Programmation Turbo Pascal / Delphi – infos supplémentaires

Les points-virgules (';')' et begin/end

Significations :

- Le point-virgule sert à séparer les instructions. Théoriquement, on peut écrire le programme entier dans une seule ligne.
- Les mots **begin** et **end** servent à regrouper des instructions, comme des parenthèses.

Règles :

- à la fin du programme derrière le dernier **end**, se trouve un point ('.') et pas de ';'.
- il faut séparer toutes les instructions par un ';'. En principe on place le ';' après chaque instruction, à la fin de la ligne.
- pas de ';' après **begin**, pas de ';' avant **end** (ils sont permis mais pas nécessaires puisque **begin** et **end** ne sont pas des instructions, ils ont la fonction de parenthèses).
- pas de ';' après **repeat**, pas de ';' avant **until** (ils sont permis mais pas nécessaires puisque **repeat** et **until** ont aussi la fonction de parenthèses).
- pas de ';' avant **then**, **do**, **to** et (surtout) **else** sinon on coupe ces éléments de la structure dont ils dépendent (**if**, **for** ou **while).** Seul exception pour **else** : dans l'instruction **case** qui ne figure pas au programme.
- pas de ';' après **then**, **else** ou **do** (dans les structures **if**, **for** ou **while**). Théoriquement, elles sont permises. Or, le ';' servant à séparer les instructions, un ';' après **then** par exemple marquerait la fin de la structure **if**.

Exemples d'erreurs :

```
if A<0 then;
   writeln('A est négatif')
```

Dans ce cas le programme affiche toujours "A est négatif", même si A est positif.

```
for I:=1 to 10 do;
  writeln('Bonjour')
```

Dans ce cas le programme fait 10 fois rien, puis affiche une seule fois "Bonjour".

```
if A<0 then
   writeln('A est négatif');
else
   writeln('A est positif')
```

Dans ce cas le compilateur annonce une erreur. Puisque la ';' marque la fin de l'instruction **if**, le mot **else** n'a donc plus de **if** à qui il correspond.

```
if A<0 then
  writeln('Le nombre est négatif. ');
  writeln('Racine carrée pas possible.')
```

Dans ce cas le programme affiche toujours "Racine carrée pas possible", car l'instruction s'arrête après le ';'.
Pour corriger, il faut entourer les deux instructions d'affichage de **begin** et **end**.

Notions de base

Noms des identifiants (variables, programme, constantes, procédures, fonctions… :
Choisir un nom significatif composé de lettres, chiffres et '_', pas d'accents. Commencer avec une lettre.
Par convention : identifiants en majuscules, mots clefs du langage Pascal en minuscules.

Formatage dans write(ln) :

d'une chaîne de caractères	writeln(S : <minimum>)
d'un entier	writeln(I : <minimum>)
d'un réel en notation normale	writeln(R : <minimum> : <déc>)
	writeln(15.60:6:1) affiche `··15.6`
	writeln(15.6:3:3) affiche `15.600`
d'un réel en notation scientifique	writeln(R : <largeur>)
	writeln(15.60:10) affiche `·1.560E+01`

Description (réel en notation scientifique)	**Turbo Pascal**	**Delphi**
Format *minimal* (·=espace ou '–', e…=exposant)	`·x.xE±ee`	`·x.xE±eeee`
Nombre *minimal , par défaut , maximal* de décimales	1 , 10 , 10	1 , 14 , 17
Largeur *minimale , par défaut , maximale*	8 , 17 , 17	10 , 23 , 26

Opérateurs, priorités :

1	**not**, signe +, signe -	2	***, /*, div, mod, and**	3	**+, -, or, xor**	4	**=, <, >, <>, <=, >=**

Dans le cas de deux opérateurs de la même priorité, celui le plus à gauche l'emporte sur l'autre.

Opérateurs, division avec reste :

Soient A **div** B = D et A **mod** B = M. Alors B ***** D **+** M = A. Opérateurs **div** et **mod** interdits pour les réels !
10 **mod** 3 = 1, 10 **mod** -3 = 1, -10 **mod** 3 = -1, -10 **mod** -3 = -1, 10 **div** -3 = -3, -10 **div** 3 = -3, -10 **div** -3 = 3

Opérateurs logiques :

A	B	A or B	A and B	A xor B	A nand B	not A	Algèbre logique
true	**true**	true	true	false	false	false	**not not** A = A
true	**false**	true	false	true	true	false	
false	**true**	true	false	true	true	true	**not** (A **and** B) = **not** A **or not** B
false	**false**	false	false	false	true	true	**not** (A **or** B) = **not** A **and not** B

Fonctions système avec un seul paramètre numérique :

résultat réel	**sqrt, sin, cos, arctan, ln, exp, frac, int**
résultat entier	**round, trunc, random**
résultat même type que paramètre	**abs, sqr**

Nombres aléatoires :

Uutiliser **randomize** une seule fois au début du programme pour initialiser les nombres aléatoires.

0<=**random**<1 *(réel)*	0<=**random**(N)<N *(entier)*	A<=**random**(B-A+1)<=B *(entier)*

Opérations sur les caractères :

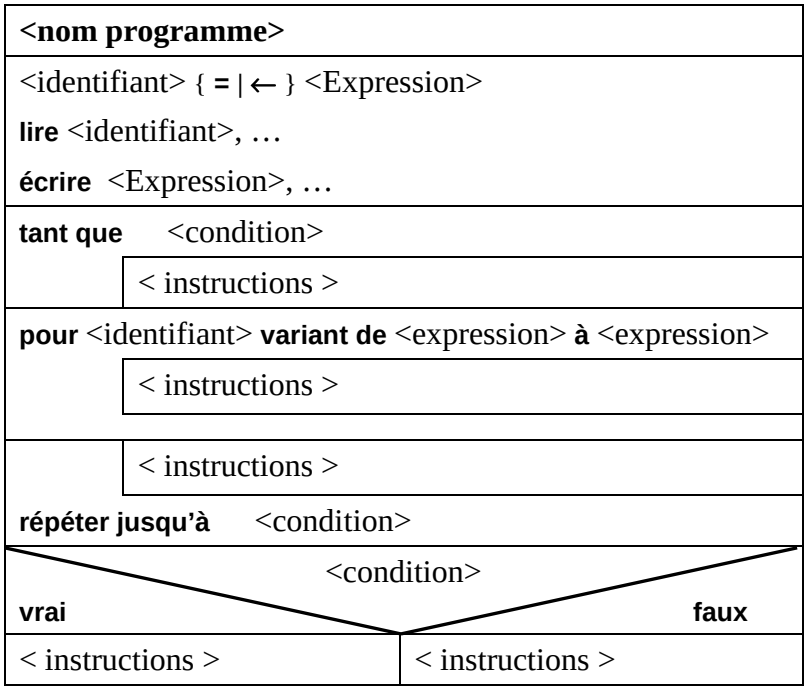
Accès à une cellule	<identifiant>[<position>]	ex : S[2]
Concaténation de deux chaînes	<chaîne 1> + <chaîne 2>	ex : S+' '+S2
Longueur d'une chaîne	*Length*(<chaîne>)	
Recherche d'une sous-chaîne	*Pos*(<sous-chaîne>, <chaîne>)	
Extraction d'une sous-chaîne	*Copy*(<chaîne>, <indice>, <nombre>)	
Suppression d'une sous-chaîne	Delete(<chaîne>, <début>, <nombre>)	
Insertion d'une sous-chaîne	Insert(<sous-chaîne>, <chaîne>, <position>)	
Conversion caractère en code	*Ord*(<car>)	
Conversion code en caractère	*Chr*(<code>)	
Conversion nombre en chaîne	Str(<entier>,<identifiant>)	
Conversion chaîne en nombre	Val(<chaîne>,<identifiant>,<code_erreur>)	

Delphi : les fonctions IntToStr et StrToInt remplacent les instructions Str et Val

Conversion nombre en chaîne	*IntToStr*(<entier>) et *FloatToStr*(<réel>)
Conversion chaîne en nombre	*StrToInt*(<chaîne>) et *StrToFloat*(<chaîne>)

Remarques : les fonctions sont formatées en *italique*, les procédures ne le sont pas
les paramètres soulignés sont des identifiant dont la valeur est modifiée par la procédure
<code_erreur> : position du premier caractère erroné, 0 si pas d'erreur
position, indice, entier, début, code_erreur et code sont des **entiers**

Le structogramme



- *pas de déclaration de variables ou de constantes*
- *une instructions par ligne (pas de* **';'***)*
- *groupement des instructions par des cadres/traits (pas de* **begin/end***)*
- *lecture succinctes (une instruction* **lire A** *peut être traduite par* **write('Entrez un nombre');readln(A);**
- *affichage succincte (une instruction* **écrire A** *peut être traduite par* **write('Le résultat est ',A);**

Liste des Composants

Liste des composants, propriétés, événements et méthodes à connaître pour l'épreuve en informatique à l'examen de fin d'études secondaires techniques - division technique générale

Composant	*Préfixe*	Propriétés	Evénements	Méthodes
Tous les composants visuels		Name, Width, Height, Top, Left, Enabled, Visible		
Certains composants visuels		Hint, ShowHint, Font, Color, Alignment, Align	OnClick	SetFocus
TForm	*frm*	Caption, ClientWidth, ClientHeight	OnCreate, OnClose	Close
TButton	*btn*	Caption		
TEdit	*edt*	Text	OnChange	Clear, SelectAll
TLabel	*lbl*	Caption		
TPanel	*pnl*	Caption		
TImage	*img*	Picture, Autosize, Canvas	OnMouseDown, OnMouseMove, OnMouseUp	Picture.LoadFromFile, Picture.SaveToFile
TTimer	*tm*	Interval	OnTimer	
TSpinEdit	*spe*	Value, MaxValue, MinValue, Increment	OnChange	
TMenu	*mm*			
TMenuItems	*mmi*	Caption, Checked		
TOpenDialog, TSaveDialog	*dlg*	FileName, Filter, DefaultExt		Execute
TFontDialog	*dlg*	Font		Execute
TColorDialog	*dlg*	Color		Execute
TStrings (applicable à Items)	-	Count, Strings[ ]		Clear, Append, Insert, Delete, LoadFromFile, SaveToFile
TListBox	*lb*	Items (voir TStrings), ItemIndex, Sorted		
TComboBox	*cb*	Items (voir TStrings), Text, ItemIndex, Sorted	OnChange	
TCanvas	-	Pen.Color, Pen.Width, Brush.Color, Pixels[ ]		LineTo, MoveTo, Rectangle, Ellipse, TextOut
TPaintBox	*pb*	Canvas	OnMouseDown, OnMouseMove, OnMouseUp, OnPaint	RePaint
TStringGrid	*sg*	Cells[], Col, Row, FixedCols, FixedRows, RowCount, ColCount, Option - goEditing		
TRadioGroup	*rg*	Items (voir TStrings), ItemIndex		

Fonctions / Procédures	
Manipulation de chaînes de caractères	Copy, Length, Pos, Delete, Insert, + (concaténation), LowerCase, UpperCase
Conversion de types	IntToStr, FloatToStr, StrToInt, StrToFloat
Fonctions mathématiques	Abs, Sin, Cos, Exp, Ln, Sqr, Sqrt, Round, Trunc, Random
Affichage de messages	ShowMessage

Syntaxe de fonctions et procédures agissant sur des chaînes de caractères	
Copy(<chaîne>, <position début>, <nombre de caractères>)	*(fonction de type string)*
Pos(<sous-chaîne>, <chaîne>)	*(fonction de type integer)*
Delete(<chaîne>, <position début>, <nombre de caractères>)	*(procédure)*
Insert(<sous-chaîne>, <chaîne>, <position d'insertion>)	*(procédure)*

Valeurs de paramètres d'événements
Shift: *ssShift, ssCtrl, ssAlt, ssLeft, ssRight, ssMiddle, ssDouble*
Button: *mbLeft, mbRight, mbMiddle*

Liste de Propriété/Méthodes supplémentaires

Liste des composants, propriétés, événements et méthodes à connaître pour les devoirs en classes

Composant	*Préfixe*	Propriétés	Evénements	Méthodes
Certains composants visuels		Font.Color	DblClick	
TButton	*btn*	Cancel, Default		
TEdit	*edt*	MaxLength, PasswordChar		
TImage	*img*	Stretch		
TSpinEdit	*spe*	EditorEnabled, MaxLength ?		
TComboBox	*cb*	Style (*Simple*, *DropDown*, *DropDownList*)		
TRadioGroup	*rg*	Caption		
TStringGrid	*sg*		SetEditText, SelectCell	
TCheckbox	***chb***	Caption, Checked		

Paramètres d'événements	
Click	····· (Sender: TObject)
MouseMove	····· (Sender: TObject; Shift: ShiftState; X,Y: Integer)
MouseDown, MouseUp	····· (Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Integer)
KeyPress, KeyDown, KeyUp	····· (Sender: TObject; var Key: Word; Shift: ShiftState)
SetEditText	····· (Sender: TObject; ACol, ARow: Integer; const Value: String)
SelectCell	····· (Sender: TObject; ACol, ARow: Integer; var CanSelect: Boolean)

Remarques supplémentaires	
*Soient **lb, edt, pnl, sg** etc. des composant de type **TListBox, TeditText, TPanel, TStringGrid** etc. et **S** de type **string***	

insert('X', edt.Text, 3);
incr(lb.ItemIndex); *ou bien*
delete(edt.Text,1,1);
	ERROR: "types of actual and formal var parameters must be identical".
	ERROR: "constant object cannot be passed as var parameter".
	Ne peut pas passer un attribut comme paramètre par référence (var) !

edt.Text[2] := 'A'; I := ord(copy(S,1,1));	ERROR: "Left side cannot be assigned to" ERROR: "incompatible types". copy rend un string, ord demande un char. I := ord(S[1]) fonctionne
I := StrToInt(''); (string vide) S := IntToStr(3.2); (3.2 est réel) S := copy('Bonjour',20,3); S := 'Bonjour'; insert('x',S,20); S := 'Bonjour'; delete(S,20,3);	Runtime ERROR: '' is not a valid Integer value ERROR: "There is no overloaded version of 'IntToSTr' that can be called…" : le paramètre doit être un string avec un entier correct rend un string vide ajoute le 'x' à la fin : 'Bonjourx' laisse S inchangé

lb.Items.Insert(lb.Items.Count+1, S);
lb.Items.Insert(lb.Items.Count, S);
lb.Items.Insert(-1, S);
lb.Items.Insert(-2, S);
	ERROR: "unable to insert a line"
	ajoute le texte S en dernière position
	ajoute le texte S en première position
	ERROR: "unable to insert a line"

Focus : Dans une fiche, il y a toujours un seul composant qui détient le focus. En général ce sont les composants de saisie comme bouton, combobox, champ d'édition, groupe à options etc.	• Si une forme ne contient aucun composant pouvant recevoir le focus, alors la fiche détient elle-même le focus. S'il existe un autre composant qui peut détenir le focus, alors la fiche ne le détient pas. • Les composants de type TForm, TLabel et TPanel ne peuvent pas recevoir le focus avec la méthode setfocus. • C'est le composant détenant le focus qui réagit à l'événement OnKey…
SpinEdit	• Attention : si MinValue=MaxValue, alors le spinedit n'est plus limité ! • Quand on donne à MaxValue une valeur plus petite que Value, alors Value ne change pas tout de suite mais seulement après l'événement OnChange.
Button et Shift :	• Le paramètre *Button* indique le bouton qui a déclenché l'événement **MouseDown** ou **MouseUp**. • Le paramètre *Shift* indique quel bouton est appuyé actuellement. • **Exemple MouseDown** : *ssLeft in Shift* indique que le bouton gauche est appuyé actuellement mais on ne sais pas s'il a déclenché l'événement • **MouseMove** : n'a pas de paramètre *Button* car ce n'est pas un bouton qui déclenche cet événement

StringGrid	• Contrairement aux listes, adresser une cellule en dehors de la grille définie par RowCount et ColCount ne produit pas d'erreur. Une valeur écrite hors de l'espace visible devient visible quand l'espace est aggrandi. • Si on veut connaître les coordonnées de la cellule cliquée, on peut ou bien utiliser l'événement **OnSelectCell** qui fourni deux paramètres ARow e t ACol (pas sur programme de l'examen) ou bien utiliser l'événement **OnClick**, puis trouver les coordonnées dans les attributs sg.Col et sg.Row. • Le seul moyen de vider une StringGrid est une double boucle.

Listes – exemples

*soit **lb1** un composant du type **ListBox***	
lb1.ItemIndex := 0;	sélectionner le premier élément de la liste
lb1.ItemIndex := -1;	faire en sorte qu'aucun élément la liste soit sélectionné
lb1.ItemIndex := lb1.Items.Count-1;	sélectionner le dernier élément de la liste
lbl.Sorted := True; lbl.Sorted := False;	trier la liste, mais accepter par après des ajouts non-triés
lb1.Clear;	vider la liste
lb1.Items.Add('Paris');	ajouter le texte "Paris" comme dernier élément de la liste
lb1.Items.Delete(1);	supprimer le deuxième élément de la liste
lb1.Items.Insert(0,'0');	insérer le texte "0" comme premier élément de la liste
lb1.Items.LoadFromFile('C:\log.txt');	Remplir la liste avec le contenu du fichier C:\log.txt. Chaque ligne du fichier deviendra un élément de la liste.
lb1.Items.SaveToFile('C:\log.txt');	La liste est enregistrée dans le fichier texte C:\log.txt. Chaque élément de la liste deviendra une ligne du fichier.
lb1.Items[0] := edt1.Text;	le premier élément de la liste est remplacé par le texte du champ d'édition edt1
ShowMessage(lb1.Items[2]);	afficher par un message à l'écran le troisième élément de la liste

Images – exemples & remarques

*soit **img** un composant du type **Image***	
img.Picture.LoadFromFile('ecole.bmp');	Charger l'image 'ecole.bmp'.
img.Picture := nil;	Effacer l'image affichée dans *img*.
img.Stretch := true;	Adapter l'image au cadre. (Ne modifie pas la taille du cadre.)
img.AutoSize := false;	Ne plus adapter le cadre à l'image. Le cadre gardera toujours la taille actuelle (mais ne revient pas aux dimensions initiales si elles ont changé).
uses (… ,) jpeg;	Permettre à Delphi le chargement d'une image de format jpeg.
img.Canvas.Rectangle(x1,y1,x2,y2)	Fonctionne avec n'importe quelles coordonnées de coins opposés du rectangle.
curseur	MoveTo et LineTo (dé)placent un curseur virtuel sur le canvas. Au début du programme, les coordonnées du curseur sont (0,0) les procédures Rectangle et Ellipse ne modifient pas le curseur.
persistence des couleurs etc.	Les valeurs de **img**.Pen.Color etc. sont persistantes tandis que les valeurs de **pb**.Pen.Color etc. sont perdues après chaque événement.
Width et ClientWidth Heigth et ClientHeigth	* pour les composants img et pb Width=ClientWidth etc. * en définissant p.ex. Width=200, la largeur est de 201 (des pixels 0 à 200)
img.Canvas.MoveTo(img.ClientWidth,0); img.Canvas.LineTo(0,img.ClientWidth);	Tracer une diagonale du coin en haut à droite vers le coin en bas à gauche.

version du 24 février 2010

Syntaxe Turbo Pascal

<Programme>	**program** <Identifiant> **;** [<Bloc Uses> **;**] [<Bloc Const> **;**] [<Bloc ProcFunc> **;**] [<Bloc Var> **;**] **begin** <Suite d'instructions> **end.**
<Bloc Uses>	**uses** <Unit> [**,** <Unit> **…**]
<Bloc Const>	**const** <Identifiant> **=** <Expression> [**;** <Identifiant> **=** <Expression> **…**]
<Bloc Var>	**var** <Décl Var> [**;** [**var**] <Décl Var> **…**]
<Décl Var>	<Identifiant> [**,** <Identifiant> **…**] **:** <Type>
<Type>	{ **integer	real	char	boolean	string** }
<Identifiant>	*Un nom de variable, de constante, de programme, de procédure, de fonction…*
<Bloc d'instructions>	{ <Instruction>	**begin** <Suite d'instructions> **end** }
<Suite d'instructions>	<Instruction> [**;** <Instruction> **…**]
<Instruction>	{ <Affectation> **|** <Procédure> **|** <Bloc d'instructions> **|** <Bloc If> **|** <Bloc While> **|** <Bloc For> **|** <Bloc Repeat> }
<Bloc If>	**if** <condition> **then** <Bloc d'instructions> [**else** <Bloc d'instructions>]
<Bloc while>	**while** <condition> **do** <Bloc d'instructions>
<Bloc for>	**for** <Identifiant> **:=** <Expression> { **to	downto**} <Expression> **do** <Bloc d'instructions>
<Bloc repeat>	**repeat** <Suite d'instructions> **until** <condition>
<Condition> *(synonyme de :)* <Expression logique>	<Condition> {**and	or	xor**} <Condition> *ou bien* **not** <Condition> *ou bien* **(**<Condition>**)** *ou bien* <Expression> {**<**	**>**	**<=**	**>=	=	<>** } <Expression>
<Affectation>	<Identifiant> **:=** <Expression>
<Expression>	<Expression> { **+**	**-**	*****	**/**	**mod**	**div** } <Expression> *ou bien* { **(**<Expression>**)**	<Identifiant> **|** <Constante> **|** <Fonction> }
<Procédure> *(instcruction élémentaire)*	**write**[**ln**]**(**<Expression> [**,** <Expression> **…**] **)** *ou bien* **read**[**ln**]**(**<Identifiant> [**,** <Identifiant> **…**] **)** *ou bien* { **randomize**	**clrscr**	*etc* }
<Fonction>	*par exemple* **sqr(**<Expression>**)**, **abs(**<Expression>**)** *etc*
<Bloc ProcFunc>	<Décl ProcFunc> [**;** <Décl ProcFunc> **…**]
<Décl ProcFunc>	{ **procedure** <Identifiant>**(**[[**var**] <Décl Var> [**;** [**var**] <Décl Var> **…**]]**)** **| function** <Identifiant>**(**[[**var**] <Décl Var> [**;** [**var**] <Décl Var> **…**]]**) :** <Type>**};** [<Bloc Var> **;**] **begin** <Suite d'instructions> **end**

Remarques :

Les éléments entre crochets [ ] sont optionnels. Les éléments entre accolades { } et séparés par des pipes ('|') représentent un ensemble d'éléments parmi lesquels il faut choisir un seul.

Le mot clé **downto** ne figure pas au programme de l'examen.

Dans <Programme>, les variables déclarées dans un <Bloc Var> placé au-dessus de <Bloc ProcFunc> sont globales (et donc connues à l'intérieur des sous-programmes).