirées (voir **fonctions**).

Emplois

Les variables peuvent êtres utilisées dans de nombreux emplois :

- Pour des comparaisons dans une structure conditionnelle (voir **chapitre 5** sur les conditions);
- Pour l'affichage de résultats (voir **chapitre 1** sur l'affichage);
- Pour le dialogue avec l'utilisateur du programme (voir **chapitre 1** sur les entrées au clavier);
- Pour exécuter des boucles (voir **chapitre 6** sur les structures répétitives);
- ...

Opérations

Syntaxe	Utilisation	Type des variables	Description
Inc(a);	Procédure	Tout scalaire	Le nombre a est incrémenté de 1
Inc(a,n);	Procédure	Tout scalaire	Le nombre a est incrémenté de n
Dec(a);	Procédure	Tout scalaire	Le nombre a est décrémenté de 1
Dec(a,n);	Procédure	Tout scalaire	Le nombre a est décrémenté de n
Trunc(a);	Fonction	Tout réel	Prise de la partie entière du nombre a sans arrondis
Int(a);	Fonction	a : tout réel Int(a) : Longint	Prise de la partie entière du nombre a sans arrondis
Frac(a);	Fonction	Tout réel	Prise de la partie fractionnaire du nombre a
Round(a);	Fonction	a : tout réel Round(a) : Longint	Prise de la partie entière du nombre a avec

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

			arrondi à l'unité la plus proche
Pred(x);	Fonction	Tout scalaire	Renvoie le prédécesseur de la variable x dans un ensemble ordonné
Succ(x);	Fonction	Tout scalaire	Renvoie le successeur de la variable x dans un ensemble ordonné
High(x);	Fonction	Tout scalaire	Renvoie la plus grande valeur possible que peut prendre de la variable x
Low(x);	Fonction	Tout scalaire	Renvoie la plus petite valeur possible que peut prendre de la variable x
Odd(a);	Fonction	a : entier Odd(a) : Boolean	Renvoie true si le nombre a est impair et false si a est pair
SizeOf(x);	Fonction	x : tous SizeOf(x) : Integer	Renvoie le nombre d'octets occupés par la variable x

Format

Sachez encore que le format (le nombre de signes) d'une variable de type réel peut être modifié lors de son affichage.
Exemples :

```
WriteLn(nombre:5);
```

Cela signifie : afficher nombre en utilisant une largeur minimale de 5 caractères, en ajoutant si nécessaire des espaces à gauche pour atteindre cette largeur minimale (cadrage 'à droite'). S'il faut plus de 5 caractères pour afficher nombre, alors la largeur utilisée sera augmentée en conséquence.

- **Pour un entier :**

```
Var
  n : Integer;
...
  n := 56;
  WriteLn(n:3);
...
```

donnera pour affichage ([et] montrent les limites du champ affiché) :

```
[ 56]
```

soit 1 espace + 2 caractères pour n.
Alors que :

```
Var
  n : Integer;
...
  n := 35767;
  WriteLn(n:3);
...
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

donnera

```
[35767]
```

soit plus large que ce qu'on a demandé, car on ne peut pas afficher la valeur avec la largeur demandée.

- **Pour un réel :**

Si on n'a pas précisé le nombre de chiffres pour la partie décimale, il y aura affichage au format "exponentiel", c'est-à-dire quelque chose comme *5.6E+002*.

```
Var
  n : Real;
...
  n := 56;
  WriteLn(n:3);
...
```

donnera pour affichage :

```
[ 5.6E+02]
```

alors que si la largeur demandée excède la longueur le nombre de caractères nécessités pour l'affichage comme ci dessus, on va compléter la partie décimale par des 0 (zéro), ceci tant qu'on n'a pas affiché un nombre de caractères maximum, variant selon le type de réel utilisé (real, single, ...). Si ce nombre maximal est dépassé, alors on complète avec des espaces à gauche.
Ainsi :

```
Var
  n : Real;
...
  n := 56;
  WriteLn(n:17);
...
```

donnera (on est au max de chiffres pour real) :

```
[ 5.6000000000E+01]
```

alors que

```
WriteLn(n:20);
```

donnera

```
[    5.6000000000E+01]
```

Autre syntaxe :

```
WriteLn(Nombre:0:5);
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

Cela signifie : afficher nombre (réel) sans espace avant, et en affichant 5 chiffres de la partie décimale, en complétant ces décimales par des 0 (zéro) si nécessaire.
A la longueur 5 des chiffres décimaux, s'ajoutent les chiffres de la partie entière, + le point décimal.
Exemples :

```
Var
  r : Real;
...
  r := 10.1245214;
  WriteLn(r:0:5);
...
```

donnera pour affichage ([et] montrent les limites du champ affiché) :

```
[10.12452]
```

alors que

```
Var
  r : Real;
...
  r := 10.12;
  WriteLn(r:0:5);
...
```

donnera

```
[10.12000]
```

Dans la définition du Pascal (Jensen et Wirth), l'affichage d'un type réel commence toujours par au moins un espace, ce qui n'est pas le cas pour tous les autres types.

Pour pouvez appliquer un format pour tous les autres types de variable de manière générale, si vous ne stipulez que le nombre d'espaces à afficher devant votre texte ou valeur.
Exemple :

```
WriteLn('Coucou':20);
```

Cela signifie : comme pour un nombre, on demande une largeur minimale pour l'affichage, donc ici 20 caractères, en ajoutant si nécessaire des espaces à gauche pour atteindre cette largeur minimale (cadrage "à droite").
L'affichage sera donc :

```
[              Coucou]
```

soit 14 espaces, puis la chaîne qui fait 6 caractères = 20 au total.

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

Chapitre 4 - Différents types de variables

On peut donner n'importe quel nom aux variables, à condition qu'il ne fasse pas plus de 127 caractères et qu'il ne soit pas utilisé par une fonction, procédure, unité ou instruction déjà existante.
Les identificateurs ne doivent pas contenir de caractères accentués, ni d'espace. Ils doivent exclusivement être composés des 26 lettres de l'alphabet, des 10 chiffres et du caractère de soulignement. De plus, Turbo Pascal ne différencie aucunement les majuscules des minuscules et un chiffre ne peut pas être placé en début de nom de variable.
Voici une petite liste-exemple très loin d'être exhaustive :

Désignation	Description	Bornes	Place en mémoire
Real	Réel	2.9E-039 et 1.7E+038	6 octets
Single (*)	Réel	1.5E-045 et 3.4E+038	4 octets
Double (*)	Réel	5.0E-324 et 1.7E+308	8 octets
Extended (*)	Réel	3.4E-4932 et 1.2E+4932	10 octets
Comp (*)	Réel (en réalité, entier 64 bits)	-2E+063 +1 et 2E+063 +1	8 octets
Integer	entier signé	-32768 et 32767	2 octets
LongInt	Entier signé	-2147483648 et 2147483647	4 octets
ShortInt	Entier signé	-128 et 127	1 octet
Word	Entier non signé	0 et 65535	2 octets
Byte	Entier non signé	0 et 255	1 octet
Boolean	Booléen	True ou False	1 octet
Array [1..10] of xxx	Tableau de 10 colonnes fait d'éléments de type défini xxx (Char, Integer...)		
Array [1..10,1..50, 1..13] of xxx	Tableau en 3 dimensions fait d'éléments de type défini xxx (Char, Integer...)		
String	Chaîne de caractères		256 octets
String [y]	Chaîne de caractère ne devant pas excéder y caractères		y + 1 octets
Text	Fichier texte		
File	Fichier		
File of xxx	Fichier contenant des données de type xxx (Real, Byte...)		
Char	Caractère ASCII		1 octet
"Pointeur"	Adresse mémoire		4 octets
DateTime	Format de date		
PathStr	Chaîne de caractères (nom complet de fichier)		
DirStr	Chaîne de caractères (chemin de fichier)		
NameStr	Chaîne de caractères (nom de fichier)		
ExtStr	Chaîne de caractères (extension de fichier)		

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

Chapitre 5 - Structures conditionnelles

If .. then ... else ...

Cette instruction se traduit par : **Si … alors … sinon ….**

```
Program Exemple3a;

Var Nombre : Integer;

Begin
  Write('Entrez un entier pas trop grand : ');
  Readln(Nombre);
  if Nombre < 100 then WriteLn(Nombre, ' est inférieur à cent.') else
  WriteLn(Nombre, ' est supérieur ou égal à cent.');
End.
```

Ce programme *Exemple3a* compare un nombre entré par l'utilisateur au scalaire 100. Si le nombre est inférieur à 100, alors il affiche cette information à l'écran, sinon il affiche que le nombre entré est supérieur ou égal à 100.

```
Program Exemple3b;

Var Nombre : Integer;

Begin
  Write('Entrez un entier pas trop grand : ');
  Readln(Nombre) ;
  if Nombre < 100 then
    begin
      WriteLn(Nombre, ' est inférieur à cent.');
    end
  else
    begin
      WriteLn(Nombre, ' est supérieur ou égal à cent.');
    end;
End.
```

Ce programme *Exemple3b* fait strictement la même chose que le 3a mais sa structure permet d'insérer plusieurs autres commandes dans les sous-blocs then et else. Notez que le end terminant le then n'est pas suivi d'un point virgule car, si c'était le cas, le else n'aurait rien à faire ici et le bloc condition se stopperait avant le else.
Il est également possible d'insérer d'autres blocs if dans un else, comme l'illustre l'exemple qui suit :

```
Program Exemple3c;

Var i : Integer;

Begin
  Randomize;
  i := random(100);
  if i < 50 then WriteLn ( i, ' est inférieur à 50.')
  else if i < 73 then WriteLn ( i, ' est supérieur ou égal à 50, et inférieur à 73.')
  else WriteLn ( i, ' est supérieur ou égal à 73.')
End.
```

Case ... of ... end

Cette instruction compare la valeur d'une variable de type scalaire à tout un tas d'autres valeurs constantes.

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

Attention car Case of ne permet de comparer une variable qu'avec des constantes.

```
Program Exemple4;

Var Age : Integer;

Begin
  Write('Entrez votre âge : ');
  Readln(Age);
  case Age of
    18 : WriteLn('La majorité, pile-poil !');
    0..17 : WriteLn('Venez à moi, les petits enfants...');
    60..99 : WriteLn('Les infirmières vous laissent jouer sur l''ordinateur à votre âge ?!!!')
  else WriteLn('Vous êtes d''un autre âge...');
  end ;
End.
```

Ce programme *Exemple4a* vérifie certaines conditions quant à la valeur de la variable age dont la valeur est entrée l'utilisateur. Et là, attention : le point-virgule avant le else est facultatif. Mais pour plus sécurité afin de ne pas faire d'erreur avec le bloc if, choisissez systématiquement d'ommettre le point-virgule avant un else.
Note : On peut effectuer un test de plusieurs valeurs en une seule ligne par séparation avec une virgule si on souhaite un même traitement pour plusieurs valeurs différentes. Ainsi la ligne :

```
0..17 : writeln('Venez à moi, les petits enfants…');
```

peut devenir :

```
0..10, 11..17 : writeln('Venez à moi, les petits enfants…');
```

ou encore :

```
0..9, 10, 11..17 : writeln('Venez à moi, les petits enfants…');
```

ou même :

```
0..17, 5..10 : writeln('Venez à moi, les petits enfants…');
```

(cette dernière ligne est stupide mais syntaxiquement correcte !).

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

Chapitre 6 - Structures répétitives

For ... := ... to ... do ...

Cette instruction permet d'incrémenter une variable à partir d'une valeur inférieure jusqu'à une valeur supérieure et d'exécuter une ou des instructions entre chaque incrémentation. Les extrema doivent être de type scalaire. La boucle n'exécute les instructions de son bloc interne que si la valeur inférieure est effectivement inférieure ou égale à celle de la borne supérieure.

Le pas de variation est l'unité et ne peut pas être changé.

Syntaxe :

```
For variable := borne_inferieure to borne_superieure do {instruction};
```

Autre syntaxe :

```
For variable := borne_inferieure to borne_superieure do
  begin
    ...
  end;
```

Exemple :

```
Program Exemple5;

Var i : Integer;

Begin
  For i := 10 to 53 do WriteLn ('Valeur de i : ', i);
End.
```

For ... := ... downto ... do ...

Cette instruction permet de décrémenter une variable à partir d'une valeur supérieure jusqu'à une valeur inférieure et d'exécuter une ou des instructions entre chaque décrémentation.
S'appliquent ici les mêmes remarques que précédement.
Syntaxe :

```
For variable := borne_superieure downto borne_inferieur do {instruction};
```

Autre syntaxe :

```
For variable := borne_superieure downto borne_inferieure do
  begin
    ...
  end;
```

Exemple :

```
Program Exemple6;
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

```
Var i : Integer;

Begin
  For i := 100 downto 0 do
    begin
      WriteLn ('Valeur de i : ', i );
    end;
End.
```

Repeat ... until ...

Cette boucle effectue les instructions placées entre deux bornes (repeat et until) et évalue à chaque répétition une condition de type booléenne avant de continuer la boucle pour décider l'arrêt ou la continuité de la répétition.

Il y a donc au moins une fois exécution des instructions.

Syntaxe :

```
Repeat
  ...
until condition_est_vraie;
```

Exemple :

```
Program Exemple7;

Var i : Integer;

Begin
  Repeat
    Inc(i);
    Writeln ('Boucle itérée ', i, ' fois.');
  until i > 20 ;
End.
```

Ce programme *Exemple7* permet de répéter l'incrémentation de la variable i jusqu'à ce que i soit supérieure à 20.
Note :
L'instruction Inc permet d'incrémenter une variable d'une certaine valeur. L'instruction Dec permet au contraire de décrémenter une variable d'une certaine valeur. Si la valeur en question est 1, elle peut être omise.
Ces instructions permettent d'éviter la syntaxe : variable := variable + 1 et variable := variable - 1.
Syntaxe :

```
Inc(variable,nombre);
Dec(variable);    (* Décrémentation de 1 *)
```

While ... do ...

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

C'est-à-dire qu'on peut ne pas entrer dans la structure de répétition si les conditions ne sont pas favorables.

Syntaxe :

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

```
While condition_est_vraie do {instruction};
```

Autre syntaxe :

```
While condition_est_vraie do
  begin
    ...
  end;
```

Exemple :

```
Program Exemple8;

Var
  CodeCorrect : Boolean;
  Essai : String;

Const
  LeVraiCode = 'password';

Begin
  CodeCorrect := False;
  While not CodeCorrect do
    begin
      Write('Entrez le code secret : ');
      Readln(Essai);
      if Essai = LeVraiCode then CodeCorrect := True;
    end ;
End.
```

Break

Il est possible de terminer une boucle For, While ou Repeat> en cours grâce à la commande Break lorsque celle-ci est placée au sein de la boucle en question.
Exemple :

```
Program Arret;

Var i, x : Integer;

Begin
  x := 0 ;
  Repeat
    Inc(i);
    Break;
    x := 50;
  until i > 10;
  WriteLn(i);
  WriteLn(x);
End.
```

L'exécution des instructions contenues dans la boucle repeat est stoppée à l'instruction Break, et donc x := 50; n'est jamais exécuté. En sortie de boucle, i vaudra donc 1 et x vaudra 0.

Continue

Continue interrompt l'exécution du tour de boucle en cours, et renvoie à l'instruction de fin du bloc d'instructions de la boucle, qui passe alors au tour suivant (ou s'arrête, si la condition d'arrêt est vraie).
Par exemple, avec le programme suivant :

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

```
Program test_continue;

Var
  i : Integer

Begin
  i := 0;
  while i < 13 do
    begin
      Inc(i);
      if i < 10 then Continue;
      WriteLn('i = ',i);
    end;
End.
```

on obtient comme résultat :

```
i = 10
i = 11
i = 12
i = 13
```

car la ligne

```
if i < 10 then Continue;
```

renvoie à la fin de la boucle si i est inférieur à 10, et renvoie donc au end; ,ce qui fait sauter le Writeln('i = ',i); et la boucle reprend au début tant que la condition de sortie reste fausse.
Le principe est le même pour les autres types de boucle.

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

Chapitre 7 - Procédures

Introduction

Les procédures et fonctions sont des sortes de sous-programmes écrits avant le programme principal mais appelés depuis ce programme principal, d'une autre procédure ou d'une autre fonction. Le nom d'une procédure ou d'une fonction (ou comme celui d'un tableau, d'une variable ou d'une constante) de doit pas excéder 127 caractères et ne pas contenir d'accent. Ce nom doit, en outre, être différent de celui d'un des mots réservés du Pascal. L'appel d'une procédure peut dépendre d'une structure de boucle, de condition, etc.

Procédures simples

Une procédure *peut* utiliser des **variables globales**, définies par le programme principal, c'est-à-dire que ces variables sont valables pour tout le programme et accessibles partout dans le programme principal mais aussi dans les procédures et fonctions qui auront été déclarées après. La déclaration des variables se fait alors avant la déclaration de la procédure, pour qu'elle puisse les utiliser. Car une procédure déclarée avant les variables ne peut pas connaître leur existence et ne peut donc pas les utiliser.

Il faut éviter, tant que faire se peut, d'utiliser des variables globales directement dans une procédure.
Idéalement, une procédure doit recevoir les données dont elle a besoin sous forme de paramètres *(développés un peu plus loin).*

Syntaxe :

```
Program nom_de_programme;

Var variable : type;

Procedure nom_de_procedure;
Begin
  ...
  ...
End;

BEGIN
  ...
  nom_de_procedure;
  ...
END.
```

Exemple :

```
Program Exemple9a;

Uses Crt;

Var a, b, c : Real;

Procedure Maths;
Begin
  a := a + 10;
  b := sqrt(a);
  c := sin(b);
End;

BEGIN
  ClrScr;
  Write('Entrez un nombre :');
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

```
  ReadLn(a);
  repeat
    Maths;
    Writeln(c);
  until KeyPressed;
END.
```

Ce programme *Exemple9a* 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 (fonction KeyPressed). Durant cette procédure, on additionne 10 à la valeur de a entrée par l'utilisateur, puis on effectue la racine carrée (sqrt) du nombre ainsi obtenu, et enfin, on cherche le sinus (sin) de ce dernier nombre.

Variables locales et sous-procédures

Une procédure peut avoir ses propres **variables locales**.

> ***Il n'y a pas d'initialisation automatique des variables locales; il faut donc le faire explicitement dans la procédure.***
> ***Il ne faut pas compter sur la pérennité des valeurs de ces variables locales entre deux appels de la procédure.***

Les variables n'existent 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. Une procédure ne peut pas appeler une variable locale appartenant à une autre procédure. Les variables locales peuvent porter le nom de variables globales. Enfin, on peut utiliser, pour une variable locale dans une procédure, le nom d'une variable locale déjà utilisé dans une autre procédure.
Une procédure étant un sous-programme complet, elle peut contenir ses propres procédures et fonctions, qui ne sont accessibles que par leur procédure "mère", c'est à dire la procédure dans laquelle elles sont déclarées. Une sous-procédure ne peut appeler d'autres procédures ou fonctions que si ces dernières font partie du programme principal ou de la procédure qui la contient.
Syntaxe :

```
Procedure nom_de_procedure;
Var variable : type;

     Procedure nom_de_sous_procedure;
     Var variable : type;
     Begin
       ...
     End;

Begin
  ...
  ...
End;
```

Procédures paramétrées

On peut aussi créer des **procédures paramétrées**, c'est-à-dire qui reçoivent des variables comme paramètres. Le programme principal (ou une autre procédure qui aurait été déclarée après) affecte alors des valeurs de variables à la procédure en passant des variables en paramètres. Et ces valeurs s'incorporent dans les variables propres à la procédure.
La déclaration des paramètres se fait alors en même temps que la déclaration de la procédure; ces variables sont des **paramètres formels** car existant uniquement dans la procédure. Lorsque que le programme appelle la procédure et lui envoie des valeurs de type simple, celles-ci sont appelées **paramètres réels** car les variables sont définies dans le programme principal et ne sont pas valables dans la procédure.
A noter qu'on peut passer en paramètre directement des valeurs (nombre, chaînes de caractères...) aussi bien que des variables.

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

Syntaxe :

```
Program nom_de_programme;

Procedure nom_de_procedure ( noms_de_variables : types );
Begin
  ...
  ...
End;

BEGIN
  nom_de_procedure ( noms_d_autres_variables_ou_valeurs );
END.
```

Note : on peut passer en paramètre à une procédure des types simples et structurés. Attention néanmoins à déclarer des types spécifiques de tableau à l'aide du mot-clé Type (voir **chapitre 20** sur les types simples et structurés).
Exemple :

```
Program Exemple9b;

Uses Crt;

Procedure Maths (Param : Real);
Begin
  WriteLn('Procédure de calcul. Veuillez patienter.');
  Param := Sin(Sqrt(Param + 10));
  WriteLn(Param);
End;

Var Nombre : Real;

BEGIN
  ClrScr;
  Write('Entrez un nombre :');
  ReadLn(Nombre);
  Maths(Nombre);
  ReadLn;
END.
```

Ce programme *Exemple9b* appelle une procédure paramétrée appelée Maths qui effectue les mêmes calculs que dans le programme *Exemple9a*. Mais ici, la variable est déclarée après la procédure paramétrée. Donc, la procédure ne connaît pas l'existence de la variable nombre; ainsi, pour qu'elle puisse l'utiliser, il faut la lui passer en paramètre !

Le mot-clé Var

Il est quelquefois nécessaire d'appeler une procédure paramétrée sans pour autant avoir de valeur à lui passer mais on souhaiterait que ce soit elle qui nous renvoie des valeurs (exemple typique d'une procédure de saisie de valeurs par l'utilisateur) ou alors on désire que la procédure puisse modifier la valeur de la variable passée en paramètre. Les syntaxes précédentes ne conviennent pas à ce cas spécial. Lors de la déclaration de paramètre au sein de la procédure paramétrée, le mot-clé 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. Et lorsque Var n'est pas là, les paramètres formels doivent impérativement avoir une valeur lors de l'appel de la procédure.
Pour expliquer autrement, si Var n'est pas là, la variable qu'on envoie en paramètre à la procédure doit absolument déjà avoir une valeur (valeur nulle acceptée). De plus, sans Var, la variable (à l'intérieur de la procédure) qui contient la valeur de la variable passée en paramètre, même si elle change de valeur, n'aura aucun effet sur la valeur de la variable (du programme principal) passée en paramètre. C'est à dire que la variable de la procédure n'existe qu'à l'intérieur de cette dernière et ne peut absolument pas affecter en quoi que ce soit la valeur initiale qui fut envoyée à la procédure : cette valeur initiale reste la même avant et après l'appel de la procédure. Car en effet, la variable de la procédure est dynamique : elle est créée lorsque la procédure est appelée et elle est détruite lorsque la procédure est finie, et ce, sans retour d'information vers le programme principal. La procédure paramétrée sans Var évolue sans

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

aucune interaction avec le programme principal (même si elle est capable d'appeler elle-même d'autres procédures et fonctions).
Par contre, si Var est là, la valeur de la variable globale passée en paramètre à la procédure va pouvoir changer (elle pourra donc ne rien contenir à l'origine). Si, au cours de la procédure, la valeur est changée (lors d'un calcul, d'une saisie de l'utilisateur...), alors la nouvelle valeur de la variable dans la procédure ira se placer dans la variable globale (du programme principal) qui avait été passée en paramètre à la procédure. Donc, si on veut passer une variable en paramètre dont la valeur dans le programme principal ne doit pas être modifiée (même si elle change dans la procédure), on n'utilise pas le Var. Et dans le cas contraire, si on veut de la valeur de la variable globale placée en paramètre change grâce à la procédure (saisie, calcul...), on colle un Var.

```
Program Exemple9c;

Uses Crt;

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(Shaine) ;
  Affichage(Chaine) ;
END.
```

Ce programme *Exemple9c* illustre l'utilisation du mot-clé Var. En effet, le programme principal appelle pour commencer une procédure paramétrée Saisie et lui passe la valeur de la variable Chaine. Au sein de la procédure paramétrée, cette variable porte un autre nom : Nom, et comme au début du programme cette variable n'a aucune valeur, on offre à la procédure la possibilité de modifier le contenu de la variable qu'on lui passe, c'est-à-dire ici d'y mettre le nom de l'utilisateur. Pour cela, on utilise le mot-clé Var. Lorsque cette procédure Saisie est terminée, la variable Chaine du programme principal a pris la valeur de la variable Nom de la procédure. Ensuite, on envoie à la procédure Affichage la valeur de la variable Chaine (c'est-à-dire ce que contenait la variable Nom, variable qui fut détruite lorsque la procédure Saisie se termina). Comme cette dernière procédure n'a pas pour objet de modifier la valeur de cette variable, on n'utilise pas le mot clé Var; ainsi, la valeur de la variable Chaine ne pourra pas être modifiée par la procédure. Par contre, même sans Var, la valeur de la variable Info pourrait varier au sein de la procédure si on le voulait mais cela n'aurait aucune influence sur la variable globale Chaine. Comme cette variable Info n'est définie que dans la procédure, elle n'existera plus quand la procédure sera terminée.
Il faut savoir qu'une procédure paramétrée peut accepter, si on le désire, plusieurs variables d'un même type et aussi plusieurs variables de types différents. Ainsi, certaines pourront être associées au Var, et d'autres pas. Si l'on déclare, dans une procédure paramétrée, plusieurs variables de même type dont les valeurs de certaines devront remplacer celles initiales, mais d'autres non; on pourra déclarer séparément (séparation par une virgule) les variables déclarées avec Var et les autres sans Var. Si on déclare plusieurs variables de types différents et qu'on veut que leurs changements de valeur affecte les variables globales, alors on devra placer devant chaque déclaration de types différents un Var.
Syntaxes :

Paramètres de même type

```
Procedure nom_de_procedure (Var var1, var2 : type1 ; var3 : type1);
Begin
  ...
End;
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

Paramètres de types différents

```
Procedure nom_de_procedure (Var var1 : type1 ; Var var2 : type2);
Begin
  ...
End;
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

Chapitre 8 - Fonctions

Contrairement aux procédures, les fonctions renvoient directement un résultat. Elles sont appelées à partir du programme principal, d'une procédure ou d'une autre fonction. Le programme affecte des valeurs à leurs paramètres (comme pour les procédures paramétrées, il faudra faire attention à l'ordre d'affectation des paramètres). Ces fonctions, après lancement, sont affectées elles-mêmes d'une valeur intrinsèque issue de leur fonctionnement interne. Il faut déclarer une fonction en lui donnant tout d'abord un identifiant (c'est-à-dire un nom d'appel), en déclarant les paramètres formels dont elle aura besoin et, enfin, il faudra préciser le type correspondant à la valeur que renverra la fonction (string, integer, etc.).

Une fonction ne peut pas renvoyer un type complexe (array, record) *(voir **chapitre 20** sur les types simples et complexes)*

De plus, comme le montrent les syntaxes suivantes, on peut fabriquer une fonction sans paramètre (ex : Random). Il ne faut surtout pas oublier, en fin (ou cours) de fonction, de donner une valeur à la fonction, c'est-à-dire d'affecter le contenu d'une variable ou le résultat d'une opération (ou autre...) à l'identifiant de la fonction (son nom), comme le montrent les syntaxes suivantes.
Syntaxes :

Fonction avec paramètre

```
Function nom_de_fonction (variable : type ) : type;
Var { déclaration de variables locales }
Begin
  ...
  nom_de_fonction := une_valeur;
End;
```

Fonction sans paramètre

```
Function nom_de_fonction : type;
Var { déclaration de variables locales }
Begin
  ...
  nom_de_fonction := une_valeur;
End;
```

Exemple :

```
Program Exemple10;

Uses Crt;

Function Puissance (i, j : Integer) : Integer;
Var Res, a : Integer ;
Begin
  Res := 1;
  for a := 1 to j do Res := Res * i;
  Puissance := Res;
End;

Var Resultat, x, n : Integer;

BEGIN
  Write('Entrez un nombre : ');
  ReadLn(x);
  Write('Entrez un exposant : ');
  ReadLn(n);
  Resultat := Puissance(x,n);
  WriteLn(Resultat);
  ReadLn;
END.
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

Chapitre 9 - Audio

Sound ... Delay ... NoSound

Pour faire du son, il faut indiquer la fréquence (f) en Hz et la durée (t) en ms.
Syntaxe :

```
Sound(f);
Delay(t);
NoSound;
```

Exemple :

```
Program Exemple11;

Uses Crt;

Var i, f : Integer;

BEGIN
  for i := 1 to 20 do
    begin
      for f := 500 to 1000 do Sound(f);
      Delay(10);
    end;
  Nosound;
END.
```

Chr(7)

La fonction Chr permet d'obtenir le caractère de la table ASCII correspondant au numéro. Il se trouve que les 31 premiers caractères sont des *caractères de contrôle* : beep, delete, insert, return, esc... Le caractère **7** correspond au **beep**.
Syntaxes :

```
Write(Chr(7));
Write(#7);
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

Chapitre 10 - Manipulation des fichiers

Déclaration

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

Les fichiers textes (Text)

Les fichiers textes (Text) sont écrits au format texte (chaînes de caractères, nombres) dans lesquels ont peut écrire et lire ligne par ligne ou à la file avec les procédures Write(Ln) et Read(Ln). Chaque fin de ligne du fichier se termine par les caractères 10 et 13 de la table ASCII (qui signifient respectivement **Retour ligne** (Line Feed, LF) = #10 et **Retour chariot** (carriage return, CR) = #13). Ces deux derniers caractères sont transparents au programmeur.
On pourra donc écrire ou lire indifféremment des chaînes ou des nombres, cela dépend du type que l'on affecte à la variable passée en paramètre aux procédures d'entrée/sortie (voir plus bas).

S'il y a lieu de faire une conversion nombre/chaîne, le compilateur le fait tout seul. Par contre, si le type de la variable ne correspond pas avec la donnée lue dans le fichier et qu'aucune conversion n'est possible alors cela produit une erreur.

Syntaxe :

```
Var f : Text;
```

Les fichiers typés (File of ...)

Les fichiers typés (File of) sont des fichiers dans lesquels les données sont écrites telles qu'elles se présentent en mémoire. On dit qu'elles sont écrites dans leur représentation binaire. La taille du fichier résultera directement et exactement de la taille en mémoire qu'occupe telle ou telle variable.
Cela accroît la vitesse d'accès aux données du fichier. Mais le plus grand avantage c'est que l'on obtient ainsi des fichiers parfaitement formatés, c'est-à-dire qu'on peut y lire et écrire directement des variables de type structuré qui contiennent plusieurs champs de données sans avoir à se soucier des divers champs qu'elles contiennent.
Syntaxe :

```
Var f : File of {type};
```

Exemple :

```
Var f : File of Integer;
```

Les fichiers simples (File)

Les fichiers... tout court ! (File) sont des fichiers dont on ne connaît pas le contenu. Ils peuvent servir à faire une simple copie d'un fichier dans un autre, pour stocker des données de différents types (en se donnant les moyens de retrouver ce que sont ces types lors de la lecture), pour sauver tout le contenu d'une zone mémoire pour analyse ultérieure, etc.
Syntaxe :

```
Var f : File;
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

Lecture et écriture

Avant de travailler sur un fichier, il faut le déclarer en lui affectant une variable qui servira à désigner le fichier tout au long du programme ou de la procédure dans laquelle il est utilisé. Assign s'applique à tous les types de fichiers.
Syntaxe :

```
Assign (variable_d_appel, nom_du_fichier);
```

Ensuite, il faut ouvrir le fichier.
Syntaxe :

```
Reset (variable_d_appel);
```

On ne peut pas écrire dans un fichier Text ouvert avec Reset !

Il est possible, pour le type File uniquement, de spécifier la taille de chaque bloc de donnée lu ou écrit sur le fichier, en rajoutant en argument à Reset une variable (ou un nombre directement) de type Word (entier) spécifiant cette taille en octet. Cela nécessite de connaître la taille mémoire de chaque type de variable (voir **chapitre 4** sur les différents types de variables) - la fonction SizeOf sert à cela. Par exemple, cette taille vaudra 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.
Syntaxe :

```
Reset (variable_d_appel, taille);
```

Pour créer un fichier qui n'existe pas ou bien pour en effacer son contenu, on emploie Rewrite qui permet d'effectuer des lectures (File of et File) et écritures (Text, File of et File).

On ne peut pas lire dans un fichier Text ouvert avec Rewrite !

Syntaxe :

```
Rewrite (variable_d_appel);
```

L'utilisation de Rewrite avec un fichier existant ouvre ce fichier, mais le vide !

Tout comme Reset, Rewrite permet de spécifier une taille aux échanges de données sur un File seulement (aussi bien en écriture qu'en lecture). Avec Rewrite, dans le cas où le fichier n'existe pas encore alors qu'avec Reset c'est dans le cas où il existe déjà.
Syntaxe :

```
Rewrite (variable_d_appel, taille);
```

Syntaxe	Lecture	Ecriture
Reset(f)	Text File of File	File of File
Rewrite(f)	File of	Text

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

	File	File of File
Reset(f,taille)	File	File
Rewrite(f,taille)	File	File

Pour lire le contenu d'une ligne d'un fichier Text ouvert, on utilise la même instruction qui permet de lire la valeur d'une variable au clavier à savoir ReadLn. (2) Sera alors lue, la ou les variable(s) correspondant au contenu de la ligne courante (celle pointée par le pointeur). Si la ou les variable(s) n'étai(en)t pas de taille suffisante pour contenir toutes les données de la ligne, alors l'excédent serait perdu. C'est même un peu plus compliqué que ça : si on lit une variable de type String, la lecture se poursuivra jusqu'à la fin de la ligne en cours, ou jusqu'à la fin du fichier si elle survient avant, et cela quelle que soit la longueur maximale possible pour la chaine. Il faut donc faire très attention.
Syntaxes :

```
ReadLn (variable_d_appel, variable);
ReadLn (variable_d_appel, var1, var2, ... varN); { Attention, pas de Strings (sauf la dernière) }
```

Pour écrire dans un fichier Text, il suffit d'employer la procédure WriteLn. (3)
Syntaxes :

```
WriteLn (variable d_appel, variable);
WriteLn (variable_d_appel, var1, var2, ... varN); { Attention : voir la remarque ci-après }
```

Il faut éviter cette seconde syntaxe, à moins de préciser, pour chaque variable à écrire, la **largeur minimum** à utiliser, en prenant soin que pour tous les cas possibles cette largeur minimum garantisse qu'il y aura au moins un espace pour séparer les données de chaque variable, sinon on ne pourra pas relire correctement les données.
Pour lire et écrire sur un File, il faut utiliser les procédures BlockRead et BlockWrite.
Syntaxes :

```
BlockRead (f, variable, nbr);
BlockRead (f, variable, nbr, result);
BlockWrite (f, variable, nbr);
BlockWrite (f, variable, nbr, result);
```

BlockRead lit sur le fichier f de type File une variable qui contiendra un nombre de blocs mémoire (dont la taille est définie par Reset ou Rewrite) égale à nbr. La variable (facultative mais dont l'utilisation systématique est fortement conseillée) result prend pour valeur le nombre de blocs effectivement lus (car il peut y en avoir moins que prévu initialement).
BlockWrite écrit sur le fichier f de type File une variable sur un nombre de blocs mémoire égal à nbr. La variable (facultative mais dont l'utilisation systématique est fortement conseillée) result prend pour valeur le nombre de blocs effectivement écrits. Dans le cas ou cette variable result est ommise et qu'il est impossible d'écrire autant de blocs que voulu, il est généré une erreur d'exécution !

Les variables nbr et result doivent être de type Word.

Syntaxe	Types de fichiers associés
WriteLn	Text
ReadLn	Text
Write	Text

(2) On peut également utiliser Read, mais cela peut poser des problèmes car en arrivant à la fin de la ligne, le pointeur ne passera pas à la suivante si on n'utilise que des Read.
(3) On peut également utiliser Write, avec la même remarque que pour le Read.

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

	File of
Read	Text File of
BlocWrite	File
BlockRead	File

Fermeture

Il est impératif de fermer les fichiers ouverts avant de terminer le programme, sous peine de voir les données inscrites en son sein perdues.

Syntaxe :

```
Close (variable_d_appel);
```

Il est possible de rouvrir un fichier Text en cours de programme même s'il a déjà été refermé, grâce à sa variable d'appel. La position courante du curseur sera alors positionnée à la fin du fichier. Cela ne fonctionne qu'en écriture et qu'avec un Text.
Syntaxe :

```
Append (variable_d_appel);
```

Exemple :

```
Program Exemple12;

Uses
  Crt, Dos;

Var
  f : Text;
  Nom : String;
  Choix : Char;

Procedure Lecture;
Begin
  Append(f);
  Readln(f,nom);
  Writeln(nom);
End;

BEGIN
  Clrscr;
  Assign(f,'init.txt');
  Rewrite(f);
  Write('Entrez un nom d''utilisateur : ');
  Readln(Nom);
  Nom := 'Dernier utilisateur : ' + Nom;
  Writeln(f,Nom) ;
  Close(f);
  Write('Voulez-vous lire le fichier init.txt ? [O/N] ');
  Choix := ReadKey;
  if (UpCase(Choix) = 'O') then Lecture;
END.
```

Ce programme *Exemple12* illustre les principales commandes qui permettent de travailler sur des fichiers de type texte. Ici, le programme réinitialise à chaque lancement le fichier *init.txt* et y inscrit une valeur entrée par l'utilisateur (son nom, en l'occurrence). Il permet également à l'utilisateur de lire le contenu du fichier (qui ne contient qu'une seule ligne de texte).

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

Fonctions supplémentaires

Gestion du curseur

Il est possible de déplacer à volonté le curseur en spécifiant à la procédure suivante le fichier de type File of ou File ainsi que le numéro de l'élément (le premier à pour numéro :"0", le second : "1", le troisième : "2", etc...) où l'on veut mettre le curseur. Cela s'appelle l'**accès direct** à un fichier contrairement à l'**accès séquentiel** qui consiste à parcourir toutes les informations précédant celle qu'on cherche.
Note : Seek n'est pas utilisable avec des Text.
Syntaxe :

```
Seek (variable_d_appel, position);
```

Exemple :

```
Program Exemple13 ;
Uses
  Crt, Dos;
Var
  f : Text;
  s : String ;
BEGIN
  Assign(f,'c:\autoexec.bat');
  Reset(f);
  Writeln('Affichage du contenu du fichier AUTOEXEC.BAT : ');
  while not Eof(f) do
    begin
      ReadLn(f,s);
      WriteLn(s);
    end;
END.
```

Ce programme *Exemple13* permet de lire un fichier texte et d'en afficher le contenu à l'écran. La fonction Eof permet de vérifier si le pointeur arrive en fin de fichier (elle aura alors la valeur true).
Il est possible de connaître la taille d'un fichier en octets, sauf lorsque celui-ci est déclaré en Text.
Syntaxe :

```
FileSize (variable_d_appel);
```

Il est possible de connaître la position du pointeur, en octets, dans le fichier lorsque celui-ci est déclaré en File of Byte. La fonction suivante renvoie une valeur de type LongInt.
Syntaxe :

```
FilePos (variable_d_appel);
```

Exemple :

```
Program Exemple14;
Var
  f : File of Byte;
  Taille : LongInt;
BEGIN
  Assign(f,'c:\autoexec.bat');
  Reset(f);
  Taille := FileSize(f);
  Writeln('Taille du fichier en octets :',Taille);
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

```
  Writeln('Déplacement du curseur...');
  Seek(f,Taille div 2);
  Writeln('Le pointeur se trouve à l''octet : ',FilePos(f));
  Close(f);
END.
```

Le programme *Exemple14* déclare le fichier autoexec.bat comme File of Byte et non plus comme Text, puisqu'on ne désire plus écrire ou lire du texte dedans mais seulement connaître sa taille et accessoirement faire mumuse avec le pointeur.

Fin de ligne, fin de fichier

Il est possible de savoir si, lors de la lecture d'un fichier, on se trouve ou non en fin de ligne ou de fichier, grâce aux fonctions suivantes qui renvoient une valeur de type Boolean.
Syntaxe :

```
Var f : Text;
Eof(f); { Pointeur en fin de fichier }
SeekEoLn(f); { Pointeur en fin de ligne }
```

Autre syntaxe :

```
Var f : File;
EoLn (f); { Pointeur en fin de ligne }
SeekEof (f); { Pointeur en fin de fichier }
```

Exemple :

```
Program Exemple15 ;
Var
  f : Text;
  i, j : String;
BEGIN
  Assign(f,'c:\autoexec.bat');
  Reset(f);
  while not SeekEof (f) do
    begin
      if SeekEoln(f) then ReadLn(f);
      Read(f,j);
      Writeln(j);
    end;
END.
```

Effacer un fichier

On peut également effacer un fichier préalablement fermé.
Syntaxe:

```
Erase (f);
```

Renommer un fichier

On peut aussi renommer un fichier (qui ne doit pas être ouvert).
Syntaxe :

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

```
Rename (f, NouveauNom);
```

Tronquer un fichier

Il est possible de tronquer un fichier, c'est-à-dire de supprimer tout ce qui se trouve après la position courante du pointeur.
Syntaxe :

```
Truncate (f);
```

Gestion des erreurs

Dans tous les exemples de manipulation de fichiers donnés jusqu'ici, il n'a pas été tenu compte (pour des raisons de clarté, pour présenter les différentes notions) des éventuelles erreurs qui auraient pu survenir.
En effet, si on laisse les choses en l'état, la moindre erreur de création, d'ouverture, de lecture, d'écriture de fichier provoquera une terminaison immédiate et inconditionnelle du programme. Or, il y a moyen d'intercepter proprement toutes ces erreurs et d'offrir à l'utilisateur un produit mieux fini.
La première chose à faire est de donner une **directive** au compilateur, lui demandant non pas de mettre fin au programme en cas d'erreur, mais plutôt d'aller indiquer un code d'erreur dans une variable interne que le programme pourra consulter. Cette directive est

```
{$I-}
```

Après chaque exécution d'une routine de gestion de fichier, le programme pourra aller tester la valeur de cette variable interne, au moyen de la fonction IOResult.
Exemple : une procédure de copie de fichier :

```
Function CopyFile (const SourceFile, DestFile : String) : Boolean;
Var
  Source, Dest : File;
  Buffer : Array [0..1023] of Byte;
  Count : Word;
  OldFileMode : Word;
Begin
  CopyFile := True;
  { Désactivation des erreurs E/S }
  {$I-}

  { Assignation des fichiers }
  Assign(Source,SourceFile);
  Assign(Dest,DestFile);

  { Ouverture en lecture seule de Source et création de Dest }
  OldFileMode := FileMode;
  FileMode := 0;
  Reset(Source,1);
  if IOResult = 0
     then
       begin
         Rewrite(Dest,1);
         if IOResult = 0
            then
              begin
                { Boucle principale de copie, s'arrête quand la fin du fichier est atteinte }
                repeat
                  { Remplissage du buffer : 1 Ko prévus, Count octets réels }
                  BlockRead(Source,Buffer,SizeOf(Buffer),Count);
                  { Ecriture du contenu du buffer }
                  BlockWrite(Dest,Buffer,Count);
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

```
                until (Count = 0) or (Count <> SizeOf(Buffer));
                { Fermeture du fichier }
                Close(Dest);
                if IOResult <> 0
                   then   { Erreur de fermeture du nouveau fichier }
                     CopyFile := False;
              end
            else   { Erreur de création du nouveau fichier }
              CopyFile := False;
          { Fermeture du fichier }
          Close(Source);
        end
      else   { Erreur d'ouverture du fichier original }
        CopyFile := False;
  { Réactivation des erreurs d'E/S et rétablissement du mode de lecture }
  FileMode := OldFileMode;
  {$I+}
End;
```

Notez que le mode normal de gestion des erreurs est restauré à la fin de la procédure, avec la directive

```
{$I+}
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

Chapitre 11 - Graphismes

Les instructions qui vont suivre nécessitent l'unité Graph. Pour créer des graphismes, il faut tout d'abord initialiser le mode graphique à l'aide de InitGraph. Par exemple, pour un écran VGA 640x480x16 :

```
InitGraph (VGA,VGAHi,'c:\bp\bgi');
```

Attention : le chemin du répertoire BGI peut changer d'une machine à l'autre.
Ici, la valeur VGA correspond au pilote graphique, VGAHi au mode graphique; on obtient en général une résolution de 640x480 pour 16 couleurs. En fin de programme, afin de retourner en mode texte, il faut exécuter la procédure CloseGraph.
Pour une **autodétection** du mode graphique maximal supporté par le système, si celui-ci est inférieur à 640x480x16 (qui est la résolution maximale de l'unité Graph de Turbo Pascal 7.0), initialisez l'unité Graph de la manière suivante :

```
Uses Graph;
Var Pilote, Mode : Integer;
BEGIN
  Pilote := Detect;
  InitGraph(Pilote, Mode, 'c:\turbo7\bgi');
  ...
  CloseGraph;
END.
```

Constantes de pilote et de mode graphique :

Pilote	Valeur
CGA	1
EGA	9
VGA	3

Résolution	Valeur
640x200	0
640x320	1
640x480	2

Ayez toujours en tête que la résolution de l'écran graphique, en Turbo Pascal, est au maximum de 640x480 pixels et de 16 couleurs.

A noter que l'origine de l'écran graphique se trouve en haut à gauche de l'écran, c'est-à-dire que le point de coordonnées (0,0) est le premier pixel de l'écran; ainsi, le point à l'opposé qui est de coordonnées (639, 479) se trouve en bas à droite.
Pour un affichage de meilleure résolution, fabriquez vous-même une unité spécifique (voir **chapitre 14** sur les unités). Souvent, dans ce genre d'unité censée permettre de faire plus que permis avec les unités de base, le code est en Assembleur...

```
SetColor (couleur);
```

Cette instruction permet de sélectionner une couleur (voir tableau ci-après) qui affectera toutes les commandes graphiques. Il vous suffit d'entrer en paramètre la couleur de votre choix.

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

Valeur	Constante	Couleur
0	Black	Noir
1	Blue	Bleu
2	Green	Vert foncé
3	Cyan	Cyan foncé
4	Red	Rouge
5	Magenta	Mauve foncé
6	Brown	Marron
7	LightGray	Gris clair
8	DarkGray	Gris foncé
9	LightBlue	Bleu flou
10	LightGreen	Vert clair
11	LightCyan	Cyan clair
12	LightRed	Rouge clair
13	LightMagenta	Mauve clair
14	Yellow	Jaune
15	White	Blanc

```
SetFillStyle (style,couleur);
```

Sélectionne un motif de remplissage spécifique (voir tableau ci-après) ainsi qu'une couleur parmi 16. Ces paramètres ne seront utilisés que par quelques instructions dont celle qui suit (Bar). Il vous suffit d'entrer en paramètre le motif de votre choix.

Valeur	Constante	Rendu
0	EmptyFill	Couleur du fond
1	SolidFill	Couleur du tracé
2	LineFill	Lignes horizontales
3	LtSlashFill	Barres obliques (/)
4	SlashFill	Barres obliques (/) épaisses
5	BkSlashFill	Antibarres (\) épaisses
6	LtBkSlashFill	Antibarres (\)
7	HatchFill	Hachures fines
8	XHatchFill	Hachures épaisses
9	InterLeaveFill	Croisillons
10	WideDotFill	Points aérés
11	CloseDotFill	Points serrés
12	UsesFill	Motif défini par le programmeur

```
FloodFill (x,y,border);
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

Remplit une surface fermée identifiée par sa couleur de bordure : border à partir du point de coordonnées (x,y). La couleur de remplissage sera celle choisie par un SetColor ou un SetFillStyle.

```
Bar (x1,y1,x2,y2);
```

Trace un rectangle plein aux coordonnées indiquées.

```
Bar3d (x1,y1,x2,y2,z,TopOn);
```

Trace un parallélépipède aux coordonnées indiquées et de profondeur z. TopOn est une constante signifiant que les lignes de perspectives supérieures sont activées (pour les cacher : TopOff).

```
SetLineStyle (style,$c3,épaisseur);
```

Sélectionne un style et une épaisseur de line utilisés par les instructions graphiques de base : Line, Rectangle, Circle (seulement l'épaisseur). Voir les tableaux suivants pour les valeurs possibles. Il vous suffit d'entrer en paramètre le style et l'épaisseur de votre choix.

Valeur	Constante	Rendu
0	SolidLn	Ligne pleine
1	DottedLn	Ligne pointillée
2	CenterLn	Ligne mixte
3	DashedLn	Ligne tiretée
4	UseBitLn	Motif défini par le programmeur

Valeur	Constante	Rendu
1	NormWidth	Trait normal
3	ThickWidth	Trait épais

```
Line (x1,y1,x2,y2);
```

Trace une ligne débutant au point de coordonnées (x1,y1) et se terminant au point de coordonnées (x2,y2).

```
LineTo (x,y);
```

Trace une ligne à partir de la position courante du curseur graphique, jusqu'aux coordonnées x et y spécifiées.

```
LineRel (deltaX,deltaY);
```

Trace une ligne à partir de la position courante du curseur graphique, jusqu'aux coordonnées calculées à partir des pas de variation deltaX et deltaY spécifiés.
A noter qu'il peut y avoir équivalence entre diverses combinaisons des troix dernières instructions.

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

```
MoveTo (x,y);
```

Positionne le pointeur graphique aux coordonnées x et y spécifiées.

```
MoveRel (deltaX,deltaY);
```

Déplace relativement le pointeur graphique depuis sa position courante jusqu'aux coordonnées calculées à partir des pas de variation deltaX et deltaY.

```
Rectangle (x1,y1,x2,y2);
```

Trace un rectangle de coin haut-gauche de coordonnées (x1,y1) et de coin bas-droite de coordonnées (x2,y2).

```
Circle (x,y,rayon );
```

Trace un cercle de coordonnées et de rayon spécifiés.

```
Ellipse (x,y,angle1,angle2,axe1,axe2);
```

Trace une ellipse de centre (x,y) de largeur axe1 et de hauteur axe2. On peut ne tracer qu'une partie de l'ellipse en spécifiant l'angle de départ angle1 et l'angle d'arrivée angle2, exprimés en degrés et dans le sens trigonométrique. Pour tracer une ellipse complète : angle1 = 0 et angle2 = 360.

```
Arc (x,y,angle1,angle2,rayon);
```

Trace un arc de cercle de centre (x,y) et de rayon rayon. On peut tracer un arc en spécifiant l'angle de départ angle1 et l'angle d'arrivée angle2, exprimés en degrés et dans le sens trigonométrique. Pour tracer un arc maximum, c'est-à-dire un cercle : angle1 = 0 et angle2 = 360.

```
Program Coquille;
Uses Graph;
Var r, a, Mode, Pilote : Integer;
BEGIN
  Mode := Detect ;
  InitGraph(Pilote,Mode,'c:\bp\bgi');
  SetColor(Yellow);
  r := 0;
  repeat
    Inc(r,8);
    Arc(GetMaxX Div 3,GetMaxY Div 2,0,r,180 - (r div 2));
  until r > 360;
  ReadLn;
  CloseGraph;
END.
```

Ce programme dessine à l'écran une série d'arcs de cercles afin de former la coquille d'un mollusque. Ce programme montre également comment il est simple d'introduire des fonctions et des opérateurs au sein d'une ligne de commande de procédure paramétrée (Arc();).

```
GetPixel (x,y);
```

Retourne la couleur du pixel aux coordonnées considérées.

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

```
PutPixel (x,x,couleur);
```

Affiche un pixel de couleur choisie aux coordonnées spécifiées.

```
GetMaxX;
```

Retourne la valeur maximale d'une coordonnée sur l'axe des x.

```
GetMaxY;
```

Retourne la valeur maximale d'une coordonnée sur l'axe des y.

```
GetMaxColor;
```

Retourne le nombre total de couleurs disponibles.

```
GetMaxMode;
```

Retourne la plus grande valeur du mode graphique supporté par le pilote courant.

```
SetTextStyle (fonte,orientation,taille );
```

Définit les paramètres d'affichage du texte qui suivra (avec la commande suivante). Il vous suffit d'entrer en paramètres la fonte et l'orientation voulues.
Note : Il est possible d'utiliser un très grand nombre de fontes supplémentaires, sous la forme de fichiers au format CHR qui doivent être placés dans le répertoire /BIN de Turbo Pascal 7.0.

Valeur	Fonte
0	DefaultFont
1	TriplexFont
2	SmallFont
3	SansSerifFont
4	GothicFont

Valeur	Constante	Signification
0	HorizDir	Orientation horizontale
1	VertDir	Orientation verticale

```
OutText (chaine);
```

Affiche du texte à la position actuelle du curseur.

```
OutTextXY (x,y,chaine);
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

Permet d'afficher du texte aux coordonnées voulues.

```
ClearDevice;
ClearViewPort;
```

Effacent le contenu de l'écran graphique.

```
CloseGraph;
RestoreCrtMode;
```

Ferment le mode graphique pour retourner à l'affichage MS-DOS habituel.

```
SetGraphMode (GetGraphMode);
```

Retourne au mode graphique de la procédure initialisation.

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

Chapitre 12 - Affichage en mode texte

En règle générale, les programmes dialoguent avec l'utilisateur : entrées et sorties de données respectivement avec Read(Ln) et Write(Ln). La nécessité pratique ou la volonté de présenter une interface plus conviviale imposent l'utilisation d'instructions spécifiques : effacer une ligne d'écran, changer la couleur des lettres… Ce chapitre énumère la quasi-totalité des instructions en Pascal vous permettant de gérer l'affichage en mode texte.

```
ClrScr;
```

Efface tout l'écran et place le curseur en haut à gauche de l'écran; souvent utilisé au démarrage d'un programme.

```
DelLine;
```

Efface la ligne courante, c'est-à-dire celle qui contient le curseur.

```
InsLine;
```

Insère une ligne vide à la position courante du curseur.

```
ClrEol;
```

Efface la fin d'une ligne à l'écran, à partir de la position courante du curseur. Note : la position du curseur n'est pas modifiée.

La résolution par défaut en mode texte, en Turbo Pascal, est de 80 colonnes par 25 lignes et de 16 couleurs.

```
TextBackground (n);
```

Choix d'une couleur de fond pour le texte qui sera affiché par la suite. n est le numéro (entre Black et LightGray) de la couleur; il est tout à fait possible d'y mettre une variable de type Byte à la place de x. Pour la liste des couleurs, voir le **chapitre 10** sur le graphisme.

```
TextColor (n);
```

Choix d'une couleur pour le texte qui sera affiché par la suite.

```
TextColor (n + Blink);
```

Choix d'une couleur pour le texte qui sera affiché en mode clignotant.

Le clignotement ne fonctionnera que sous un vrai MS-DOS, pas dans une console Windows.

```
Window (x1,y1,x2,y2);
```

Pour définir une fenêtre à l'écran. Cette fenêtre ainsi définie devient la fenêtre courante, et toutes les coordonnées écran (sauf les coordonnées de définition de la fenêtre) sont relatives à la fenêtre courante.

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

(x1,y1) désignent les coordonnées du coin supérieur gauche de la fenêtre, (x2,y2) les coordonnées du coin inférieur droit.

```
GotoXY (x,y);
```

Positionne le curseur à la position voulue dans l'écran ou dans une fenêtre définie par Window.
x et y sont respectivement le numéro de colonne et le numéro de ligne (axes des abscisses et des ordonnées).

```
WhereX;
WhereY;
```

Renvoient respectivement l'abscisse et l'ordonnée courantes du curseur.

```
HighVideo;
```

Sélectionne le mode haute densité des caractères. C'est-à-dire que la couleur sélectionnée pour l'affichage du texte est modifiée en son homologue plus vive dans la liste des 16 couleurs.

```
LowVideo;
```

Au contraire, sélectionne le mode faible densité de la couleur des caractères. C'est-à-dire que la couleur sélectionnée pour l'affichage du texte est modifiée en son homologue moins vive dans la liste des couleurs.

```
NormVideo;
```

Permet de revenir au mode normal de couleur de texte, pour pouvoir utiliser indifféremment les couleurs vives et ternes.

```
TextMode (n);
```

Sélectionne un mode spécifique d'affichage du texte.
Selon le mode, le nombre de caractères par ligne, le nombre de lignes et le nombre de couleurs changent.

Constante	Valeur	Mode
BW40	0	40x25 monochrome
CO40	1	40x25 couleur
BW80	2	80x25 monochrome
CO80	3	80x25 couleur
Mono	7	80x25 monochrome
Font8x8	256	80x43 ou 80x50 couleur

```
LastMode;
```

Permet de revenir au mode texte antérieur au dernier appel de TextMode.

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

Chapitre 13 - Caractères et chaînes de caractères

Caractères

Un caractère est une variable de type Char qui prend 1 octet (= 8 bits) en mémoire. La table ASCII est un tableau de 256 caractères, numérotés de 0 à 255, où les 23 premiers sont des caractères de contrôle, associés à des fonctions de base de MS-DOS (*Suppr*, *End*, *Inser*, *Enter*, *Esc*, *Tab*, *Shift*...) et tous les autres sont directement affichables (lettres, ponctuations, symboles, caractères graphiques).
Dans un programme en Pascal, on peut travailler sur un caractère à partir de son numéro dans la table ASCII (avec la fonction Chr(n) ou #n) ou directement avec sa représentation entre apostrophes '.
Exemples :

```
Espace := ' ';
Lettre := #80;
Carac := Chr(102);
```

Table ASCII :

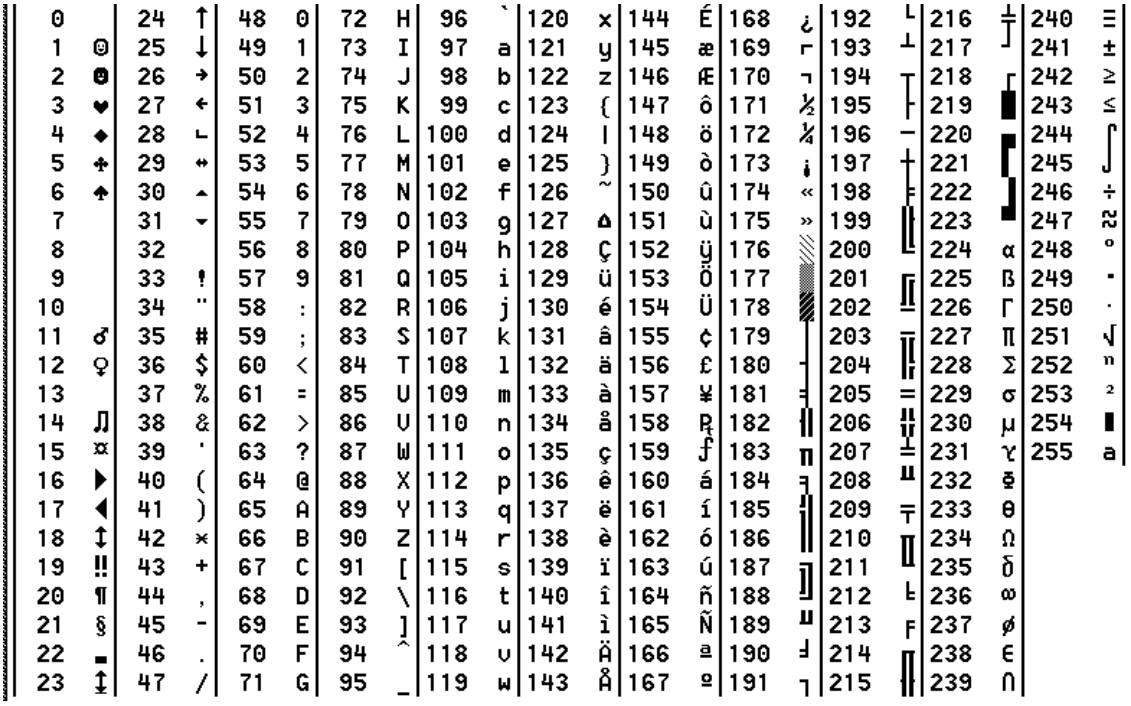

Table ASCII

Syntaxe :

```
Var Lettre : Char;
```

Lorsqu'on donne une valeur à une variable de type Char, celle-ci doit être entre apostrophes. On peut aussi utiliser les fonctions Chr et Ord ou même une variable String (voir plus loin) dont on prend un caractère à une position déterminée.
Syntaxe :

```
Lettre := Chaine[Position];
```

Voyons à présent quelques fonctions :

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

```
Upcase (k);
```

Cette fonction convertit un caractère minuscule en MAJUSCULE.
Exemple :

```
for i := 1 to Lenght(s) do
  s[i] := Upcase(s[i]);
```

```
Chr (n);
```

Cette fonction renvoie le caractère d'indice n dans la table ASCII.
Exemple :

```
k := Chr(64);
```

```
Ord (k);
```

Cette fonction renvoie l'indice (en byte) correspondant au caractère k dans la table ASCII. C'est la fonction réciproque de Chr.
Exemple :

```
i := Ord('M');
```

Chaînes de caractères

Le type String définit des variables "chaînes de caractères" ayant au maximum 255 signes, ces derniers appartenant à la table ASCII. On peut cependant définir des chaînes dont la longueur maximale sera moindre (de 1 à 255). Le premier caractère de la chaîne a pour indice 1, le dernier a pour indice 255 (ou moins si spécifié lors de la déclaration).
Syntaxe :

```
Var
  Chaine : String;
  Telephone : String[10];   { Chaîne d'une longueur maximale de 10 caractères }
```

Lorsqu'une valeur est affectée à une variable chaîne de caractères, on procède comme pour un nombre mais cette valeur doit être entre apostrophes. Si cette valeur contient une apostrophe, celle-ci doit être **doublée** dans votre code.
Exemple :

```
Animal := 'l''abeille';
```

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 dont la valeur numérique correspond à la longueur de la chaîne.

Il est donc possible, une chaîne de caractères étant un tableau, de modifier un seul caractère de la chaîne grâce à la syntaxe suivante :

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

```
Chaine[Index] := Lettre;
```

Exemple :

```
Program Exemple14;
Var
  Nom : String;
BEGIN
  Nom := 'Etiévant';
  Nom[2] := 'Z';
  Nom[0] := Chr(4);
  WriteLn(Nom);
  Nom[0] := Chr(28);
  Write(Nom,'-tagada');
END.
```

L'exemple *Exemple14* remplace la deuxième lettre de la variable Nom en un "Z" majuscule, puis spécifie que la variable ne contient plus que 4 caractères. Ainsi la valeur de la variable Nom est devenue : ***EZié***. Mais après, on dit que la variable Nom a une longueur de 28 caractères et on s'aperçoit à l'écran que les caractères de rang supérieur à 4 ont été conservés ! Ce qui veut dire que la chaîne affichée n'est pas toujours la valeur totale de la chaîne réelle en mémoire.
Attention cependant aux chaînes déclarées de longueur spécifiée (voir **chapitre 20** sur les types simples et structurés. Exemple : Type nom:String[20];) dont la longueur ne doit pas dépasser celle déclarée en début de programme.

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

Cette fonction concatène les chaînes de caractères spécifiées s1, s2, etc en une seule et même chaîne. On peut se passer de cette fonction grâce à l'opérateur + : s1 + s2 + s3 + ... + sn.
Exemples :

```
s := Concat (s1,s2);
s := s1 + s2;
```

```
Copy (s,i,j);
```

Cette fonction 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);
```

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

```
Insert (s1,s2,i);
```

Cette procédure insère la chaîne s1 dans la chaîne s2 à la position i.

```
Pos (s1,s2);
```

Cette fonction renvoie, sous forme de variable de type byte, la position de la chaîne s1 dans la chaîne s2. Si la chaîne s1 en est absente, alors cette fonction renvoie 0.

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

```
Str (n,s);
```

Cette procédure convertit la variable numérique n en sa représentation chaîne telle qu'elle serait affichée par Write(Ln).

```
Val (s,n,Error);
```

Cette procédure convertit la chaîne de caractères s en un nombre (de type numérique simple) n et renvoie un code d'erreur error (de type integer) qui est égal à 0 si la conversion est possible.

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

Chapitre 14 - Créer ses propres unités

Lorsque vous créez un programme en Turbo Pascal, vous utilisez nécessairement un certain nombre d'instructions (procédures ou fonctions) qui sont définies dans des unités extérieures au programme. Même lorsque vous ne spécifiez aucune unité par la clause Uses, l'unité System est automatiquement associée au programme (inscrite dans l'exécutable compilé). Quant aux autres unités fournies avec Turbo Pascal : Crt, Dos, Graph, Printer, etc, elles contiennent des instructions spécifiques qui ne pourront être appelées depuis le programme que si les unités correspondantes sont déclarées par la clause Uses. Le but de ce chapitre est d'apprendre à fabriquer de ses propres mains une unité qui pourra être appelée depuis un programme écrit en Turbo Pascal. Précisons qu'une unité s'écrit dans un fichier au format **PAS**. Mais une fois écrite, l'unité doit impérativement être compilée (au format **TPU**) pour être utilisable plus tard par un programme.
Un programme en Pascal débute par la déclaration du nom de programme comme suit :

```
Program nom_du_programme;
```

De manière analogue, une unité doit être (impérativement) déclarée comme suit :

```
Unit nom_de_l_unite;
```

Ensuite vient une partie déclarative très spéciale qui catalogue le contenu de l'unité visible par les programmes et/ou unités désirant utiliser la notre. Cette partie est très similaire à celle d'un programme, puisqu'on y déclare les constantes, les variables, les procédures, fonctions... et la liste des unités utilisées. Cette partie déclarative obligatoire s'écrit selon la syntaxe suivante :

```
INTERFACE
Uses ...;
Const ...;
Var ...;
Procedure ...;
Function ...;
```

Les déclarations des procédures / fonctions dans cette partie Interface ne concernent que leur prototype, c'est-à-dire qu'il ne faut pas y mettre le cors de ces procédures / fonctions.
Passons aux choses sérieuses, il faut passer à la partie la plus technique, c'est-à-dire écrire le code fonctionnel : les procédures et/ou fonctions qui seront appelées par le programme principal.
Exemple :

```
IMPLEMENTATION
Function Tangente (a : Real) : Real;
Begin
  Tangente := Sin(a) / Cos(a);
End;
```

Et la touche finale : un bloc Begin ... End., destiné à inclure du code qui doit s'exécuter au démarrage du programme afin, par exemple, d'initialiser des variables de l'unité. Ce bloc final peut très bien ne rien contenir.

Ces quatre parties doivent toutes impérativement apparaître dans le code.

Vous pouvez **télécharger l'unité TANGT.PAS** :

```
Unit Tangt;

INTERFACE
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

```
Var a : Real;

Function Tangente (a : Real) : Real;

IMPLEMENTATION

Function Tangente (a : Real) : Real;
Begin
  Tangente := Sin(a) / Cos(a) ;
End;

BEGIN
END.
```

Et **le programme TAN.PAS** :

```
Program Tan;

Uses Tang ;

Var x : Real ;

BEGIN
  Write('Entrez un angle en radians : ') ;
  ReadLn(x) ;
  WriteLn('Voici sa tangente : ',Tangente(2)) ;
END.
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

Chapitre 15 - Booléens et tables de vérité

Les booléens ont été inventés par Monsieur Boole dans le but de créer des fonctions de base, manipulant des valeurs logiques, qui, associées les unes aux autres, pourraient donner d'autres fonctions beaucoup plus complexes.
Les booléens (boolean en anglais et en Pascal) ne peuvent prendre que deux valeurs possibles : faux (false) ou vrai (true), et sont souvent codées en 0 ou 1. Ces valeurs sont analogues aux états possibles d'un interrupteur : ouvert ou fermé, d'une lampe : allumée ou éteinte.
En bref, les booléens ne sont utiles que pour connaître un état : vrai ou faux et en général pour caractériser si une condition est vraie ou fausse. Vous les utilisez déjà sans toujours le savoir dans les blocs if, until et while : *si telle condition est vraie, alors...*, *...jusqu'à ce que la condition soit vraie*, *tant que la condition est vraie...*
Boole inventa une algèbre qui porte son nom : l'algèbre de Boole. C'est cette dernière qui nous permet de faire des opérations sur les booléens grâce à des opérateurs (voir **chapitre 2** sur les opérateurs) : NOT (non), OR (ou), AND (et), XOR (ou exclusif), NAND (inverse de et), NOR (inverse de ou). En Turbo Pascal 7.0 n'existent que les opérateurs NOT, OR, AND et XOR, qui suffisent (en les combinant) à retrouver les autres. Ainsi, NAND = NOT(AND) et NOR = NOT(OR). Les tables de vérité des opérateurs logiques disponibles sur Turbo Pascal 7.0 se trouvent en fin de chapitre.
Syntaxe :

```
Var nom_de_variable : Boolean;
```

Pour donner une valeur à une variable booléenne, on procède comme pour tout autre type de variable, à l'aide de l'opérateur d'assignation := .
Syntaxes :

```
nom_de_variable := true;
nom_de_variable := false;
```

Pour assigner à une variable le résultat d'une condition, on peut remplacer la syntaxe une structure IF :

```
if condition then nom_de_variable_booleenne := true;
```

par une syntaxe bien plus simple :

```
nom_de_variable := condition;
```

Exemple :

```
Test := (x > 100) and (u = 'coucou');
```

Dans une structure if, until ou while), on peut avantageusement utiliser une condition sans spécifier sa valeur qui sera alors prise par défaut égale à true. C'est-à-dire que si on ne précise pas la valeur d'une variable booléenne ou d'une condition dans une structure if, par exemple, le compilateur Turbo Pascal se dira systématiquement : si variable est true, alors faire... Il devient donc inutile de spécifier la valeur de la variable dans ce cas là.
Syntaxes :

```
if nom_de_variable_booleenne then instruction;
{ ou }
if condition then instruction;

repeat
  instructions
until nom_de_variable;
{ ou }
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

```
repeat
  instructions
until condition;

while nom_de_variable do instruction;
{ ou }
while condition do instruction;
```

Nous avons vu plus haut que les opérateurs spécifiques aux booléens (NOT, OR, AND, XOR) pouvaient se composer pour donner des expressions plus complexes. Il est bien entendu possible d'introduire dans ces expressions le opérateurs relationnels (=, <, >, <=, >=, <>) et, plus généralement, tous les autres opérateurs disponibles en Turbo Pascal. Vous pouvez même utiliser directement des expressions, qu'elles soient mathématiques ou non.
Exemples :

```
Test := (Length(u) <= 20) or ((sin(a) * pi) < x);
```

Ici, la variable booléenne test devient vraie si la longueur de la chaîne u n'excède pas 20 caractères ou si le sinus de l'angle a multiplié par la valeur de pi est strictement inférieur à la valeur de x.

```
if (a = 0) or ((b = 0) and (c = 0)) then Writeln('La lampe est allumée');
```

Ici, écriture à l'écran d'un message si a est nul ou si (b et c) sont simultanément nuls.
Note : si vous affichez à l'écran la valeur d'une variable booléenne, il s'affichera FALSE ou TRUE (selon sa valeur effective) en caractères majuscules.

Tables de vérité des opérateurs logiques

NOT

X	NOT X
false	true
true	false

AND

X	Y	X AND Y
false	false	false
false	true	false
true	false	true
true	true	true

OR

X	Y	X OR Y
false	false	false
false	true	true
true	false	true
true	true	true

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

XOR

X	Y	X XOR Y
false	false	false
false	true	true
true	false	true
true	true	false

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

Chapitre 16 - Gestion des dates et heures

Le système de l'ordinateur travaille avec l'horloge à quartz qui donne le tempo de calcul. Cette horloge possède sa propre date et heure qu'il est possible d'afficher ou de modifier. A partir du prompt MS-DOS, il suffit d'utiliser les commandes **Date** ou **Time**, mais en Turbo Pascal, c'est un peu plus délicat. En effet, en Pascal, il est nécessaire de déclarer un grand nombre de variables qu'il faut formater avant l'affichage.

Toutes les instructions qui suivent nécessitent l'unité Dos.

GetDate

```
GetDate (an, mois, jour, joursem);
```

Pour obtenir la date courante du système, avec :

- an : le numéro de l'année (compris entre 1980 et 2099),
- mois : le numéro du mois (1 à 12),
- jour : le numéro du jour dans le mois (1 à 31),
- joursem : le numéro du jour dans la semaine (0 à 6, le zéro correspondant au dimanche).

Ces variables sont déclarées en word (entiers positifs de 0 à 65535).

SetDate

```
SetDate (an, mois, jour);
```

Pour changer la date du système. Ici, les variables obéissent aux mêmes conditions décrites précédemment. Si une date entrée est invalide, alors elle ne sera pas prise en compte et la date restera inchangée.

GetTime

```
GetTime (heure, minute, seconde, centieme);
```

Pour obtenir l'heure courante, avec :

- heure : le numéro de l'heure (0 à 23),
- minute : le numéro de la minute (0 à 59),
- seconde : numéro de la seconde (0 à 59),
- centieme : les centièmes de seconde (0 à 99).

SetTime

```
SetTime (heure, minute, seconde, centieme);
```

Pour changer l'heure système. Les variables obéissant aux mêmes conditions décrites plus haut. Si une heure entrée est invalide, alors elle ne sera pas prise en compte et l'heure courante n'en sera aucunement affectée.

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

Exemple récapulatif sur la date et l'heure système

Vous pouvez télécharger le programme annoté et explicatif **DATE.PAS** :

```
Program Date;

{ * DECLARATION DE L'UNITE UTILISEE : }
Uses
  Dos; {unité utilisée}

{ * DECLARATION DES CONSTANTES UTILISEES : }
Const
  {tableau contenant les jours de la semaine}
  Jours : Array [0..6] of String[8] =
 ('dimanche','lundi','mardi','mercredi','jeudi','vendredi','samedi');

  {tableau contenant les mois de l'année}
  Mois : Array [0..11] of String[9] =
 ('décembre','janvier','février','mars','avril','mai','juin','juillet','août','septembre','octobre','novembre');

{ * DECLARATION DES VARIABLES UTILISEES : }
Var
  a, m, j, Jour, Heure, Min, Sec, Cent : Word;
  { a : numéro de l'année [1980..2099]
    m : numéro du mois [1..12]
    j : numéro du jour dans le mois [1..31]
    Jour : nom du jour dans la semaine [0..6]
    Heure : numéro de l'heure [0..23]
    Min : numéro de la minute [0..59]
    Sec : numéro de la seconde [0..59]
    Cent : numéro du centième de seconde [0..99] }

{ * DECLARATION DE LA FONCTION UTILISEE : }
Function Format (w : Word) : String;
{ Convertit un word en chaîne sur 2 caractères }
Var s : String;
Begin {début de la fonction}
  Str(w,s); {convertit w en chaîne}
  if Length(s) = 1 then s := '0' + s;   {toujours sur 2 caractères}
  Format := s; {la fonction Format prend la valeur de S}
End; {fin de la fonction}

BEGIN {début du programme principal}
  GetDate(a,m,j,Jour); {lecture de la date système}
  Write('Nous sommes le ',Jours[Jour],' ',j,' ',mois[m],' ',a);
  GetTime(Heure,Min,Sec,Cent); {lecture de l'heure système}
  WriteLn(' et il est',Format(Heure),':',Format(Min),':',Format(Sec),'.',Format(Cent));
END. {fin du programme}
```

GetFTime

```
GetFTime (f, Heure);
```

Pour obtenir la date de dernière modification de fichier. Ici, f est une variable d'appel de fichier (file, file of ... ou text) et Heure est une variable de type LongInt.

SetFTime

```
SetFTime (f, Heure);
```

Vous l'aurez deviné, c'est la réciproque de l'instruction GetFTime.

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

Unpacktime

```
UnpackTime (Heure, dt);
```

Une information obtenue avec l'instruction GetFTime est illisible sans avoir été décodée avec cette instruction, où Heure est la même variable que précédemment et dt est une variable de type DateTime. Ensuite pour pouvoir utiliser les informations contenues dans dt, il faut les sortir une par une : dt.Hour représente l'heure, dt.Min : les minutes et dt.Sec : les secondes.

Packtime

```
PackTime (dt, Heure);
```

Cette instruction est l'inverse de la précédente.

Exemple récapitulatif sur les dates et heures de fichiers

Vous pouvez télécharger le programme annoté **FDATE.PAS** :

```
Program FDate;

Uses
  Dos; {unité utilisée}

Var
  f : File;
  Heure : LongInt;
  dt : DateTime;

Function Format (w : Word) : String;
{ Convertit un word en chaîne sur 2 caractères }
Var
  s : String;
Begin {début de la fonction}
  Str(w,s); {convertit w en chaîne}
  if Length(s) = 1 then s := '0' + s;   {toujours sur 2 caractères}
  Format := s; {la fonction FORMAT prend la valeur de s}
End; {fin de la fonction}

BEGIN {début du programme}
  Assign(f,'c:\autoexec.bat'); {la variable f est assignée au fichier indiqué}
  {lecture de l'heure de dernière modification du fichier.
   L'information correspondante est consignée dans la variable Heure}
  GetFTime(f,Heure);
  WriteLn(Heure); {écriture à l'écran de la valeur de cette variable}
  UnpackTime(Heure,dt); {conversion de l'information vers dt}
  WriteLn(Format(dt.Hour),':',Format(dt.Min),':',Format(dt.Sec));
END. {fin du programme}
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

Chapitre 17 - Commandes système

Tout comme sous Dos ou Windows, il est quelquefois nécessaire de créer des répertoires, de déplacer des fichiers... Turbo Pascal 7.0 propose un certain nombre de procédures et fonctions permettant ces manipulations. Certaines d'entre elles seront discutées au cours de ce chapitre.
Les instructions suivantes nécessitent l'unité Dos.

Répertoires et lecteurs

MkDir

```
MkDir (s);
```

Procédure qui crée le sous-répertoire s (qui est une variable de type String) dans le lecteur et répertoire courant.

RmDir

```
RmDir (s);
```

Procédure qui supprime le sous-répertoire s (qui est une variable de type String) dans le lecteur et répertoire courant. Le sous-répertoire en question doit être **vide**.

ChDir

```
ChDir (s);
```

Procédure qui change de répertoire courant pour aller dans le répertoire s (qui est une variable de type String). Ce dernier doit bien entendu exister.

GetDir

```
GetDir (b, s);
```

Procédure qui renvoie le répertoire courant dans la variable s (de type String) du lecteur b (spécifié en Byte).

Valeur	Lecteur
0	Lecteur courant
1	A:
2	B:
3	C:
	... etc

Exemple récapitulatif

```
Program Exemple16;
Uses
  Dos;
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

```
Var
  s, r, t : String;
  i : Integer;
BEGIN
  GetDir(0,s);
  Writeln('Lecteur et répertoire courant: ',s);
  {$I-}
  Write('Aller dans quel répertoire ? -> ');
  ReadLn(r);
  for i := 1 to Length(r) do r[i] := UpCase(r[i]);
  ChDir(r);
  if IOResult <> 0 then
    begin
      Write(r,' n''existe pas, voulez-vous le créer [o/n] ? -> ');
      Readln(s);
      if (s = 'o') or (s = 'O') then
        begin
          MkDir(r);
          WriteLn('Création de ',r);
        end;
    end
  else Writeln('Ok : ',r,' existe !');
  ReadLn;
  ChDir(s);
END.
```

Ce programme *Exemple16* affiche le répertoire courant du disque courant et propose à l'utlisateur de changer de répertoire. Si le répertoire choisi n'existe pas, il le crée.

DiskFree

```
DiskFree (b);
```

Fonction qui retourne, dans une variable de type LongInt, la taille libre (en octets) du disque se trouvant dans le lecteur b (variable de type Byte).

Le type LongInt limite la taille maximale gérable à 2 Gb, ce qui est très inférieur aux tailles de disque actuelles !

DiskSize

```
DiskSize (b);
```

Fonction qui retourne dans une variable de type LongInt la capacité totale exprimée en octets du disque spécifié b (de type Byte).
La limitation à 2 Gb mentionnée ci-dessus est également valable pour cette fonction.

```
Program Exemple17;
Uses
  Dos;
BEGIN
  Writeln(DiskSize(0),' octets');
  Writeln(DiskSize(0) div 1024,' kilo octets');
  Writeln(DiskSize(0) div 1048576,' méga octets');
  Writeln(DiskSize(0) div 1073741824,' giga octets');
END.
```

Ce programme *Exemple17* affiche à l'écran la capacité totale du disque dur, dans les différentes unités usuelles.

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

Rappelons que le coefficient mutiplicateur entre les différentes unités est de 1024 : 1 kilo octet = 1024 octets, 1 méga octet = 1024 kilo octets, etc.

Environnement MS-DOS

DosVersion

```
DosVersion;
```

Fonction qui retourne le numéro de version du système d'exploitation MS-DOS présent dans le système sous la forme d'une variable de type Word.

```
Program Exemple18;
Uses
  Dos;
Var
  Version : Word ;
BEGIN
  Version := DosVersion;
  WriteLn('MS-DOS version : ',Lo(Version),'.',Hi(Version));
END.
```

Ce programme *Exemple18* affiche le numéro de la version de MS-DOS, correctement formatée avec les fonctions Lo et Hi qui renvoient respectivement le byte inférieur et supérieur de l'information contenue dans la variable Version.

DosExitCode

```
DosExitCode;
```

Fonction qui renvoie le code de sortie d'un sous-processus sous la forme d'une variable de type Word.

Valeur	Signification
0	Sortie sans erreur
1	Sortie par Ctrl-C
2	Device error
3	Procédure Keep (TSR)

EnvCount et EnvStr

```
EnvCount;
```

Fonction qui renvoie, sous la forme d'une variable de type Integer, le nombre de chaînes de caractères contenues dans l'environnement MS-DOS.

```
EnvStr (i);
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

Fonction qui renvoie, sous la forme d'une variable de type String, la chaîne de caractères contenue dans l'environnement MS-DOS à la position dans l'index spécifiée par la variable i (de type Integer).

```
Program Exemple19;
Uses
  Dos;
Var
  i : Integer;
BEGIN
  for i := 1 to EnvCount do WriteLn(EnvStr(i));
END.
```

Ce programme *Exemple19* affiche l'intégralité des variables d'environnement MS-DOS à l'aide d'une boucle.

GetCBreak et SetCBreak

```
GetCBreak (Break);
```

Procédure qui permet de connaître la valeur (true ou false) de la variable **Break** de MSDOS. Avec Break de type Boolean.

```
SetCBreak (Break);
```

Procédure qui permet de fixer la valeur (true ou false) de la variable **Break** de MSDOS. Avec Break de type Boolean.

Fichiers

DosError

DosError est une variable interne de l'unité Dos qui contient le code d'erreur renvoyé par MS-DOS lors de la dernière opération qui lui a été demandée.
Voici les codes d'erreur les plus fréquents :

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

Valeur	Description
0	Pas d'erreur
2	Fichier non trouvé
3	Répertoire non trouvé
5	Accès refusé
6	Procédure non valable
8	Pas assez de mémoire
10	Environnement non valable
11	Format non valable
18	Plus de fichiers

SetFAttr et GetFAttr

```
SetFAttr (f, Attr);
```

Procédure qui attribue au fichier f la variable Attr (de type Word).

```
GetFAttr (f, Attr);
```

Procédure qui renvoie dans la variable Attr (de type Word) la valeur de l'attribut du fichier f.
Les attributs de fichiers sont les suivants :

Valeur	Nom	Description
$01	ReadOnly	Lecture seule
$02	Hidden	Caché
$04	SysFile	Système
$08	VolumeID	VolumeID
$10	Directory	Répertoire
$20	Archive	Archive
$3F	AnyFile	tous

FExpand

```
FExpand (Fichier);
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

Fonction qui rajoute le chemin d'accès du fichier spécifié dans le nom de ce fichier. La variable Fichier doit être de type PathStr mais vous pouvez entrer directement une chaîne de caractères.

FSplit

```
FSplit (Fichier, Dir, Name, Ext);
```

Procédure qui découpe un nom de fichier (Fichier) de type PathStr en ses trois composantes :

- chemin (Dir), de type DirStr;
- nom (Name), de type NameStr;
- extension (Ext), de type ExtStr.

```
Program Exemple20;
Uses
  Dos;
Var
  P : PathStr;
  D : DirStr;
  N : NameStr;
  E : ExtStr;
BEGIN
  Write('Entrez un nom complet de fichier : ');
  Readln(P);
  FSplit(P,D,N,E);
  Writeln('Son chemin : "',D,'", son nom : "',N, '" et son extension : "',E,'".');
END.
```

Ce programme *Exemple20* demande à l'utilisateur d'entrer un nom de fichier avec son chemin, et il affiche séparément toutes les informations : le chemin, le nom et l'extension.

FileSize

```
FileSize (f);
```

Fonction qui renvoie, sous la forme d'une variable LongInt, la taille du fichier f.
L'utilisation du type LongInt entraîne à nouveau qu'une taille de fichier supérieure à 2 Gb ne pourra être correctement déterminée.

Recherche de fichiers et répertoires

Il est possible de rechercher des fichiers selon certains critères de nom et d'attribut avec les commandes FindFirst et FindNext. Regrouper ces commandes permet de simuler aisément la commande DIR de MS-DOS ou l'option RECHERCHER de Windows.
Exemple :

```
Program Exemple21;
Uses
  Dos;
Var
  Fichier : SearchRec;
BEGIN
  FindFirst('*.PAS',Archive,Fichier);
  while DosError = 0 do
    begin
      WriteLn(Fichier.Name);
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

```
      FindNext(Fichier);
    end;
END.
```

Ce programme *Exemple21* permet de rechercher et d'afficher le nom de tous les fichiers correspondant aux critères de recherche, c'est-à-dire les fichiers d'extension *PAS* et d'attribut *archive*.
Voir **chapitre 10** pour l'utilisation des fichiers externes, voir aussi **chapitre 16** pour la gestion des dates et heures.

Mémoire vîve

MemAvail

```
MemAvail;
```

Cette fonction retourne la mémoire totale libre en octets.

MaxAvail

```
MaxAvail;
```

Cette fonction retourne la longueur en octets du bloc contigu le plus grand de la mémoire vîve. Très utile pour connaître la taille allouable à un pointeur en cours d'exécution.

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

Chapitre 18 - Pseudo-hasard

Il est quelquefois nécessaire d'avoir recours à l'utilisation de valeurs de variables (scalaire ou réel) qui soient le fruit du hasard. Mais l'ordinateur n'est pas capable de créer du vrai hasard. Il peut cependant fournir des données dites **pseudo-aléatoires**, c'est-à-dire issues de calculs qui utilisent des paramètres issus de l'horloge interne. On appelle cela un pseudo-hasard car il est très difficile de déceler de l'ordre et des cycles dans la génération de ces valeurs pseudo-aléatoires. Ainsi, on admettra que Turbo Pascal 7.0 offre la possibilité de générer des nombres aléatoires.
Avant l'utilisation des fonctions qui vont suivre, il faut initialiser le générateur pseudo-aléatoire (tout comme il faut initialiser la carte graphique pour faire des dessins) avec la procédure Randomize. Cette initialisation est indispensable : en son absence, on obtiendra toujours la même séquence de nombres.
Syntaxe :

```
Randomize;
```

Randomize doit être appelée **une seule fois**, de préférence au début du programme principal.
On peut générer un nombre **réel** pseudo-aléatoire compris entre 0 et 1 grâce à la fonction Random.
Syntaxe :

```
X := Random;
```

On peut générer un nombre **entier** pseudo-aléatoire compris entre 0 et Y-1 grâce à la fonction Random(Y).
Syntaxe :

```
X := Random(Y);
```

Exemple :

```
Program Exemple22;
Uses
  Crt;
Const
  Max = 100;
Var
  Test : Boolean;
  x, y : Integer;
BEGIN
  ClrScr;
  Randomize;
  y := Random(Max);
  repeat
    Write('Entrez un nombre : ');
    ReadLn(x);
    Test := (x = y);
    if test then
      WriteLn('Ok, en plein dans le mille.')
    else
      if x > y then
        WriteLn('Trop grand.')
      else
        WriteLn('Trop petit.');
  until Test;
  ReadLn;
END.
```

Dans ce programme *Exemple22* (programme *Chance* typique des calculettes), on a génération d'un nombre pseudo-aléatoire compris entre 0 (inclus) et une borne Max (exclue) définie comme constante dans la partie déclarative du programme : **0 <= nombre < Max**. Ici, on prendra la valeur 100. L'utilisateur saisit un nombre, le programme effectue

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

un test et donne la valeur true à une variable Boolean nommée Test si le nombre du joueur est correct; sinon, affiche les messages d'erreurs correspondants. Le programme fonctionne à l'aide d'une boucle repeat...until.

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

Chapitre 19 - Paramètres de ligne de commande

Il peut être utile de pouvoir passer des paramètres en ligne de commande lors du lancement d'un programme.
La fonction ParamCount (de l'unité System) renvoie le nombre de paramètres passés en ligne de commande lors du lancement du programme, sous une valeur de type Word.
Syntaxe :

```
i := ParamCount;
```

La fonction ParamStr (également de l'unité System) renvoie la chaîne passée en commande selon sa place i (word) dans l'index.

La chaîne d'indice 0 contient le chemin d'accès et le nom de fichier du programme en cours.

Syntaxe :

```
s := ParamStr(i);
```

Exemple :

```
Program Exemple23;

Uses
  Dos ;

Var
  i : Word;
  f : Text;
  s : String;

Procedure Acces;
Begin
  WriteLn('Ok');
  ...
End;

BEGIN
  Assign(f,'password.dat');
  Reset(f);
  if IOResult = 0 then
    begin
      ReadLn(f,s);
      if IOResult = 0 then
        begin
          if ParamStr(1) = s then
            Acces
          else WriteLn('Invalid password.');
        end
      else
        WriteLn('File read error.');
      Close(f);
    end
  else
    WriteLn('File open error.');
END.
```

L'accès au programme *Exemple23* est protégé par un mot de passe. C'est-à-dire que seul un code passé en ligne de commande (et contenu dans un fichier) permet à l'utilisateur de faire tourner le programme.

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

Chapitre 20 - Types

Il est possible de créer de nouveaux types de variables sur Turbo Pascal 7.0. Il y a encore quelques décennies, un "bon" programmeur était celui qui savait optimiser la place en mémoire que prenait son programme, et donc la "lourdeur" des types de variables qu'il utilisait. Par conséquent, il cherchait toujours à n'utiliser que les types les moins gourmands en mémoire. Par exemple, au lieu d'utiliser un Integer pour un champ de base de données destiné à l'âge, il utilisait un Byte (1 octet contre 2). (voir **chapitre 4** sur les types de variables). Il est donc intéressant de pouvoir manipuler, par exemple, des chaînes de caractères de seulement 20 signes : String[20] (au lieu de 255 pour String, ça prend moins de place).
Les variables de types simples comme celles de types complexes peuvent être passées en paramètre à une procédure ou fonction.

Types simples

On déclare les nouveaux types simples dans la partie déclarative du programme, avant toute utilisation dans la déclaration de variables.
Syntaxe :

```
Type nom_du_type = nouveau_type;
```

Exemples :

```
Type Nom = String[20];
Type Entier = Integer;
```

```
Program Exemple24;
Type
  Chaine = String[20];
Var
  Nom : Chaine;
  Age : Byte;
BEGIN
  Write('Entrez votre nom : ');
  ReadLn(Nom);
  Write('Entrez votre âge : ');
  ReadLn(Age);
  WriteLn('Votre nom est : ',Nom,' et votre âge : ',Age);
END.
```

Ce programme *Exemple24* utilise un nouveau type appelé Chaine qui sert à déclarer la variable Nom.

Types intervalles

Les types intervalles sont très utilisés. Ils peuvent être de n'importe quel type scalaire. Un intervalle est forcément ordonné et continu.
Syntaxe :

```
Type mon_type = borne_inf..borne_sup;
```

On doit obligatoirement avoir :

- borne_inf et borne_sup de n'importe quel type scalaire
- borne_inf <= borne_sup

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

Exemples :

```
Type Binaire = 0..1;
Type Alpha = 'A'..'Z';
Type Cent = 1..100;
```

Les instructions suivantes :

- Inc() (incrémentation de la variable passée en paramètre),
- Dec() (décrémentation de la variable passée en paramètre),
- Succ() (renvoie le successeur de la variable passée en paramètre),
- Pred() (renvoie le prédécesseur de la variable passée en paramètre),
- Ord() (renvoie l'index de la variable dans l'intervalle auquel elle appartient)

s'appliquent aux types intervalles, qu'ils soient de type nombre entier ou caractère et énumérés.
Exemple :

```
Program Exemple31;
Const
  Max = 100;
Type
  Intervalle = 1..Max;
Var
  x : Intervalle;
BEGIN
  x := 1;
  {...}
  if not(Succ(x) = Max) then Inc(x);
  {...}
END.
```

Types énumérés

Un type énuméré est un type dont les variables associées n'auront qu'un nombre très limité de valeurs (au maximum 256 différentes possibles). La définition d'un type énuméré consiste à déclarer une liste de valeurs possibles associées à un type.

```
Program Exemple32;
Type
  Jours = (dim, lun, mar, mer, jeu, ven, sam);
Var
  Today : Jours;
BEGIN
  Today := mar;
  Today := Succ(Today);
  Inc(Today,2);
  case Today of
    dim : WriteLn('Dimanche');
    lun : WriteLn('Lundi');
    mar : WriteLn('Mardi');
    mer : WriteLn('Mercredi');
    jeu : WriteLn('Jeudi');
    ven : WriteLn('Vendredi');
    sam : WriteLn('Samedi');
  else
    WriteLn('autre, ',Ord(today));
  end;
END.
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

Dans ce programme *Exemple32*, il est déclaré un type Jours de type énuméré composé de 7 éléments représentant les jours de la semaine. Remarquez que les éléments sont uniquement des identifiants qui n'ont aucune valeur intrinsèque; on peut tout juste les repérer par leur index (l'ordre dans lequel ils apparaissent dans la déclaration, où le premier élément a le numéro 0 et le dernier : n-1). Tout d'abord, une affectation à l'aide de l' opérateur habituel := vers la variable Today. Puis on lui affecte son successeur dans la déclaration. Ensuite, on l'incrémente de 2. Et, selon sa valeur, on affiche à l'écran le jour de la semaine correspondant si cela est possible.
Remarque : La fonction Chr() (propre au type Char) ne s'applique pas aux types intervalles et énumérés.

Il est impossible d'utiliser les procédures Write(Ln) et Read(Ln) avec les variables de type énuméré.

Exemple :

```
Program Exemple35;
Type
  Couleurs = (red, yellow, green, black, blue);
Var
  Color : Couleurs;
BEGIN
  Color := red;
  repeat
    Inc(Color);
  until Color > green;
  if Color = black then WriteLn('Noir');
END.
```

Ce programme *Exemple35* montre que, comme toute variable, Color - qui est de type énuméré - peut être sujette à des tests booléens. Ici, sa valeur est incrémentée dans une boucle Repeat qui ne s'arrête que lorsque Color atteint une valeur qui, dans le type énuméré, est supérieure à la valeur green. Ensuite un test If vient confirmer que la dernière valeur prise par Color (à laquelle on s'attendait au vu de la définition du type énuméré) est black.

Types structurés (enregistrements)

Un type structuré (ou enregistrement) est une structure de données consistant en un nombre fixé de composants, appelés **champs**. A la différence du tableau, ces composants ne sont pas obligatoirement du même type, et ne sont pas indexés.
La définition du type **enregistrement** précise pour chaque composant un identificateur de champ, dont la portée est limitée à l'enregistrement, et le type de ce composant.
Syntaxe :

```
Type nom_du_type = Record
                     champ1 : type1;
                     champ2 : type2;
                     champ3 : type3;
                   end ;
```

Note : les champs sont placés dans un bloc Record...end ; et un champ peut lui-même être de type Record.
Syntaxe :

```
Type nom_du_type = Record
                     champ1 : type1 ;
                     champ2 = Record;
                                champ2_1 : type2;
                                champ2_2 : type3;
                                champ2_3 : type4;
                              end ;
                   end;
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

On ne peut pas afficher le contenu d'une variable structurée sans passer par une syntaxe spécifiant le champ dont on veut connaître la valeur.
Note : les champs d'une variable de type structuré peuvent être de tout type (même tableau).

avec with

```
Program Exemple25a;
Type
  Descendance = 0..15;
  Formulaire = Record
                 Nom : String[20];
                 Age : Byte;
                 Sexe : Char;
                 Nb_enfants : Descendance;
               end ;
Var
  Personne : Formulaire;
BEGIN
  with Personne do
    begin
      Nom := 'Etiévant';
      Age := 18;
      Sexe := 'M';
      Nb_enfants := 3;
    end ;
END.
```

sans with

```
Program Exemple25b;
Type
  Descendance = 0..15;
  Formulaire = Record
                 Nom : String[20];
                 Age : Byte;
                 Sexe : Char;
                 Nb_enfants : Descendance;
               end;
Var
  Personne : Formulaire ;
BEGIN
  Personne.Nom := 'Etiévant';
  Personne.Age := 18;
  Personne.Sexe := 'M';
  Personne.Nb_enfants := 3;
END.
```

Ces programmes *Exemple25a* et *Exemple25b* sont pratiquement identiques. Ils utilisent tous deux une variable Personne de type Formulaire qui comprend quatre champs : Nom, Age, Sexe et Nb_enfants. L'utilisation de ces champs se fait ainsi : variable.nom_du_champ (*Exemple25b*). Lorsqu'on les utilise à la chaîne (*Exemple25a*), on peut faire appel à with.

```
Program Exemple25c;
Type
  Bornes_Jour = 1..31;
  Bornes_Mois = 1..12;
  Bornes_An = 1900..2000;
  Date = Record
           Jour : Bornes_Jour;
           Mois : Bornes_Mois;
           An : Bornes_An;
         end ;
Type
  Formulaire = Record
                 Nom : String[20];
                 Date_naissance : Date;
               end;
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

```
Var
  Personne : Formulaire;
BEGIN
  with Personne do
    begin
      Nom := 'Etiévant';
      with Date_naissance do
        begin
          Jour := 21;
          Mois := 10;
          An := 1980;
        end ;
    end ;
END.
```

```
Program Exemple25d;
Type
  Bornes_Jour = 1..31;
  Bornes_Mois = 1..12;
  Bornes_An = 1900..2000;
  Formulaire = Record
                 Nom : String[20];
                 Date_naissance : Record
                                    Jour : Bornes_Jour;
                                    Mois : Bornes_Mois;
                                    An : Bornes_An;
                                  end;
               end ;
Var
  Personne : Formulaire;
BEGIN
  with Personne do
    begin
      Nom := 'Etiévant';
      with Date_naissance do
        begin
          Jour := 21;
          Mois := 10;
          An := 1980;
        end;
    end;
END.
```

Là aussi, les programmes *Exemple25c* et *Exemple25d* sont pratiquement identiques. Ils utilisent tous deux une variable Personne de type Formulaire qui comprend deux champs : Nom et Date_naissance, qui elle-même est de type structuré et comprenant les champs Jour, Mois et An.

Enregistrements conditionnels

Lors de la création d'un enregistrement (type structuré), il est quelquefois nécessaire de pouvoir, en fonction d'un champ, décider de la création d'autres champs de tel ou tel type.
Une telle déclaration s'effectue grâce à la syntaxe Case of que l'on connaissait déjà.
Syntaxe :

Exemple avec un type énuméré

```
Type type_enumere = (element1, element2, ... elementN);
     mon_type = Record
                  champ1 : type_enumere;
                  case champ2 : type_enumere of
                    element1 : (champ3 : type3);
                    element2 : (champ4 : type4; champ5 : type5; ... champM : typeM);
                    ...
                    elementN : ( );
                end;
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

Exemple avec un autre type (Integer)

```
Type mon_type = Record
                  case champ1 : Integer of
                    0 : (b1 : Integer);
                    1000 : (b2 : Single);
                    ...
                end;
```

Le principe est que, suivant la valeur d'un champ fixe, on va procéder à la création d'un ou plusieurs champs.

```
Program Exemple30a;

Const
  Nmax = 1;

Type
  Materiaux = (metal, beton, verre);
  Produit = Record
              Nom : String[20];
              case Matiere : Materiaux of
                metal : (Conductivite : Real);
                beton : (Rugosite : Byte);
                verre : (Opacite : Byte; Incassable : Boolean);
            end;
  Tab = Array [1..Nmax] of Produit;

Procedure affichage(prod : produit) ;
Begin
  with prod do
    begin
      WriteLn('Produit ', nom);
      case Matiere of
        metal : WriteLn('Conductivité : ',Conductivite);
        beton : WriteLn('Rugosité : ',Rugosite);
        verre : begin
                  WriteLn('Opacité : ',Opacite);
                  if Incassable then WriteLn('Incassable');
                end;
      end;
    end ;
End;

Var
  x : Tab;
  i : Integer;

BEGIN
  with x[1] do
    begin
      Nom := 'Lampouille';
      Matiere := verre;
      Opacite := 98;
      Incassable := True;
    end;
  for i := 1 to Nmax do Affichage(x[i]);
END.
```

Il est absolument nécessaire de remplir le champ qui conditionne le choix des autres champs avant de remplir les champs qui sont soumis à condition.
Sinon, il est renvoyé des résultats absurdes.

Les résultats sont absurdes parce que les champs variables occupent la même place en mémoire, et que la valeur d'un type n'est que rarement totalement compatible avec les valeurs d'un autre type.

```
Program Exemple30b;
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

```
Type
  Toto = Record
           case i : Integer of
             1 : ( );
             2 : (a:Real);
             3 : (x, y : String);
         end;
Var
  x : Toto;
BEGIN
  x.i := 2;
  x.a := 2.23;
  WriteLn(x.a);
  x.i := 3;
  x.x := 'Castor';
  WriteLn(x.x);
END.
```

Ce dernier programme *Exemple30b* montre l'utilisation du type Integer, dont la valeur dans le programme conditionne l'existence d'autres champs.

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

Chapitre 21 - Tableaux

Il est courant d'utiliser des tableaux afin d'y stocker temporairement des données. Ainsi, une donnée peut être en relation avec 1, 2 ou 3 (ou plus) entrées. L'intérêt du tableau est de pouvoir stocker en mémoire des données que l'on pourra retrouver grâce à d'autres valeurs à l'aide de boucles, de formules, d'algorithmes. On peut utiliser un tableau afin de représenter l'état d'un échiquier, le résultat d'une fonction mathématique...
Un tableau est constitué d'un nombre fixe (donné à la définition) de composants du même type. Chaque composant peut être explicitement référencé par le nom du tableau suivi de son indice entre crochets. Les indices sont des valeurs calculables appartenant toutes au type d'indice. Le type d'indice est tout type pouvant être interprété comme un entier (Integer, byte, types énumérés, caractères...).
Il est possible d'introduire des variables de tous les types au sein d'un tableau : Char, Integer, Real, String, Byte, Record, etc.
Un tableau, tout comme une variable quelconque, doit être déclaré dans la partie déclarative du programme. On doit toujours spécifier le type des variables qui seront introduites dans le tableau.
Syntaxe :

```
Type Tableau = Array [MinDim..MaxDim] of type;
Var nom_du_tableau : Tableau;
```

MinDim doit être inférieur ou égal à MaxDim. L'un ou les deux peuvent êtres négatifs.
Exemples :

```
Var tab1 : Array [0..10] of Byte;

Var tab2 : Array [1..100] of Integer;

Var tab3 : Array [-10..10] of Real;

Var tab4 : Array [50..60] of String;

Var tab5 : Array ['A'..'S'] of Char;

Var tab6 : Array ['a'..'z'] of Extended;
```

Exemples avec type énuméré

```
Type
  jours_enum = (dimanche,lundi,mardi,mercredi,jeudi,vendredi,samedi);

  tableau_1 = Array [dimanche..samedi] of Integer;

  tableau_2 = Array [jours_enum] of Integer;

  tableau_3 = Array [mardi..vendredi] of Integer;
```

Remarque : le fait que les bornes d'un tableau soient déclarées par des valeurs de type caractère (Char) n'interdit pas pour autant de remplir le tableau de nombres réels (voir le tab6 ci-dessus). Car, en effet, le type des bornes d'un tableau n'influe aucunement sur le type des variables contenues dans le tableau. Et réciproquement, le type des variables d'un tableau ne renseigne en rien sur le type des bornes ayant servi à sa déclaration.
Un tableau peut avoir **plusieurs dimensions**. Si toutefois, vous imposez trop de dimensions ou des index trop importants, une erreur lors de la compilation vous dira : ***Error 22: Structure too large***.
Syntaxes :

```
Var nom_du_tableau : Array [MinDim1..MaxDim1, MinDim2..MaxDim2] of type;
Var nom_du_tableau : Array [MinDim1..MaxDim1, MinDim2..MaxDim2,MinDim3..MaxDim3] of type;
```

Exemples :

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

```
Var tab1 : Array [0..10, 0..10] of Byte;

Var tab2 : Array [0..10, 0..100] of Integer;

Var tab3 : Array [-10..10, -10..10] of Real;

Var tab4 : Array [5..7, 20..22] of String;

Var tab5 : Array [1..10, 1..10, 1..10, 0..2] of Char;
```

La technique pour introduire des valeurs dans un tableau est relativement simple. Il suffit de procéder comme pour une variable normale, sauf qu'il ne faut pas oublier de spécifier la position index qui indique la place de la valeur dans la dimension du tableau.
Syntaxes :

```
nom_du_tableau[index] := valeur;

nom_du_tableau[index1,index2] := valeur;
```

La variable index doit nécessairement être du même type que celui utilisé pour la déclaration du tableau.

On peut copier dans un tableau les valeurs d'un autre tableau. Mais pour cela, il faut que les deux tableaux en question soient du même type, qu'ils aient le même nombre de dimension(s) et le même nombre d'éléments.
Syntaxe :

```
nom_du_premier_tableau := nom_du_deuxième_tableau;
```

Il existe une autre manière de déclarer un tableau de dimensions multiples en créant un tableau de tableau. Mais cette technique n'est pas la plus jolie, pas la plus pratique, pas la plus appréciée aux examens...
Syntaxe :

```
Var nom_du_tableau : Array [MinDim1..MaxDim1] of Array [MinDim2..MaxDim2] of type; {syntaxe désuette}
```

Voici également une autre manière d'introduire des valeurs.
Syntaxe :

```
nom_du_tableau [index1][index2] := valeur;
```

Exemple :

```
Const mt = 3;

Type
  Tab = Array [1..mt,1..mt] of Integer;

Var
  i, j : Integer;
  t : Tab;

BEGIN
  { forme habituelle }
  for i := 1 to mt do
    for j := 1 to mt do t[i,j] := i + j * 10;
  { autre forme }
  for i := 1 to mt do
    for j := 1 to mt do write(t[i][j]:3);
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

```
END.
```

Le passage d'un tableau (type complexe) en paramètre à une procédure pose problème si on ne prend pas des précautions.
Syntaxe :

```
Type nom_du_nouveau_type_tableau = Array [DimMin..DimMax] of Type;

Var nom_du_tableau : nom_du_nouveau_type_tableau;

Procedure nom_de_la_procedure (Var nom_du_tableau : nom_du_nouveau_type_tableau);
```

Exemple :

```
Program Exemple27;

Uses
  Crt;

Type
  Tableau = Array [1..10] of Integer;

Procedure Saisie (var Tab : Tableau);
Var i : Integer;
Begin
  for i := 1 to 10 do
    begin
      Write('Entrez la valeur de la case n°',i,'/10 : ');
      ReadLn(Tab[i]);
    end;
End;

Procedure Tri (var Tab : Tableau);
Var i, j, x : Integer;
Begin
  for i := 1 to 10 do
    begin
      for j := i to 10 do
        begin
          if Tab[i] > Tab[j] then
            begin
              x := Tab[i];
              Tab[i] := Tab[j];
              Tab[j] := x;
            end;
        end;
    end;
End ;

Procedure Affiche (var Tab : Tableau);
Var i : Integer;
Begin
  for i := 1 to 10 do Write(Tab[i],' ');
  WriteLn;
End;

Var
  Tab1, Tab2 : Tableau;
  i : Integer;

BEGIN
  ClrScr;
  Saisie(Tab1);
  Tab2 := Tab1;
  Tri(Tab2);
  WriteLn('Série saisie :');
  Affiche(Tab1);
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

```
  WriteLn('Série triée :');
  Affiche(Tab2);
END.
```

Ce programme *Exemple27* a pour but de trier les éléments d'un tableau d'entiers dans l'ordre croissant. Pour cela, il déclare un nouveau type de tableau grâce à la syntaxe Type. Ce nouveau type est un tableau à une dimension, de 10 éléments de type Integer. Une première procédure saisie charge l'utilisateur d'initialiser le tableau Tab1. Le programme principal copie le contenu de ce tableau vers un autre appelé Tab2. Une procédure Tri range les éléments de Tab2 dans l'ordre. Et une procédure Affiche affiche à l'écran le tableau Tab1, qui contient les éléments dans introduits par l'utilisateur, et le tableau Tab2, qui contient les mêmes éléments mais rangés dans l'ordre croissant.
Il est également possible d'introduire dans un tableau des données complexes, c'est-à-dire de déclarer un tableau en type complexe (Record).
Syntaxe :

```
Var Enreg = Record
              Nom : String[20];
              Age : Byte;
            end;
Tab : Array [1..10] of Enreg;
```

Introduire des valeurs dans un tel tableau nécessite d'utiliser en même temps la syntaxe des tableaux et des types complexes.
Syntaxe :

```
Tab[5].Nom := 'CyberZoïde';
```

Il est possible de déclarer un tableau en tant que constante (voir **chapitre 26** sur les constantes).
Syntaxe :

```
Const a : Array [0..3] of Byte = (103, 8, 20, 14);

      b : Array [-3..3] of Char = ('e','5','&','Z','z',' ',#80);

      c : Array [1..3, 1..3] of Integer = ((200,23,107), (1234,0,5), (1,2,3));

      d : Array [1..26] of Char = 'abcdefghijklmnopqrstuvwxyz';
```

Déclarées ainsi (***variable : type = valeur***), les constantes sont en fait des **variables initialisées**. Il est possible de modifier leur valeur au cours du programme.

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

Chapitre 22 - Une bonne interface DOS

Lors de la création d'un programme, l'élaboration de l'**interface utilisateur** est très critique, demande du code et de la patience. Outre les procédures systématiques de gestion des erreurs et de contrôle des entrées, la présentation des données aussi bien en mode MS-DOS qu'en mode graphique conditionne bien souvent la qualité de la diffusion d'un programme. Il est capital de présenter à l'utilisateur les données de façon ordonnée, aérée et claire. Car quand, dans un programme, des informations apparaissent à l'écran et qu'on ne sait pas d'où elles viennent, ni comment intervenir sur son déroulement, l'utilisateur est frustré et l'abandonne.
Il est donc nécessaire, ne serait-ce que pour être à l'aise devant son écran, de construire une interface simple et claire. Doter son programme d'une interface efficace commence déjà par lui donner un nom ou, du moins, un descriptif de l'objectif ou du travail qu'il doit produire ainsi que des diverses opérations qu'il est capable de réaliser. En général, cela commence par l'affichage d'un menu du type suivant :

```
- GESTION D'UN TABLEAU -
[A] - Quitter
[B] - Création du tableau
[C] - Affichage du tableau
[D] - Modification d'un élément du tableau
[E] - Ajout d'un élément à la fin du tableau
[F] - Suppression d'un élément du tableau
[G] - Insertion d'un élément dans le tableau
[H] - Recherche du plus petit élément dans le tableau
[I] - Recherche du plus grand élément dans le tableau
Entrez votre choix : _
```

Il faut être capable de gérer correctement l'entrée de l'utilisateur pour être sûr qu'il entre un paramètre correct :

```
Procedure Menu (var Reponse : Char);
Begin
  repeat
    ClrScr;
    WriteLn('[A] - Quitter');
    WriteLn('[B] - Création du tableau');
    WriteLn('[C] - Affichage du tableau');
    WriteLn('[D] - Modification d''un élément du tableau');
    WriteLn('[E] - Ajout d''un élément à la fin du tableau');
    WriteLn('[F] - Suppression d''un élément du tableau');
    WriteLn('[G] - Insertion d''un élément dans le tableau');
    WriteLn('[H] - Recherche du plus petit élément dans le tableau');
    WriteLn('[I] - Recherche du plus grand élément dans le tableau');
    WriteLn;
    Write('Entrez votre choix : ');
    Reponse := ReadKey;
  until UpCase(Reponse) in ['A'..'I'];
End;
```

Ensuite, il faut pouvoir détailler les opérations et leurs résultats afin d'offrir le maximum d'informations à l'utilisateur :

```
[H] - RECHERCHE DU PLUS PETIT ELEMENT DU TABLEAU
Résultat de la recherche :
            - rang : 27
            - valeur : -213

Recherche terminée, tapez sur <ENTREE> pour retourner au menu.
```

La recherche d'erreurs fait également partie des devoirs du programmeur, c'est-à-dire que le programme doit être capable de repérer des entrées erronées, d'expliquer à l'utilisateur ce qui ne vas pas, pourquoi ça ne vas pas et lui permettre de recommencer sa saisie :

```
Procedure Modifier (var Tab1 : Tableau; n : Integer);
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

```
Var Rang, Valeur : Integer;
    Test : Boolean;
Begin
  ClrScr;
  WriteLn('[D] - MODIFICATION D''UN ELEMENT DU TABLEAU');
  if n >= 1 then
    begin
      repeat
        Write('Entrez le rang [1,',n,'] de l''élément à modifier : ');
        ReadLn(Rang);
        Test := (Rang > 0) and (rang <= n);
      until Test;
      Write('Entrez la nouvelle valeur de l''élément : ');
      ReadLn(Valeur);
      Tab1[Rang] := Valeur;
      WriteLn('Modification terminée, tapez sur ENTREE pour retourner au menu.');
    end
  else
    WriteLn('Aucun élément dans le tableau.');
  ReadLn ;
End;
```

Très souvent, le tableau est la forme la plus appropriée pour présenter des données à l'écran texte. Il est donc souhaitable de construire des tableaux multi-cadres afin d'avoir à l'écran plusieurs informations simultanément comme le montre l'exemple suivant :

```
*=================================================================*
|                       SuperCalculator 2.5                       |
*=================================================================*
+-------------------------------------------------+----------------+
¦  Module : CALCUL DE FONCTION                     ¦   Page :   3   ¦
+-------------------------+-------------------------+----------------+
¦      Valeur de X        ¦     Valeur de Y=f(X)    ¦
¦                         ¦                         ¦
¦          1450           ¦          2103818        ¦
¦          1500           ¦          2251364        ¦
¦          1550           ¦          2403909        ¦
¦          1600           ¦          2561455        ¦
¦          1650           ¦          2724000        ¦
¦          1700           ¦          2891545        ¦
¦          1750           ¦          3064091        ¦
+-------------------------+-------------------------+
¦      Appuyez sur <ENTREE> pour continuer...       ¦
+-------------------------------------------------+
```

(4)

```
Program Exemple28;

Uses
  Crt;

Var
  i, j : Integer;
  x : Real;
  Test : Boolean;

Procedure Menu (Page : Integer; Test : Boolean) ;
Begin
  WriteLn('*=================================================================*');
  WriteLn('|                       SuperCalculator 2.5                       |');
  WriteLn('*=================================================================*');
  if Test then
    WriteLn('Faire afficher le menu principal...')
```

(4) Ici, les caractères spéciaux de la table ASCII (pour MS-DOS) ont été remplacés par des -, des + et des ¦ car non affichables sous Windows.

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

```
  else
    begin
      WriteLn('+-------------------------------------------------+----------------+');
      WriteLn('¦  Module : CALCUL DE FONCTION                     ¦   Page :   ',Page,'   ¦');
      WriteLn('+-----------------------+-------------------------+----------------+');
      WriteLn('¦      Valeur de X      ¦     Valeur de Y=f(X)    ¦');
      WriteLn('¦                       ¦                         ¦');
    end ;
End;

Function f (x : Real) : Real;
Begin
  f := Sqr(x);
End;

BEGIN
  ClrScr;
  i := 0;
  j := 1;
  Test := False;
  Menu(j,Test) ;
  x := 0;
  repeat
    Inc(i);
    x := x + 50;
    WriteLn('¦      ',Round(x):12,'       ¦     ',Round(f(x)):16,'    ¦');
    Test := x > 1700;
    if ((i Mod 14) = 0) or test then
      begin
        WriteLn('+-----------------------+-------------------------+');
        WriteLn('¦      Appuyez sur <ENTREE> pour continuer...      ¦');
        WriteLn('+-------------------------------------------------+');
        ReadLn;
        ClrScr;
        Inc(j);
        Menu(j,Test);
      end;
  until Test;
  ReadLn;
END.
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

Chapitre 23 - Gestion de la mémoire par l'exécutable

Limite virtuelle de la mémoire

Une fois compilé (commande **Run**, **Compile** ou **Make** de l'éditeur), un programme gère la mémoire très rigoureusement. Le tableau ci-dessous vous montre que les variables globales, les variables locales des sous-programmes et les pointeurs ne sont pas stockés dans les mêmes parties de la mémoire. En effet, les variables globales ont la part belle, la mémoire allouée aux pointeurs est variable et celle destinée aux variables locales est assez restreinte.

Zone mémoire	Taille	Utilisation
Pile (Stack)	16 ko par défaut	Destiné aux variables locales des sous-programmes (procédures, fonctions) ainsi qu'aux valeurs passées par valeur aux sous-programmes
Tas (Heap)	de 0 à 640 ko	Réservé aux pointeurs
Segment de données	64 ko	Destiné aux variables du programme principal

Explications :
les sous-programmes étant destinés à des calculs intermédiaires, ils n'ont guère besoin d'énormément de ressource mémoire. Quant aux pointeurs que l'on verra dans le chapitre suivant, ils sont destinés à la création de variables (dites **dynamiques**) au cours de l'exécution du programme.
Mais il est toutefois possible de modifier manuellement une telle organisation de la mémoire afin, par exemple, de privilégier la pile grâce à la directive de compilation suivante :

```
{$M n1, n2, n3}
```

Ce type de directive est destiné au compilateur, qui inscrira les informations spécifiées dans le programme compilé. Une directive de compilation se présente entre accolades, comme n'importe quel autre commentaire, mais un signe dollar "$" signifie qu'il est destiné au compilateur. Quand au "M" il dit au compilateur qu'on souhaite réorganiser la disposition de la mémoire à l'aide des valeurs n1, n2 et n3 qui spécifient respectivement : la **taille** en kilo octets **de la pile** (doit être inférieur à 64 ko), la **taille minimale** et la **taille maximale** (inférieur à 640 ko) **du tas**.
Mais pourquoi s'enquiquiner avec ça ? Tout simplement parce qu'il pourra vous arriver d'avoir insuffisamment de mémoire à cause d'un tableau trop long par exemple. Si vous déclarez une telle variable dans une procédure (une variable locale, donc) :

```
Var Tableau : Array [1..50, 1..100] of Real;
```

vous obtiendrez le message d'erreur n° 22 : ***Structure too large***, qui veut dire que votre variable tient trop de place pour être stockée dans la mémoire allouée. Car en effet, ce tableau occupe : 50 * 100 * 6 octets = 29 ko (1 ko = 2^10 = 1024 octets) 29 ko > 16 ko donc le compilateur renvoie une erreur. Et le seul moyen qui vous reste est de modifier les valeurs correspondantes aux grandeurs allouées à la pile par une directive de compilation ou en allant dans le menu **Option/Memory Size**. D'où l'intérêt du **chapitre 4** ("Différents types de variables"), qui vous indique la taille de chaque type de variable.

Passage de paramètre à un sous-programme

Dans le **chapitre 7** ("Procédures et sous-programmes"), vous avez appris qu'on pouvait passer un paramètre **par valeur** ou bien **par adresse** à une procédure paramétrée. Vous avez également compris l'intérêt de la syntaxe Var dans la déclaration d'une procédure.
Quand un sous-programme est appelé, le programme compilé réalise en mémoire (dans la pile) **une copie** de chaque argument passé au sous-programme. Ces copies ne sont que temporaires, puisque destinées au fonctionnement

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

de sous-programmes qui n'interviennent que temporairement dans le programme. Ainsi un changement de valeur au sein de la procédure d'un paramètre passé par valeur (sans le var) n'est pas répercuté dans le programme principal. Alors que dans le cas de la présence du mot-clé var, le programme ne duplique pas la valeur ainsi passée à la procédure *dans la pile*, mais passe l'adresse de la variable utilisée comme paramètre réel, ce qui permet à la procédure/fonction d'accéder à la variable du programme appelant, puisqu'elle connaît son adresse en mémoire. Ainsi, toute variation interne à la procédure est répercutée directement sur l'argument (la variable du programme principal passé en paramètre).

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

Chapitre 24 - Pointeurs

Un pointeur est une variable qui contient l'adresse mémoire d'une autre variable stockée en mémoire. Soit P le pointeur et P^ la variable "pointée" par le pointeur. La déclaration d'une variable pointeur réserve 4 octets nécessaires au codage de l'adresse mémoire.

⚠ ***La déclaration d'un pointeur ne réserve aucune mémoire pour la variable pointée.***
Cette mémoire, il faudra l'allouer dynamiquement *(plus loin dans ce chapitre).*

Jusqu'alors nous avons vu que la déclaration d'une variable provoque automatiquement la réservation d'un espace mémoire qui est fonction du type utilisé. Voir **chapitre 4** ("Différents types de variables") pour la taille en mémoire de chacun des types de variables utilisés ci-après.
Exemples :

```
Var Somme : Integer;
   { Réservation de 2 octets dans la mémoire }

Var Moyenne : Real;
   { Réservation de 6 octets dans la mémoire }

Var Tableau : Array [1..100] of Integer;
   { Réservation de 400 octets (100 * 4) dans la mémoire }

Var Nom : String [20];
   { Réservation de 21 octets dans la mémoire }

Var x, y, z : Integer;
   { Réservation de 6 octets (3 * 2) dans la mémoire }

Var Tab1, Yab2 : Array [0..10,0..10] of Integer;
   { Réservation de 484 octets (2 * 11 * 11 * 2) dans la mémoire }

Type Personne = Record
                  Nom, Prenom : String [20];
                  Age : Byte;
                  Tel : Integer;
                end;
Var Client, Fournisseur : Personne;
   { Réservation de 90 octets (2 * (2 * 21 + 1 + 2)) dans la mémoire }
```

On comprend rapidement que s'il vous prenait l'envie de faire une matrice de 100 * 100 réels (100 * 100 * 6 = 60 ko) à passer en paramètre à une procédure, le compilateur vous renverrait une erreur du type : ***Structure too large*** car il lui est impossible de réserver plus de 16 Ko en mémoire pour les variables des sous-programmes. Voir **chapitre 23** ("Gestion de la mémoire par l'exécutable").
D'où l'intérêt des pointeurs car, quelle que soit la taille de la variable pointée, la place en mémoire du pointeur est toujours la même : 4 octets. Ces quatres octets correspondent à la taille mémoire nécessaire pour stocker l'adresse mémoire de la variable pointée.
Mais qu'est-ce qu'est-ce qu'une adresse mémoire ? C'est en fait un enregistrement comprenant deux nombres de type Word (2 fois 2 octets font bien 4), qui représentent respectivement l'adresse du segment de donnée utilisé et l'indice (le *déplacement*) du premier octet servant à coder la variable à l'intérieur de ce même segment (un segment étant un bloc de 65536 octets). Cette taille de segment implique qu'une variable ne peut pas dépasser la taille de 65536 octets, et que la taille de l'ensemble des variables globales ne peut pas dépasser 65536 octets ou encore que la taille de l'ensemble des variables d'un sous-programme ne peut dépasser cette même valeur limite.
La déclaration d'un pointeur permet donc de réserver une petite place de la mémoire qui pointe vers une autre qui peut être très volumineuse. L'intérêt des pointeurs est que la variable pointée ne se voit pas réserver de mémoire dans la zone mémoire des variables globales, mais plutôt dans le **tas**. Puisque la **pile**, normalement destinée aux variables des sous-programmes, est trop petite (16 Ko), on utilise donc le **tas**, dont la taille peut atteindre 640 Ko.

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

Déclaration

Avant d'utiliser une variable de type pointeur, il faut déclarer ce type en fonction du type de variable que l'on souhaite pointer.
Exemple :

```
Type PEntier = ^Integer;
Var P : PEntier;
```

On déclare une variable P de type PEntier qui est en fait un pointeur pointant vers un Integer (à noter la présence indispensable de l'**accent circonflexe** !). Donc la variable P contient une adresse mémoire, celle d'une autre variable qui est elle, de type Integer. Ainsi, l'adresse mémoire contenue dans P est l'endroit où se trouve le premier octet de la variable de type Integer. Il est inutile de préciser l'adresse mémoire de fin de l'emplacement de la variable de type Integer car une variable de type connu, quelle que soit sa valeur, occupe toujours le même espace. Le compilateur sachant à l'avance combien de place tient tel ou tel type de variable, il lui suffit de connaître grâce au pointeur l'adresse mémoire du premier octet occupé et de faire l'addition **adresse mémoire contenue dans le pointeur + taille mémoire du type utilisé** pour définir totalement l'emplacement mémoire de la variable pointée par le pointeur.

Accès à la variable pointée

Tout ça c'est très bien mais comment fait-on pour accéder au contenu de la variable pointée par le pointeur ? Il suffit d'utiliser l'identificateur du pointeur à la fin duquel on rajoute un accent circonflexe.
Exemple :

```
P^ := 128;
```

Donc comprenons-nous bien, P est le pointeur contenant l'adresse mémoire d'une variable et P^ (avec l'accent circonflexe) contient la valeur de la variable pointée. On passe donc du pointeur à la variable pointée par l'ajout du symbole spécifique **^** à l'identificateur du pointeur.

```
Type Tableau = Array [1..100] Of Real;
PTableau = ^Tableau;
Var P : PTableau;
```

Ici, on déclare un type Tableau qui est un tableau de 100 Real. On déclare aussi un type de pointeur PTableau pointant vers le type Tableau. C'est-à-dire que toute variable de type PTableau, contiendra l'adresse mémoire du premier octet d'une variable de type Tableau. Ce type Tableau occupe 100*6 = 600 octets en mémoire; le compilateur sait donc parfaitement comment écrire une variable de type Tableau en mémoire. Quant à la variable P de type PTableau, elle contient l'adresse mémoire du premier octet d'une variable de type Tableau. Pour accéder à la variable de type Tableau pointée par P, il suffira d'utiliser la syntaxe P^.
P étant le pointeur et P^ étant la variable pointée. P contenant donc une adresse mémoire et P^ contenant un tableau de 100 Real. Ainsi, P^[10] représente la valeur du dixième élément de P^ (c'est donc un nombre de type Real) tandis que P[10] est une opération invalide, qui déclenche une erreur du compilateur.

New et Dispose

La déclaration au début du programme des diverses variables et pointeurs a pour conséquence que les variables se voient allouer un bloc mémoire à la compilation. Et ce dernier reste réservé à la variable associée jusqu'à la fin du programme.
Avec l'utilisation des pointeurs, tout cela change puisque la mémoire est allouée **dynamiquement**. On a vu que seul le pointeur se voit allouer (réserver) de la mémoire (4 octets, c'est très peu) pour toute la durée de l'exécution du programme mais pas la variable correspondante. Il est cependant nécessaire de réserver de la mémoire à la valeur pointée en cours de programme (et pas forcément pour toute la durée de celui-ci) en passant en paramètre un

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

pointeur P qui contiendra l'adresse mémoire correspondant à la variable associée P^. Pour pouvoir utiliser la variable pointée par le pointeur, il est absolument indispensable de lui réserver dynamiquement de la mémoire comme suit :

```
New(P);
```

Et pour la supprimer, c'est-à-dire libérer la place en mémoire qui lui correspondait et perdre bien sûr son contenu :

```
Dispose(P);
```

Ainsi, lorsqu'on en a fini avec une variable volumineuse et qu'on doit purger la mémoire afin d'en utiliser d'autres tout autant volumineuses, on utilise Dispose. Si, après,au cours du programme, on veut réallouer de la mémoire à une variable pointée par un pointeur, c'est possible (autant de fois que l'on veut !).

Une variable allouée dans le tas par New contient n'importe quoi, jusqu'à ce que lui ait affecté une valeur !

```
Type Tab2D = Array [1..10,1..10] of Integer;
     PMatrice = ^Tab2D;
Var GogoGadgetAuTableau : PMatrice;
```

On a donc une variable GogoGadgetAuTableau (4 octets) qui pointe vers une autre variable (10 * 10 * 2 = 200 octets) de type Tab2D qui est un tableau de deux dimensions contenant 10 * 10 nombres entiers. Pour être précis, la variable GogoGadgetAuTableau est d'un type PMatrice pointant vers le type Tab2D. Donc la taille de GogoGadgetAuTableau sera de 4 octets, puisque contenant une adresse mémoire, et GogoGadgetAuTableau^ (avec un ^) sera la variable de type Tab2D contenant 100 nombres de type Integer.
On pourra affecter des valeurs à la variable comme suit :

```
GogoGadgetAuTableau^[i,j] := 3;
```

Toutes les opérations possibles concernant les affectations de variables, ou leur utilisation dans des fonctions, sont valables pour les variables pointées par des pointeurs.

Il est bien entendu impossible de travailler sur la variable pointée par le pointeur sans l'avoir auparavant allouée !

```
Program Exemple29c;
Const
  Max = 10;
Type
  Tab2D = Array [1..Max,1..Max] of Integer;
  PMatrice = ^Tab2D;
Var
  GogoGadgetAuTableau : PMatrice;
  i, j, x : Integer;
BEGIN
  New(GogoGadgetAuTableau);
  for i := 1 to Max do
    for j := 1 to Max do GogoGadgetAuTableau^[i,j] := i + j;
  x := GogoGadgetAuTableau^[Max,Max] * Sqr(Max);
  WriteLn(Cos(GogoGadgetAuTableau^[Max,Max]));
  Dispose(GogoGadgetAuTableau);
END.
```

Ce court programme *Exemple29c* montre qu'on utilise une variable pointée par un pointeur comme n'importe quelle autre variable.

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

```
Program Exemple29d;
Type
  Point = Record
            x, y : Integer;
            Couleur : Byte;
          end;
  PPoint = ^Point;
Var
  Pixel1, Pixel2 : PPoint;
BEGIN
  Randomize;
  New(Pixel1);
  New(Pixel2);
  with Pixel1^ do
    begin
      x := 50;
      y := 100;
      Couleur := Random(14) + 1;
    end;
  Pixel2^ := Pixel1^;
  Pixel2^.Couleur := 0;
  Dispose(Pixel1);
  Dispose(Pixel2);
END.
```

Dans ce programme *Exemple29d*, on déclare deux variables pointant chacune vers une variable de type Point, ce dernier étant un type structuré (enregistrement). La ligne d'instruction Pixel2^ := Pixel1^; signifie qu'on égalise champ à champ les variables Pixel1 et Pixel2.
Si les symboles ^ avaient été omis, cela n'aurait pas provoqué d'erreur mais cela aurait eu une tout autre signification : Pixel2 := Pixel1; signifierait que le pointeur Pixel2 prend la valeur du pointeur Pixel1, c'est-à-dire que Pixel2 pointera vers la même adresse mémoire que Pixel1. Ainsi, les deux pointeurs pointent vers le même bloc mémoire et donc vers la même variable. Donc Pixel1^ et Pixel2^ deviennent alors une seule et même variable.

On ne peut égaliser deux pointeurs que s'ils ont le même type de base (comme pour les tableaux).

Et dans ce cas, les deux pointeurs pointent éxactement vers la même variable.
Je rappelle qu'il est impossible de travailler sur la valeur pointée par le pointeur sans avoir utilisé auparavant la procédure New qui alloue l'adresse mémoire au pointeur. Si vous compilez votre programme sans avoir utilisé New, une erreur fatale vous rappellera à l'ordre !

```
Program Exemple29e;
Const
  Max = 10;
Type
  Personne = Record
               Nom, Prenom : String;
               Matricule : Integer;
             end;
  Tableau = Array [1..Max] of Personne;
  PTableau = ^Tableau;
Var
  Tab : PTableau;
  i : Integer;
BEGIN
  New(Tab);
  with Tab^[1] do
    begin
      Nom := 'Cyber';
      Prenom := 'Zoïde';
      Matricule := 1256;
    end;
  for i := 1 to Max do WriteLn(Tab^[i].Nom);
    { Seul l'élément d'indice 1 dans le tableau est initialisé }
    { Les éléments 2 à 10 contiennent n'importe quoi (rappel) }
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

```
  Dispose(Tab);
END.
```

Il est possible de combiner les enregistrements, les tableaux et les pointeurs. Cela donne un vaste panel de combinaisons. Essayons-en quelques unes :

```
Type TabP = Array [1..100] of ^Integer;
Var Tab : TabP;
   { Tableau de pointeurs pointant vers des entiers.  }
   { Tab[i] est un pointeur et Tab[i]^ est un entier. }

Type Tab = Array [1..100] of Integer;
     PTab = ^Tab;
Var Tab : PTab;
   { Pointeur pointant vers un tableau d'entiers.  }
   { Tab^[i] est un entier et Tab est un pointeur. }
```

GetMem et FreeMem

Il existe des procédures similaires au couple New et Dispose :

```
GetMem(Pointeur,Taille_Memoire);
```

Cette procédure réserve un nombre d'octets en mémoire égal à Taille_Memoire au pointeur Pointeur. Taille_Memoire est donc la taille de la variable pointée par le pointeur Pointeur.

```
FreeMem(Pointeur,Taille_Memoire);
```

Cette fonction supprime de la mémoire la variable pointée par le pointeur Pointeur, qui occupait Taille_Memoire octets.

Si vous utilisez New pour allouer une variable dynamique, il faudra utiliser Dispose et non pas FreeMem pour la désallouer.
De même, si vous utilisez GetMem pour l'allouer, il faudra utiliser FreeMem et non pas Dispose pour la désallouer.

```
Program Exemple29f;
Const
  Max = 10;
Type
  Personne = Record
               Nom, Prenom : String;
               Matricule : Integer;
             end;
  Tableau = Array [1..Max] of Personne;
  PTableau = ^Tableau;
Var
  Tab : PTableau;
  i : Integer;
BEGIN
  GetMem(Tab,Max*SizeOf(Personne));
  for i := 1 to Max do ReadLn(Tab^[i].Nom);
  FreeMem(Tab,Max*SizeOf(Personne));
END.
```

Vous aurez remarqué que ce programme *Exemple29f* est exactement le même que *Exemple29e*, mis à part qu'il utilise le couple GetMem et FreeMem au lieu des traditionnels New et Dispose. C'est un peu moins pratique à utiliser puisqu'il faut savoir exactement quelle place en mémoire occupe la variable pointée par le pointeur spécifié. Mais ça

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

peut être très pratique si Max = 90000 (très grand) et si vous décidez de faire entrer au clavier la borne supérieure du tableau.
Voir le programme suivant :

```
Program Exemple29g;
Const
  Max = 90000;
Type
  Personne = Record
               Nom, Prenom : String;
               Matricule : Integer;
             end;
  Tableau = Array [1..Max] Of Personne;
  PTableau = ^Tableau;
Var
  Tab : PTableau;
  i : Integer;
  N : LongInt;
BEGIN
  Write('Combien de personnes ? ');
  ReadLn(N);
  GetMem(Tab,N * SizeOf(Personne));
  for i := 1 To N do ReadLn(Tab^[i].Nom);
  FreeMem(Tab,N * SizeOf(Personne));
END.
```

Nil

Nous l'avons vu en long et en large, un pointeur contient une adresse. Mais comment indiquer qu'il ne contient... aucune adresse ? Il existe une adresse qui ne pointe vers rien : Nil (on rencontre souvent NULL dans d'autres langages). Il peut en effet être utile de tester qu'un pointeur pointe vers quelque chose ou bien vers rien, par exemple lorsque l'on travaille avec des **listes chaînées**. Pour schématiser la structure d'une liste chaînée, il s'agit d'une chaîne d'éléments contenant non seulement une valeur mais aussi l'adresse de l'élément suivant.
Exemple :

```
Type pElement = ^tElement;
     tElement = Record
                  Valeur : Integer;    { Valeur de l'élément }
                  Suivant : pElement; { Pointe vers l'élément suivant dans la liste chaînée }
                end;
```

Le dernier élément de la liste chaînée doit pointer vers... rien. On lui affectera donc la valeur Nil :

```
Var Element : pElement;

BEGIN
  ...
  New(Element);
  Element^.Valeur := ...;
  Element^.Suivant := Nil;
  ...
END.
```

Pour tester qu'un pointeur pointe vers quelque chose ou rien, c'est tout simple :

```
if p = Nil
   then
     { Pointe vers rien }
   else
     ...
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

Chapitre 25 - Ensembles

Les ensembles en Pascal sont les mêmes que ceux que vous connaissez en maths. Ils sont donc régis par les mêmes lois et nécessitent les mêmes opérateurs d'inclusion...

Déclarations

Un ensemble est une variable qui contient un nombre fini d'éléments de même type. Ce type doit être de type scalaire et ne pas avoir une taille supérieure à 1 octet. La déclaration d'une telle variable se fait par l'utilisation de la syntaxe Set of.
Syntaxe :

```
Type tId = Set of type_de_l_ensemble;
Var Identificateur : tId;
Var id1, id2, id3 : tId;
```

Exemples :

```
Type Aleatoires = Set of Byte;
Var Hasard : Aleatoires;
```

Ici, la variable Hasard est un ensemble de nombres entiers dont les valeurs possibles sont dans l'intervalle 0..255.

```
Type Binaires = Set of 0..1;
Var NbBin : Binaires;
```

Ici, la variable NbBin est un ensemble de 0 et de 1.

Comme pour tout autre type de variable, on peut déclarer en un seul bloc plusieurs variables "ensemble" du même type.

En général, on cherchera à utiliser des ensembles dont le type sera défini par le programmeur lui-même, c'est-à-dire différent des types de base du Pascal. C'est le cas des types énumérés.
Syntaxe :

```
Type MonType = (Element1, Element2, Element3...);
     Ensemble = Set of MonType;
```

Ou forme plus compacte :

```
Type Ensemble = Set of (Element1, Element2, Element3...);
```

Le type se résume à une liste d'éléments séparés par des virgules à l'intérieur d'une parenthèse comme le montre la syntaxe ci-dessus.
Exemple :

```
Type Prenom = (Boris, Hugo, Aurore);
     Club : Set of Prenom;
```

Ou bien :

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

```
Type Club = Set of (Boris, Hugo, Aurore);
```

Dans cet exemple, le type Club est un ensemble de type Prenom, c'est-à-dire que cet ensemble nommé Club ne peut contenir que les éléments déclarés dans le type Prenom. Donc une variable de type Club pourra contenir une combinaison de ces trois éléments.
Les ensembles ne sont pas ordonnés. Donc il n'existe pas d'ordre d'apparition des éléments dans une variable ensemble. On est seulement être capable de comparer le contenu de deux ensembles de même type, et de déterminer si un élément est inclus ou non dans un ensemble. De plus, un même élément n'apparaît qu'une seule fois dans un ensemble. Et il n'existe pas de fonction qui renvoie le nombre d'éléments d'un ensemble.

Affectations et opérations

Après avoir vu l'aspect déclaratif des ensembles, on va apprendre à les utiliser dans un programme. L'ensemble, quel que soit son type, peut être un ensemble **vide**. Pour donner une valeur à un ensemble, c'est-à-dire, spécifier le ou les élément(s) que devra contenir l'ensemble, on utilise l'opérateur habituel d'affectation :=. Ce qu'il y a de nouveau, c'est que le ou les élément(s) doivent être séparés par des virgules (comme dans la déclaration du type) et être entre crochets (contrairement à la déclaration).
Syntaxes :

```
Ensemble := [ ]; { ensemble nul }
Ensemble := [ Element ]; { ensemble constitué de l'élément élément }
Ensemble := [ Element5, Element1 ]; { ensemble constitué des éléments Element5 et Element1 }
```

Rappel : L'ordre des éléments dans une affection ou une comparaison n'a aucune espèce d'importance puisque les ensembles ne sont pas ordonnés.

Une affectation à un ensemble en supprime les éléments qu'il contenait avant l'affectation.

Si, en cours de programme, on souhaite ajouter ou supprimer un ou des élément(s) à l'ensemble, on doit utiliser les opérateurs additif + et soustractif - traditionnels.
Syntaxes :

```
Ensemble := Ensemble + [ ];
   { inutile car n'ajoute rien ! }

Ensemble := Ensemble + [ Element4 ];
   { rajoute l'élément Element4 à l'ensemble }

Ensemble := Ensemble + [ Element3, Element2 ];
   { rajoute les éléments Elément3 et Elément2}

Ensemble := Ensemble + [ Element1 ] - [ Element7, Element3 ];
   { rajoute l'élément Element1 et supprime les éléments Element7 et Element3 }
```

Pour être exact, les éléments entre crochets représentent des ensembles à part entière. Ces ensembles sont de même type que la variable ensemble auquel on ajoute ou supprime des sous-ensembles. Cela s'explique par le fait que l'on ne peut additionner que des variables de même type : on ne peut pas additionner éléments et ensemble, mais par contre on peut additionner entre eux des ensembles. Ainsi, un élément entre crochets est un ensemble et plusieurs éléments séparés par des virgules et entre crochets est aussi un ensemble.
Pour employer le vocabulaire mathématique approprié, + est l'opérateur d'**union**, - est l'opérateur de **complément** et on peut en rajouter un troisième : * est l'opérateur d'**intersection**.

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

Comparaisons

Le seul moyen de connaître le contenu d'un ensemble est de le comparer à d'autres du même type. Ainsi, les tests booléens par l'intermédiaire des opérateurs relationnels (voir **chapitre 2**) permettent de savoir si tel ou tel élément se trouve dans un ensemble, ou bien si tel ensemble est inclus dans un autre.
Les opérateurs relationnels stricts sont incompatibles avec les ensembles; ainsi, seuls ceux du tableau ci-dessous sont à utiliser avec les ensembles :

Symbole	Description
=	égale
<>	différent
<=	inclus
>=	contient

L'opérateur In permet de connaître la présence ou non d'un élément dans un ensemble sans avoir à passer par l'utilisation des crochets.
Exemples :

```
if [Element2] <= Ensemble then ...
   { si l'ensemble constitué de l'élément Element2 est inclus dans l'ensemble Ensemble alors...}

if Element2 in Ensemble then ...
   { si l'élément Element2 appartient à l'ensemble Ensemble alors... }

if Ensemble = [ ] then ...
   { si l'ensemble Ensemble est vide, alors... }

Test := not([Element7, Element4] <= Ensemble);
   { le booléen Test prend la valeur True si l'ensemble constitué des éléments Element7
     et Element4 n'est pas inclus dans l'ensemble Ensemble et False dans l'autre cas }

if (Ensemble1 * Ensemble2) = [ ] then ...
   { si l'intersection des ensembles Ensemble1 et Ensemble2 est vide, alors...
     c'est qu'ils n'ont aucun élément en commun malgré qu'ils soient de même type }

if (Ensemble1 - Ensemble2) = [ ] then ...
   { si le complément des ensembles Ensemble1 et Ensemble2 est vide, alors...
     c'est que Ensemble1 est contenu dans Ensemble2 }
```

Il est impossible d'utiliser les procédures Write(Ln) et Read(Ln) avec les variables de type ensemble.

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

Chapitre 26 - Constantes

Dans un programme, il est indispensable d'utiliser des constantes si on veut écrire du code aisément maintenable et modifiable. Il faut systématiquement faire la chasse aux "nombres magiques", qui sont les valeurs numériques apparaissant en clair dans le code. De plus, il est souvent nécessaire d'avoir des valeurs de référence ou des conditions initiales, au lieu d'affectations en début de programme. Alors, pour alléger le code et pour améliorer la lisibilité et surtout la compréhension générale du programme, il est préférable d'avoir recours à des **constantes**.
Une constante, pour être connue de tous les sous-programmes, doit être déclarée avant ces derniers. Si une constante est déclarée après une fonction, pour que cette fonction puisse l'utiliser, la constante doit être passée en paramètre à la fonction. Une constante déclarée avant tous les sous-programmes n'a pas besoin d'être passée en paramètre à ces derniers pour y être utilisée (c'est le même principe que pour les variables, fonctions, procédures paramétrées et types).
Comme son nom l'indique, une constante ne change pas de valeur au cours du programme, une valeur de départ lui est affectée dès sa déclaration avant le code exécutable. C'est l'opérateur = qui affecte une valeur à une constante après que cette dernière se soit vue affecter un identificateur par l'utilisation du mot réservé Const.
Syntaxe :

```
Const identificateur_de_la_constante = valeur;
```

Exemples :

```
Const Nom = 'CyberZoïde';
Const TVA = 20.6;
Const Esc = #27;
```

Le compilateur décide automatiquement d'affecter à la constante le type de base compatible le plus économique. Ceci est totalement transparent au programmeur. Par exemple, lorsqu'il trouve un nombre réel affecté à une constante, il lui spécifie le type Real. Ou encore, si vous affectez un caractère à une constante, celle-ci sera de type Char et non de type String. Mais il vous est possible de forcer le type de la constante par les deux points : que vous connaissez déjà pour les variables. De plus, cette déclaration de type à une constante en plus de sa valeur est dans certains cas indispensable, si vous souhaitez leur donner une valeur qui n'est pas d'un type de base du compilateur (par exemple un ensemble ou un type que vous avez vous-même défini avant la constante).
Syntaxe :

```
Const identificateur : type_de_la_constante = valeur;
```

Exemples :

```
Const Nmax : LongInt = 60000;
Const Defaut : String = '*';
```

Les constantes peuvent être de tout type sauf de type fichier.

On peut tout aussi bien affecter aux constantes des **expressions** dont le résultat sera évalué à la compilation.
Syntaxe :

```
Const identificateur = expression;
```

Exemples :

```
Const Esc = Chr(27);
```


Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

```
Const Taux = (exp(10) - (3 * Pi)) / 100;
Const Nom = 'Cyber'+'Zoïde';
Const Softi : Integer = Length(Nom);
```

La déclaration des constantes se fait dans la partie déclarative du programme, avant le code exécutable (tout comme pour la déclarations des fonctions, procédures, variables). Pour être utilisables par tous les sous-programmes (procédures paramétrées...), les constantes doivent nécessairement êtres déclarées avant ces derniers.

```
Program Exemple40;

Const
  Nom = 'Bill';

Procedure SaisieLibre (var Toi : String);
Begin
  Write('Entrez votre nom : ');
  ReadLn(Toi);
  if Toi = '' then Toi := Nom;
End;

Procedure SaisieContrainte (var Toi : String);
Begin
  repeat
    Write('Entrez votre nom : ');
    ReadLn(Toi);
  until Toi <> '';
End;

Var
  Toi : String;

BEGIN
  SaisieLibre(Toi);
  SaisieContrainte(Toi);
END.
```

Ce programme *Exemple40* montre quelle peut être l'utilisation d'une constante. Ici, la procédure SaisieLibre permet à l'utilisateur de ne pas dévoiler son nom, la variable toi prenant une valeur par défaut qui est celle de la constante Nom. Quant à la procédure SaisieContrainte, elle ne laisse guère le choix à l'utilisateur : il doit absolument entrer un nom pour poursuivre le programme.

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/

Tests d'évaluation

Un test d'évaluation vous permet de voir si vous avez assimilé les bases de la programmation en Turbo Pascal 7.0. Il y a en tout 5 pages contenant chacune 20 questions à choix multiple, soit pas moins de **100 questions** !
Pour obtenir instantanément vos résultats ainsi que la correction, il vous suffit de cliquer sur le bouton de validation au bas de chaque page. Il faut préalablement avoir répondu à toutes les questions de la page.
Pour accéder à la première page de test : **cliquez ici**.

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2001-2013 Hugo Etievant. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
http://pascal.developpez.com/cours/cyberzoide/