DELPHI 7

Support de formation (1/3)

Initiation à Delphi

1 AVERTISSEMENT

Ce support est un document d'initiation dont le contenu se limite volontairement aux parties étudiées et ne saurait se substituer à la notice du concepteur du logiciel. La base utilisée est Delphi6, édition Professionnelle: il est évident que l'édition "Entreprise" comporte davantage de fonctionnalités qui ne seront pas abordées par ce document.

La connaissance préalable d'un langage de programmation structurée (de préférence Pascal) est fortement recommandée, mais non indispensable.

Conventions:

- le texte entre crochets représente une option facultative,
- le texte en italique des lignes de programme

2 INTRODUCTION

Delphi est un système de développement visuel rapide sous Windows (**R**apid **A**pplication **D**evelopment) qui permet de créer des applications fenêtrées directement exécutables (.EXE) et redistribuables librement sous Windows ou DOS. Il est à noter qu'i existe une version pour Linux nommée KYLIX (compatible avec les composants CLX, il suffit de recompiler). Sa simplicité d'emploi autorise une utilisation immédiate, car il suffit de cliquer-glisser des composants dans une fiche et de gérer quelques événements pour créer des applications simples. Des assistants, modèles et tuteurs interactifs facilitent la prise en main du logiciel.

Delphi utilise le langage Pascal Orienté Objet (il est toutefois possible d'utiliser d'anciennes sources en Pascal standard grâce au compilateur en ligne de commande). Ce langage est facile à apprendre et beaucoup plus simple que le C++ traditionnel. Les objets utilisés ont des propriétés et des méthodes. Les propriétés sont les caractéristiques de l'objet (couleur, taille, ...) tandis que les méthodes sont les procédures (classiques ou événementielles) et fonctions qui y sont rattachées.

Des outils puissants sont rattachés à Delphi. Le canevas facilite les graphismes et évite l'appel aux API de Windows (cette solution reste évidemment possible). Les principales boîtes de dialogue de Windows existent en tant qu'objets et sont facilement réutilisables. L'utilitaire BDE permet la création et l'accès aisé aux bases de données de tous types. La liaison avec ACCESS se fait directement, grâce à un driver natif intégré.

Il est facile d'écrire des DLL utilisables par n'importe quel autre programme sous Windows. Ces DLL peuvent contenir des fiches Delphi, des procédures, des données ou des ressources graphiques (bitmaps, icônes, curseurs).

La création d'objets (visuels ou non) réutilisables est un gain de temps appréciable pour le développement. L'intégration des objets Active X peut compléter la bibliothèque existante.

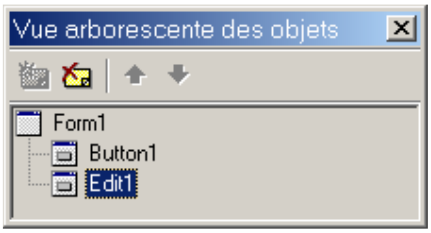
Il existe d'autres systèmes de développement rapide sous Windows. Delphi est particulièrement bien placé:

- moins de lignes de code et rapidité de compilation
- possibilité d'utiliser des procédures événementielles partagées
- notion de modèles réutilisables (fiches, menus, objets)
- orientation objet totale et native
- richesse des composants fournis: pas d'OCX à acheter en complément
- assembleur intégré, compilateur en ligne de commande
- débogage facile au niveau du source et du processeur
- possibilité d'allocation dynamique de la mémoire en utilisant les pointeurs

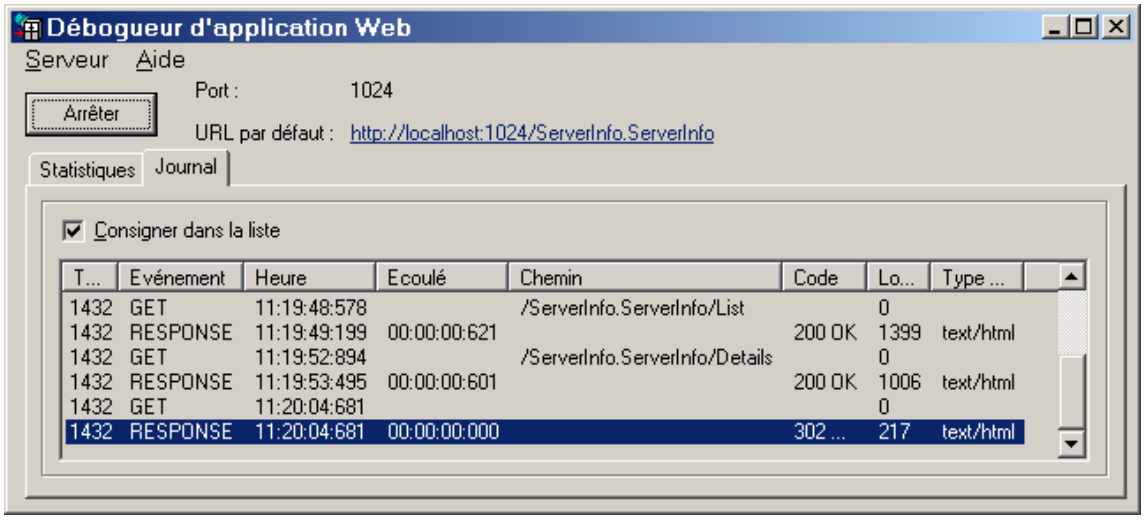
3 Nouveautés

Consulter l'aide en ligne sous "nouveautés" On note des:

- la vue arborescente des objets



- le débogueur d'applications WEB, serveur WEB



- de nombreux nouveaux composants dont
  - "TActionManager" qui permet de centraliser les procédures événementielles de chaque fenêtre
  - les composants CLX qui peuvent être portés sur LINUX
  - les composants ADO

l

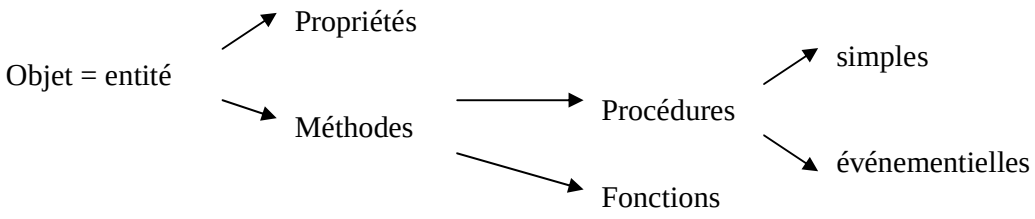
4 PROGRAMMATION ORIENTEE OBJET

4.1 PRESENTATION

Sans le savoir, vous utilisez des objets dans votre ordinateur. Par exemple, vous désirez changer la carte graphique. C'est un objet: vous ne connaissez pas exactement son fonctionnement interne mais vous désirez des résultats. Vous pouvez interchanger facilement cet objet puisque l'interface est standard (connecteurs). Vous n'avez donc pas besoin de changer toute l'unité centrale ou de faire des études poussées en électronique pour ce faire. Il suffit d'échanger la carte par une autre plus performante.

L'avantage de cette notion d'objet est la possibilité de partager les composants avec d'autres utilisateurs. Vous pouvez donner votre ancienne carte vidéo à l'un de vos collaborateurs. D'autre part, il n'est pas nécessaire de connaître le détail du fonctionnement de l'objet pour l'utiliser. La maintenance est facilitée, il suffit d'échanger l'objet sans toucher au reste de l'application.

4.2 CONCEPTS DE BASE



4.2.1 Propriétés

Ce sont les composants de l'objet. Sa taille, sa couleur, son nom, ... sont des propriétés. Les propriétés sont initialisées à la création de l'objet et certaines peuvent être modifiées par programmation. Delphi possède un éditeur de propriétés qui permet de les modifier en phase de conception de projet.

4.2.2 Méthodes

Les méthodes relèvent du comportement de l'objet: elles recensent ce que l'objet "sait" faire. S'il sait par exemple faire une multiplication, le nom du module de programme concerné est appelé méthode.

4.2.2.1 Fonctions

Les fonctions sont des modules (ou sous-programmes) qui retournent une seule valeur ou résultat après acceptation d'un certain nombre d'arguments (0 à n).

4.2.2.2 Procédures simples

Les procédures sont des modules qui effectuent des actions. Elles acceptent également des arguments. Elles peuvent retourner des arguments modifiés (s'ils sont déclarés en donnée-résultat).

4.2.2.3 Procédures événementielles

Elles conditionnent les réactions de l'objet vis-à-vis des actions extérieures (clavier, souris, ...). Ce sont des procédures appelées par un déclencheur. Windows recense un grand nombre d'événements dont la nature est précisée par un système de messages internes. Par exemple, le fait d'appuyer sur une touche du clavier ou de déplacer la souris provoque l'émission d'un message interne à Windows qui pourra être traité par la procédure événementielle.

4.2.3 Programmation structurée

Contrairement au premiers langages (basic) qui utilisaient des étiquettes avec des instructions de saut de type "goto" où l'on risquait un enchevêtrement dans le déroulement du programme, la POO demande une programmation modulaire et le langage Pascal est particulièrement adapté. L'écriture d'un programme se fait sous forme de blocs hiérarchisés, ce qui permet une meilleure compréhension. L'indentation des modules donne une lecture proche du développement algorithmique.

4.2.4 L'encapsulation

C'est le concept de base de la POO. L'idée est de rassembler les données et les modules de programme dans une seule et même structure appelée objet. Les données sont appelées "propriétés" et les modules sont les "méthodes". On peut utiliser des méthodes de l'objet sans connaître le détail de l'implémentation de chacune d'elle. L'encapsulation protège les données contre leur utilisation par d'autres modules extérieurs.

4.2.5 L'héritage

Les objets sont capables d'être dérivés à partir d'objets préexistants. Les objets "fils" héritent les propriétés et méthodes de leur parent de façon automatique. Il suffit alors de rajouter des propriétés et méthodes spécifiques: ce procédé est appelé programmation par exception. De plus, tous les changements effectués dans l'objet "parent" sont répercutés automatiquement sur les objets "descendants" sans manipulation supplémentaire.

4.2.6 Le polymorphisme

Le nom d'une méthode existante peut être réutilisé dans un objet "fils" en déclarant une implémentation différente. Le polymorphisme est la capacité de désigner des méthodes différentes (bien que portant le même nom) en fonction de l'objet concerné. Par exemple le signe "+" peut servir à l'addition, à la concaténation ou à l'incrémentation d'une date.

Dans le cas de redéfinition(OVERRIDE) d'une méthode, il est possible de réutiliser le code hérité (INHERITED) et d'ajouter simplement quelques lignes (qui pourront par exemple redéfinir certaines propriétés).

4.2.7 Développeur ou utilisateur?

On distingue deux catégories de personnes. Les développeurs qui connaissent bien la POO et qui créent des objets et les utilisateurs qui construisent leurs applications à partir de ces objets préfabriqués.

4.3 LEXIQUE

La source principale de confusion pour les débutants en POO est le grand nombre de nouveaux termes techniques

4.3.1 Classe (CLASS)

C'est l'ensemble des définitions, propriétés et méthodes qui serviront à créer l'objet. Elle définit la structure de l'objet et quelles en seront les propriétés. Ses modifications seront répercutées sur les classes descendantes.

4.3.2 Objet (OBJECT)

C'est une instance de la classe. Il a une existence physique et occupe de la place en mémoire. Il est créé à partir d'une classe. Toutes les modifications effectuées en exécution n'affectent en rien les descendants de sa classe.

4.3.3 Instanciation

C'est le processus qui crée en mémoire un objet basé sur une classe.

4.3.4 Constructeur (CONSTRUCTOR)

C'est le module de la classe qui dit au système comment effectuer l'instanciation de celle-ci. Il réserve de la place mémoire et initialise certaines propriétés. Delphi appelle cette méthode "Create".

4.3.5 Destructeur (DESTRUCTOR)

C'est l'inverse du constructeur. Ce module est chargé de libérer la mémoire occupée par l'objet. ("Destroy" en Delphi).

4.3.6 Classe ancêtre

C'est une classe à partir de laquelle sont héritées les propriétés et les méthodes. Ce n'est pas obligatoirement la classe parente immédiate, mais elle peut être plus élevée hiérarchiquement.

4.3.7 Classe parent

C'est la classe immédiatement supérieure au niveau hiérarchique

4.3.8 Classe enfant

C'est la classe immédiatement inférieure au niveau hiérarchique

4.3.9 Classe descendante

Son niveau hiérarchique est inférieur à la classe considérée. Ce n'est pas obligatoirement la classe enfant qui est située directement au niveau inférieur suivant.

4.3.10 Propriétaire(OWNER)

Une application est propriétaire des fenêtres qu'elle manipule. Une fenêtre est propriétaire des objets quelle contient. Un panneau, comme la barre d'outils, est propriétaire de ses turbo-boutons...

4.3.11 Héritage

Une classe hérite automatiquement des propriétés et méthodes de ses ancêtres.

4.4 CONCLUSION

Le but de la POO est de créer des composants réutilisables pour faciliter la création de projets par assemblage d'éléments préfabriqués.

Les avantages sont énormes:

- réutilisation de l'existant sans limite (pas d'écritures répétitives),
- développement RAD visuel possible (une application peut se créer comme un assemblage lors d'un puzzle),
- cloisonnement de sécurité entre les données (évite les effets de bord),
- facilité d'évolution (il suffit de placer des objets plus évolués),
- et de maintenance (possibilité de tester individuellement chaque objet).

L'héritage et le polymorphisme accroissent de façon sensible la productivité du programmeur.

5 Les objets de Windows (rappel)

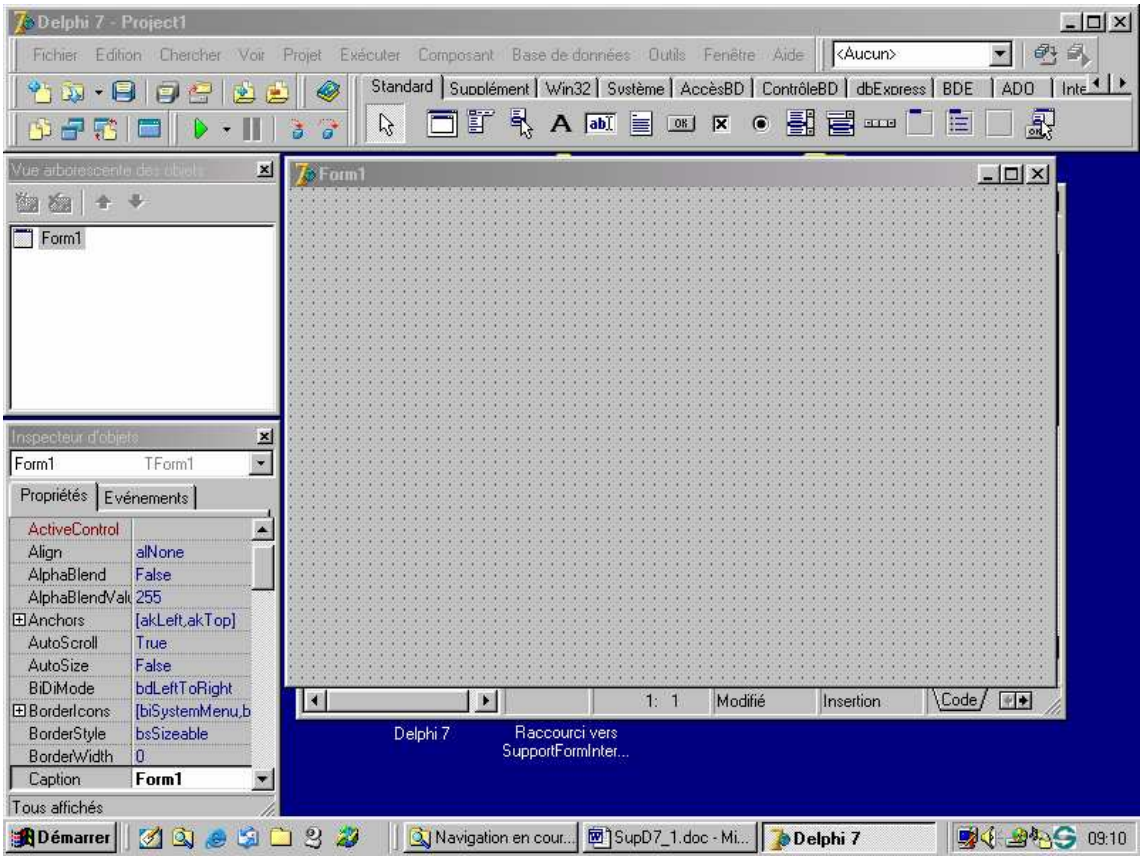
Sur le bureau de Windows, on peut trouver 4 types d'objets:

- des programmes
- des documents
- des dossiers
- des raccourcis vers les 3 types d'objets précités.

Les objets visuels peuvent être manipulés grâce à la souris:

action	manipulation souris	résultat
sélection	clic sur l'objet	l'objet change de couleur ou des poignées apparaissent
désélection	clic en dehors de l'objet	l'objet reprend sa couleur initiale ou ses poignées disparaissent
ouverture	double-clic sur l'objet	soit on se trouve dans l'application correspondante à l'objet, soit on voit ses propriétés
fermeture	clic en dehors ou sur bouton X	l'objet est fermé
déplacement	cliquer-glisser sur (parfois le bord) de l'objet	l'objet est déplacé
duplication	Ctrl + cliquer-glisser sur (parfois le bord) de l'objet	il y a deux objets **identiques**
interrogation	clic-droit sur l'objet	apparition d'un menu surgissant appelé contextuel
destruction	sélection puis touche "Suppr"	l'objet a disparu
insertion	menu contextuel sur **l'objet suivant**	il y a 1 objet en plus

6 L'écran de Delphi



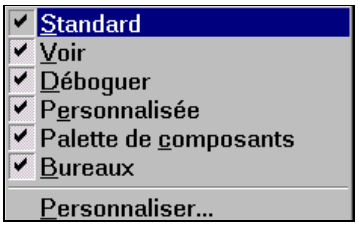
On voit la barre de titre, la barre des menus, la barre d'outils (à gauche), la palette des composants, la fenêtre principale, l'éditeur de code (en dessous) et l'éditeur de propriétés (et la vue arborescente des objets). Une fiche est créée d'office, c'est la fenêtre principale de l'application (modifiable).

Touches de fonction importantes:

- F12: bascule entre l'affichage de la fiche et son code
- F11: fait apparaître l'inspecteur d'objets.

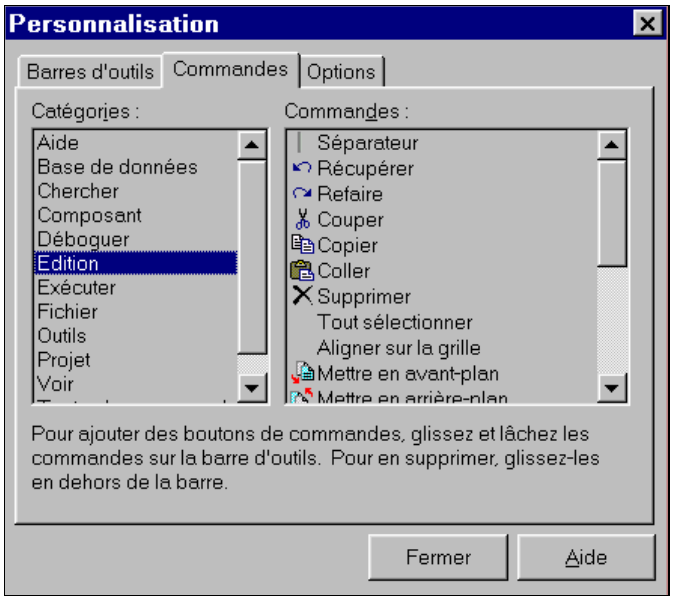
6.1 La barre d'outils (paramétrable):

Des infos-bulle donnent la signification des boutons. Le paramétrage peut se faire en appelant le menu contextuel (clic-droit).



Il est intéressant de placer les boutons suivants:

- Nouvelle application, tout enregistrer
- Défaire et refaire
- Couper, copier, coller
- Vérifier la syntaxe
- Exécuter, pas à pas
- Voir une fiche
- Voir la palette d'alignement
- Rubriques d'aide
- Aide sur les API Windows



6.2 La palette des composants

C'est une barre d'outils de type particulier: on y trouve les objets qui existent déjà dans Delphi. Les onglets correspondent aux catégories de composants: cliquer sur l'onglet pour changer de palette. Pour placer un composant dans la fiche, on peut cliquer sur un composant puis cliquer dans la fiche (cliquer-glisser pour redimensionner de suite). Il est également possible de double-cliquer sur un composant: il se mettra automatiquement au centre de la fiche en cours. Pour placer plusieurs composants identiques, sélectionner le type avec maj-clic.

Il existe divers types de composants:

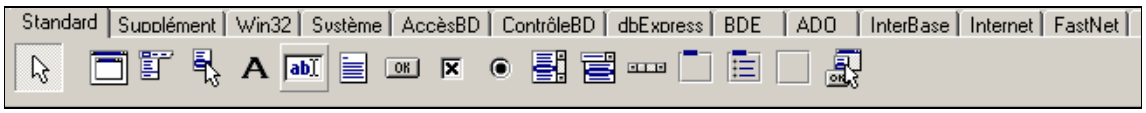
- visuels: ils sont visibles à l'exécution (sauf indication contraire); certains d'entre eux peuvent prendre le "focus", c'est à dire être sélectionnés par l'utilisateur
- non-visuels: ne sont visibles qu'en phase de création; ils disparaissent à l'exécution.

Les composants peuvent se reproduire avec copier-coller.

Voici les composants les plus utilisés dans une application standard.

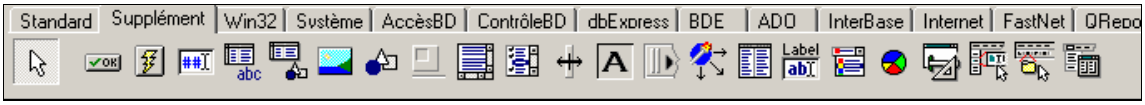
6.2.1 Les composants de la page Standard

Les composants de la page Standard de la palette des composants permettent d'utiliser dans une application Delphi les contrôles standard de Windows :



Composant	Utilisation
Cadre	Composant cadre (nouveau): sert de conteneur de composants
MainMenu	Menu principal
PopupMenu	Menu surgissant
Label	Texte en lecture seule pour l'utilisateur (ex: titre)
Edit	Texte en lecture-écriture modifiable par l'utilisateur (zone de saisie)
Memo	Mémo: comme le composant précédent, mais possibilité d'utiliser plusieurs lignes de texte
Button	Bouton d'action: c'est le composant le plus utilisé
CheckBox	Case à cocher. Propose une option que l'utilisateur peut faire passer de Oui à Non ou de Vrai à Faux. Les cases à cocher peuvent servir à afficher un groupe de choix qui ne sont pas mutuellement exclusifs. Les utilisateurs peuvent sélectionner plusieurs cases à cocher dans un groupe.
RadioButton	Bouton radio. Propose une option que l'utilisateur peut faire passer de Oui à Non ou de Vrai à Faux. Les boutons radio peuvent servir à afficher un groupe de choix qui sont mutuellement exclusifs. Les utilisateurs ne peuvent sélectionner qu'un seul bouton radio par groupe.
ListBox	Boîte liste. Affiche une liste déroulante de choix.
ComboBox	Boîte à options. Affiche une liste de choix en combinant une boîte liste et une boîte de saisie. Les utilisateurs peuvent saisir des données dans la boîte de saisie ou sélectionner un élément dans la zone boîte liste.
ScrollBar	Barre de défilement. Propose un moyen de modifier la zone visualisée d'une liste ou d'une fiche. Une barre de défilement peut également être utilisée pour se déplacer par incréments dans un intervalle de valeurs.
GroupBox	Boîte groupe. Sert de conteneur à des options associées dans une fiche.
RadioGroup	Groupe de boutons radio. Crée une boîte groupe qui contient des boutons radio sur une fiche.
Panel	Volet. Crée dans une fiche des volets pouvant contenir d'autres composants. Les volets peuvent servir à créer des barres d'outils ou des lignes d'état.
ActionList	Composant permettant la centralisation du code événementiel

6.2.2 Les composants de la page Supplément



Les composants de la page "Supplément" de la palette des composants permettent d'utiliser dans une application Delphi des contrôles Windows spécialisés :

Composant	Utilisation
BitBtn	Bouton bitmap. Crée un composant bouton pouvant afficher une image.
SpeedButton	TurboBouton. Propose un bouton pouvant afficher une image mais pas un libellé. Les TurboBoutons sont fréquemment regroupés dans un volet afin de créer une barre d'outils.
MaskEdit	Comme le composant "Edit" mais permet une saisie formatée
StringGrid	Grille de chaînes. Crée une grille pouvant être utilisée pour afficher des données chaînes de caractères en lignes et en colonnes comme dans un tableur.
DrawGrid	Grille d'affichage. Crée une grille permettant d'afficher des données en lignes et en colonnes. Plus difficile à utiliser que le composant précédent.
Image	Image, gère maintenant également les jpeg
Shape	Forme. Dessine des formes géométriques : cercle ou ellipse, carré ou rectangle (arrondi ou non).
Bevel	Biseau. Crée des lignes ou des boîtes avec un aspect tridimensionnel ciselé.
ScrollBox	Boîte de défilement. Crée un conteneur redimensionnable affichant automatiquement des barres de défilement lorsque c'est nécessaire.
CheckListBox	Boîte liste de cases à cocher. Affiche une liste déroulante semblable à une boîte liste, mais avec une case à cocher à côté de chaque élément.
Splitter	Diviseur. Placé sur une fiche entre deux contrôles alignés, il permet aux utilisateurs de redimensionner les contrôles, à l'exécution, en faisant glisser la ligne de division.
StaticText	Texte statique. Un composant texte non modifiable, comme le composant Label, mais contenant un handle de fenêtre, ce qui est utile quand la touche raccourci du composant doit appartenir à un contrôle fenêtré. Utilisez le composant StaticText pour donner à l'utilisateur des informations sur l'état en cours de l'application.
ControlBar	Composant utilisé pour contenir des barres d'outils
Application Events	Contrôle des événements au niveau application (nouveau)
Value List Editor	Gère des couples: clé, valeur
Labeled Edit	Zone de saisie avec étiquette
ColorBox	Liste de couleurs pour choix
Chart	Graphique
ActionManager	Composant centralisateur de procédures événementielles
ActionMainMenu Bar	Barre de menu d'ActionManager

ActionToolBar	Barre d'outils d'ActionManager
CustomizeDlg	Boîte de dialogue permettant à l'utilisateur de configurer les 2 composants précédents

6.2.3 Les composants de la page Win32

Les composants de la page Win32 de la palette des composants permettent d'utiliser dans une application Delphi des contrôles d'interface utilisateur standard de Windows 32 bits (Windows 95, 98 ou NT).

Composant	Utilisation
TabControl	Onglets: permet d'accéder à des pages d'informations
PageControl	Pages: comme le contrôle précédent, mais il y a de la place pour placer des composants
ImageList	Liste d'images: sert souvent à stocker les images des boutons des barres d'outils modernes
RichEdit	Editeur de texte formaté: créer un logiciel de traitement de textes devient facile avec ce composant
TrackBar	Glissière de réglage: permet à l'utilisateur de définir une valeur analogique
ProgressBar	Barre de progression: sert souvent à indiquer la proportion d'achèvement du travail
HotKey	Raccourci-clavier qui peut être défini par l'utilisateur
Animate	Clip vidéo muet
DateTimePicker	Boîte liste permettant la saisie de dates ou d'heures.
MonthCalendar	Calendrier: l'utilisateur peut sélectionner une date ou une plage de dates
TreeView	Vue arborescence
ListView	Affichage mode liste
HeaderControl	Entête de colonne
StatusBar	Barre d'état
ToolBar	Barre d'outils nouvelle génération
CoolBar	Barres d'outils multiples style "internet explorer'
PageScroller	Zone d'affichage pouvant servir aux barres d'outils
ComboBoxEx	Liste déroulante avec images

6.2.4 Les composants de la page Système

Grâce aux composants de la page Système de la palette des composants, il est possible d'utiliser dans une application Delphi des contrôles spécialisés du système.

Composant	Utilisation
Timer	Minuterie: permet de déclencher des événements à intervalles réguliers
PaintBox	Boîte à peindre. Spécifie une zone rectangulaire de la fiche constituant la délimitation des dessins de l'application.
MediaPlayer	Multimédia. Affiche un contrôle de style télécommande pour lire ou enregistrer des fichiers multimédias son ou vidéo.
OleContainer	Conteneur OLE. Crée dans une fiche une zone client OLE (Incorporation et Liaison d'Objet).
DdeClientConv	Conversation client DDE. Établit une connexion client avec une application serveur DDE (Échange Dynamique de Données).
DdeClientItem	Élément client DDE. Spécifie les données du client DDE (Échange Dynamique de Données) à transférer lors d'une conversation DDE.
DdeServerConv	Conversation serveur DDE. Établit une connexion serveur avec une application client DDE (Échange Dynamique de Données).
DdeServerItem	Élément serveur DDE. Spécifie les données du serveur DDE (Échange Dynamique de Données) à transférer lors d'une conversation DDE.

6.2.5 Les composants de la page Dialogues

Les composants de la page Dialogues de la palette des composants permettent d'utiliser dans une application Delphi les boîtes de dialogues communes de Windows. Grâce à ces boîtes de dialogue, il est possible de proposer une interface homogène pour des opérations relatives aux fichiers (comme l'enregistrement, l'ouverture ou l'impression).

Une boîte de dialogue commune est ouverte lors de l'exécution de sa méthode "Execute". Celle-ci renvoie l'une des valeurs booléennes suivantes :

- "True", si l'utilisateur choisit OK et valide la boîte de dialogue
- "False", si l'utilisateur choisit Annuler ou quitte la boîte de dialogue sans enregistrer aucune modification.

Chaque composant Boîte de dialogue commune (sauf le composant "PrinterSetup") a un ensemble de propriétés regroupées sous l'intitulé Options dans l'inspecteur d'objets. Les propriétés Options interviennent sur l'aspect et le comportement des boîtes de dialogue communes. Pour afficher les propriétés Options, double-cliquez sur "Options" dans l'inspecteur d'objets.

Pour fermer par programmation une boîte de dialogue, utilisez la méthode "CloseDialog". Pour modifier la position d'une boîte de dialogue à l'exécution, utilisez les propriétés Handle, Left, Top et Position.

Composant	Utilisation
OpenDialog	Boîte de dialogue d'ouverture.
SaveDialog	Boîte de dialogue d'enregistrement.
OpenPictureDialog	Boîte de dialogue d'ouverture d'image avec prévisualisation
SavePictureDialog	Boîte de dialogue d'enregistrement d'image avec prévisualisation
FontDialog	Boîte de dialogue des polices
ColorDialog	Boîte de dialogue des couleurs
PrintDialog	Boîte de dialogue d'impression
PrinterSetupDialog	Boîte de dialogue de configuration d'impression.
FindDialog	Boîte de dialogue de recherche
ReplaceDialog	Boîte de dialogue de remplacement

6.2.6 Les composants de la page Win3.1

Les composants de la page Win3.1 de la palette des composants permettent d'utiliser dans vos applications Delphi des contrôles de Windows 3.1 pour assurer la compatibilité avec les applications construites avec des versions précédentes de Delphi. La plupart de ces anciens contrôles offrent le même comportement que les derniers contrôles 32 bits.

Important : Evitez ces contrôles lorsque vous créez de nouvelles applications.

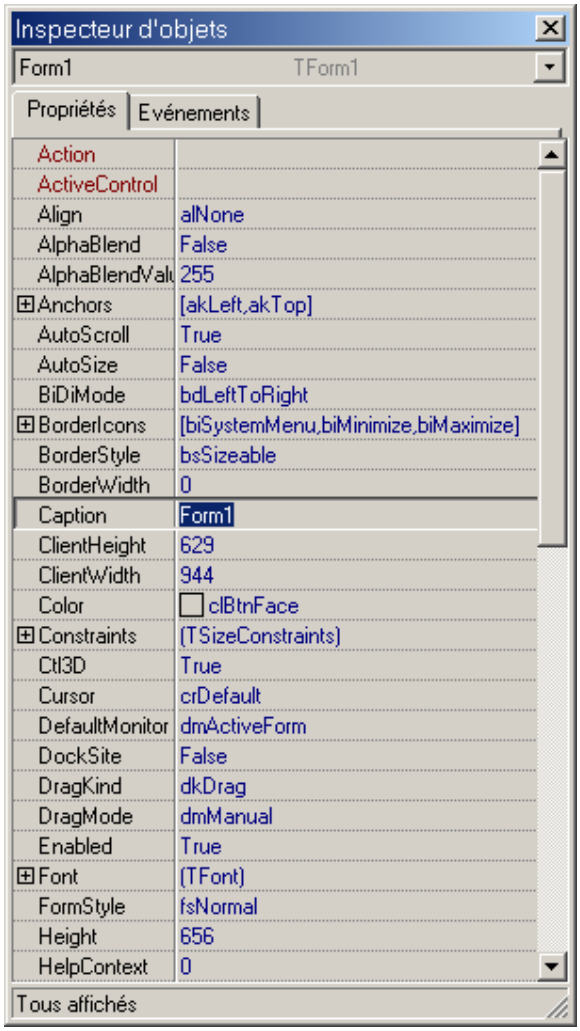
Le tableau suivant indique les contrôles à utiliser à la place des anciens contrôles :

Contrôle Win 3.1	Remplacez-le par	Page du nouveau contrôle
DBLookupList	DBLookupListBox	ContrôleBD
DBLookupCombo	DBLookupComboBox	ContrôleBD
TabSet	TabControl	Win32
Outline	TreeView	Win32
Header	HeaderControl	Win32
NoteBook	PageControl	Win32
TabbedNoteBook	PageControl	Win32

Composant	Utilisation
DBLookupList	Liste de référence de base de données. Boîte de liste orientée données pour afficher à l'exécution les valeurs trouvées dans les colonnes d'une autre table.
DBLookupCombo	Boîte à options de référence de base de données. Boîte à options orientée données pour afficher à l'exécution les valeurs trouvées dans les colonnes d'une autre table.
TabSet	Onglets. Crée des onglets semblables à ceux d'un classeur. Le composant TabSet peut s'utiliser avec le composant Notebook pour permettre aux utilisateurs de changer de page.
Outline	Arborescence. Affiche des informations sous forme d'arborescences de différents formats.
TabbedNotebook	Classeur à onglets. Crée une zone d'affichage des données. Les utilisateurs peuvent redimensionner chaque section de cette zone pour y afficher différentes quantités de données.
Notebook	Classeur. Crée un composant pouvant contenir plusieurs pages. Le composant TabSet peut s'utiliser avec le composant Notebook pour permettre aux utilisateurs de changer de page.

Composant	Utilisation
Header	En-tête. Crée une zone d'affichage des données. Les utilisateurs peuvent redimensionner chaque section de cette zone pour y afficher différentes quantités de données.
FileListBox	Boîte liste de fichiers. Affiche une liste déroulante des fichiers du répertoire en cours.
DirectoryListBox	Boîte liste des répertoires. Affiche la structure des répertoires du lecteur actif. Les utilisateurs peuvent changer de répertoire dans une boîte liste des répertoires.
DriveComboBox	Boîte à options des lecteurs. Affiche une liste déroulante des lecteurs disponibles.
FilterComboBox	Boîte à options de filtrage. Spécifie un filtre ou un masque afin d'afficher un sous-ensemble des fichiers.

6.3 L'inspecteur d'objets



L'inspecteur d'objets de Delphi est la passerelle entre l'aspect visuel de votre application et le code qui lui permet de fonctionner.

L'inspecteur d'objets vous permet de :

- définir en mode conception les propriétés des composants que vous placez sur une fiche (ou de la fiche elle-même),
- créer les gestionnaires d'événements.

Le sélecteur d'objet en haut de l'inspecteur affiche est une liste déroulante contenant tous les composants de la fiche active, ainsi que leur type. Vous pouvez ainsi sélectionner rapidement différents composants de la fiche active.

Vous pouvez modifier la largeur des colonnes de l'Inspecteur d'objets en faisant glisser la ligne de séparation vers une nouvelle position.

L'inspecteur d'objets comporte deux pages :

6.4 Page Propriétés

La page Propriétés de l'Inspecteur d'objets vous permet, en mode conception, de définir les propriétés des composants de votre fiche, mais aussi celles de la fiche. Vous pouvez définir des propriétés d'exécution en écrivant directement du code source dans les gestionnaires d'événements. La page Propriétés n'affiche que les propriétés du composant sélectionné dans la fiche. Définir les propriétés d'un composant en mode conception revient à définir son état initial.

Des propriétés peuvent être imbriquées: dans ce cas, elle sont précédées par une flèche. Pour développer l'arborescence, double-cliquez.

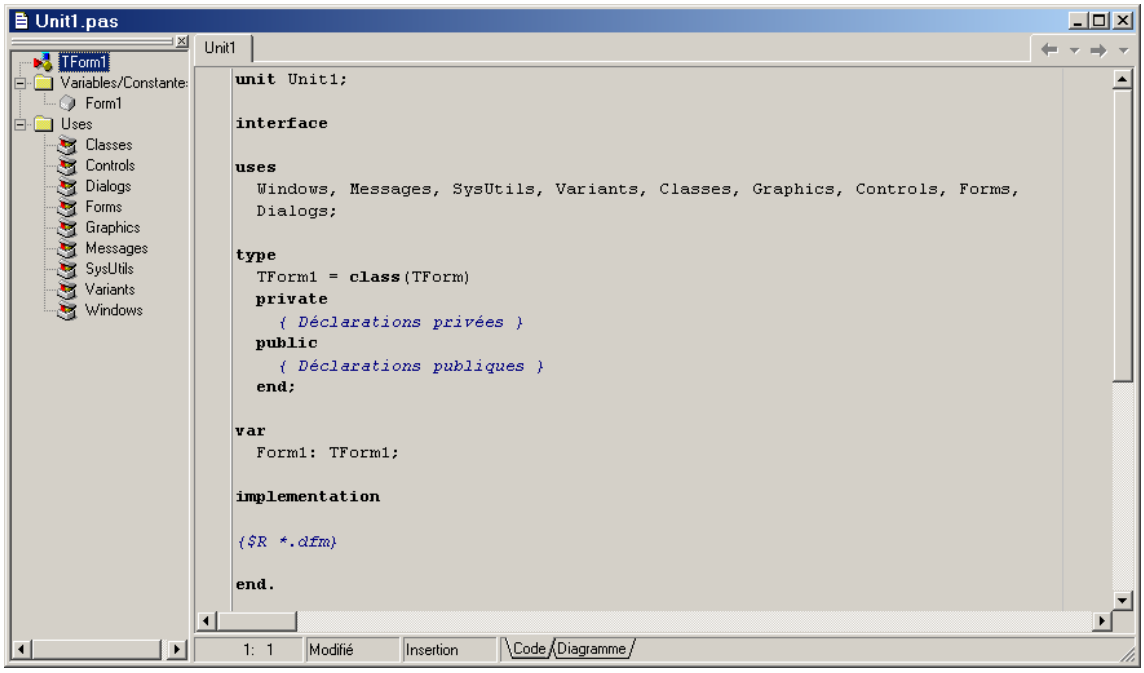
La présence de 3 points de suspension (...) indique la présence d'un éditeur de propriétés particulier. On trouve également des listes déroulantes.

6.5 Page Evénements

La page Evénements de l'Inspecteur d'objets vous permet d'associer des événements aux objets de votre projet. Trois utilisations sont possibles:

UTILISATION	EFFET
double-clic à coté de l'événement	création automatique d'un entête de procédure par défaut
introduction d'un nom de procédure	création automatique d'un entête de procédure avec le nom choisi
ouverture d'une liste déroulante (s'il y a déjà des procédures compatibles)	choix d'une procédure commune à plusieurs événements

6.6 L'éditeur de code



Dans certains cas, on voit dans la partie de gauche l'explorateur de code. Il facilite la navigation dans vos fichiers. La fenêtre de l'explorateur de code contient une arborescence qui montre tous les types, classes, propriétés, méthodes, variables globales et routines globales définis dans votre unité. Il montre également les autres unités listées dans la clause uses

L'éditeur de code est un éditeur complet. Il vous permet d'accéder au code (source) des modules de votre projet. De nombreuses commandes sont disponibles dans le menu contextuel de l'éditeur de code. Pour personnaliser l'éditeur de code, utilisez la boîte de dialogue "Outils" - "Options d'environnement". Pour obtenir de l'aide sur un élément dans l'éditeur de code, placez le pointeur de souris sur le terme qui vous intéresse et appuyez sur F1.

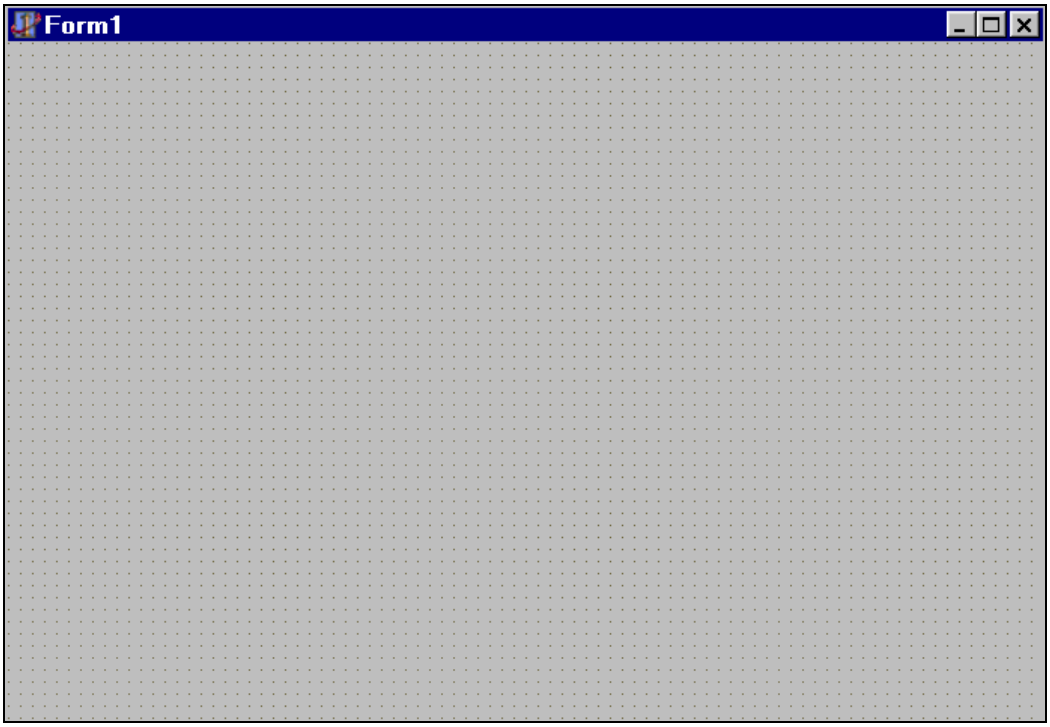
Lors de la compilation, si vous recevez un message d'erreur, Delphi :

- affiche l'erreur dans la boîte de message de l'éditeur de code
- met en surbrillance la ligne de code qui pose problème.

7 Premier projet

7.1 Notions élémentaires

Un projet est un l'ensemble des fichiers nécessaires au fonctionnement de l'application. Quand on charge Delphi, un projet de base est créé. Il comporte une fenêtre prête à recevoir différents objets.



On remarque la barre de titre, la case système comportant une icône et des boutons pour agrandir, réduire ou fermer la fenêtre.

L'inspecteur d'objets permet d'agir sur les propriétés et méthodes événementielles de l'objet.

Il est évident que cette première application n'est pas encore exécutable. Pour ce faire, il est nécessaire de compiler (traduire en binaire). Nous n'avons pas écrit une ligne de programme, le système s'en est chargé. Examinons les lignes de code générées (.PAS).

Lignes de code	Explications
unit Unit1;	nom du module source correspondant à la fiche "Form1" ne pas changer ce nom directement, mais au moment de l'enregistrement
interface	entête pour désigner la section qui sera "vue" par les autres modules
uses	permet d'inclure d'autres modules
Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs;	nom des modules inclus
type	début de la déclaration du type des objets
TForm1 = class(TForm)	déclaration d'un nouveau type d'objet de type fenêtre
private	
{ Déclarations privées }	propriétés et méthodes "privées" du type d'objet
public	
{ Déclarations publiques }	propriétés et méthodes "publiques" du type d'objet; elles seront visibles par les autres modules
end;	fin de déclaration d'objet
var	"achat" de variables globales
Form1: TForm1;	instanciation de l'objet: allocation de mémoire vive
implementation	dans cette section seront écrites les méthodes des objets manipulés
{$R *.DFM}	directive de compilation permettant d'inclure les définitions des fenêtres (.DFM)
end.	Fin du module

Ce module ne peut fonctionner seul. Un module de projet (.DPR) se charge de fédérer l'ensemble des modules, mais il n'est pas visible en ce moment. Pour le visualiser, ouvrez le menu "Projet" et choisissez "voir le source". Un onglet correspondant apparaît dans l'éditeur de code.

program Project1;	nom du programme
uses	modules utilisés
Forms,	
Unit1 in 'UNIT1.PAS' {Form1};	c'est le module de la fiche "Form1"
{$R *.RES}	directive de compilation permettant d'inclure les fichiers ressources (.RES)
begin	début du programme principal
Application.Initialize;	initialisation du programme
Application.CreateForm(TForm1, Form1);	création de la fiche "Form1" de type Tform1
Application.Run;	exécution de l'application
end.	Fin du programme principal

On peut également visualiser le code de la fiche (.DFM), grâce à l'option "voir comme texte" de son menu contextuel.

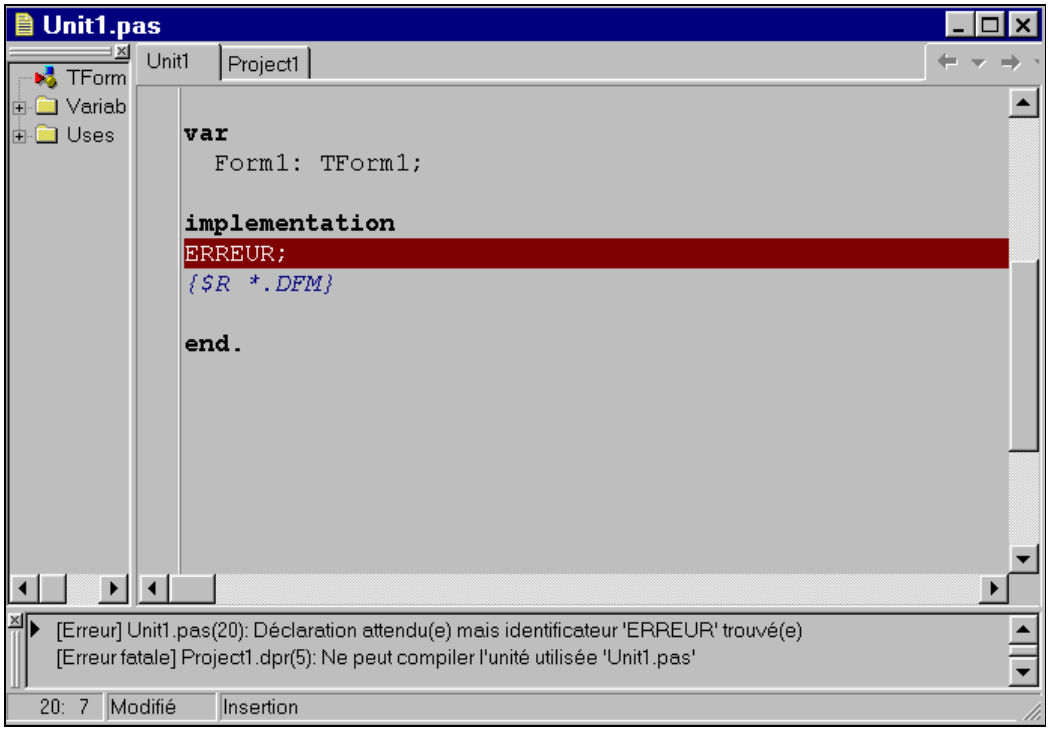
```
object Form1: TForm1
  Left = 295
  Top = 132
  Width = 696
  Height = 480
  Caption = 'Form1'
  Color = clBtnFace
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -13
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  OldCreateOrder = False
  PixelsPerInch = 120
  TextHeight = 16
end
```

Revenir à la fiche avec "voir comme fiche" du menu contextuel.

Pour enregistrer le projet, choisir "enregistrer projet sous" dans le menu fichier. Il faudra donner un nom aux différents modules (.PAS) et au projet (DPR).

7.2 Exécution

On peut vérifier la syntaxe du code, compiler ou exécuter directement le projet. Pour disposer d'un exécutable, il est nécessaire de compiler. En cas d'erreur, la compilation s'arrête et visualise un message d'erreur.



Si l'on désire des renseignements sur l'erreur, il suffit cliquer sur le message d'erreur ou d'avertissement et d'appuyer sur la touche F1 (aide en ligne) et on obtient la fenêtre suivante:

29. <Token1> attendu mais <token2> trouvé

Liste complète des messages d'erreur du compilateur

Il s'agit d'une erreur de syntaxe. Il y a probablement une erreur de typo dans les sources ou quelque chose a été oublié. Lorsque l'erreur se produit au début d'une ligne, l'erreur actuelle est souvent sur la ligne précédente.

```
program Produce;
var
  I: Integer
begin               (*<-- Message d'erreur ici: ';' attendu mais 'BEGIN' trouvé*)
end.
```

Après le type Integer, le compilateur attend de trouver un point-virgule pour terminer la déclaration de variable. Il ne trouve pas de point-virgule sur la ligne courante, ainsi il lit et trouve le mot clé 'begin' au début de la ligne suivante. Il sait alors que quelque chose est faux...

```
 program Solve;
var
  I: Integer;       (*point-virgule manquant*)
begin
end.
```

Dans ce cas, seul le point-virgule était manquant - cas fréquent en pratique. En général, regardez la ligne où apparaît le message d'erreur, et la ligne au-dessus pour trouver s'il manque quelque chose ou si quelque chose est mal orthographié.

Il suffit de rectifier le code et de relancer une compilation. Ensuite, on peut directement utiliser l'exécutable généré (.EXE). Quitter Delphi et lancer le programme depuis le l'explorateur Windows.

Examen des fichiers générés:

Suffixe	Signification
dof	options de compilation et informations de versions
dpr	source du projet
exe	programme exécutable compilé
dcu	unité compilée
dfm	définition de la feuille
pas	source pascal
res	ressources: icône de l'application
cfg	configuration du compilateur
ddp	diagrammes

Pour examiner certains de ces fichiers, il est pratique d'utiliser l'explorateur Windows. Mettre un raccourci vers le bloc-notes (notepad.exe) dans le répertoire SendTo.

8 Modification du projet

Relancer Delphi et ouvrir le projet précédent. Nous désirons à présent installer un bouton qui change la couleur du fond de la fenêtre en rouge.

8.1 Mise en place du composant

Choisir un bouton ("Tbutton") par un clic dans la palette standard et cliquer dans la fiche. On peut également double-cliquer dans la palette et le bouton se mettra automatiquement au milieu de la fiche. Ce bouton se nomme "Button1" et ne fait rien pour l'instant. Si l'on examine le module source, on trouve sa déclaration dans les types.

Deux propriétés importantes: "caption" qui permet de changer l'affichage du texte sur le bouton et "name" qui est le nom qui servira à adresser l'objet. Changer le nom du bouton (c'est indispensable pour les projets importants) avec quelque chose de plus significatif: BtRouge. On constate que la propriété "caption" est également modifiée. Changer cette propriété pour que le bouton affiche "Colorer en rouge". Le composant peut être redimensionné grâce à ses poignées (visibles quand sélectionné).

8.2 Affectation d'une procédure événementielle

La liste des événements peut s'observer grâce à l'inspecteur d'objets. Pour créer une procédure simple "OnClic", il suffit de double-cliquer sur le bouton.

Le code suivant est généré:

```
procedure Tform1.BtRougeClick(Sender: Tobject);
begin

end;
```

Cette procédure événementielle est prête à accueillir une ou plusieurs lignes d'instructions entre "begin" et "end". Si l'on désire rajouter des commentaires (et c'est indispensable pour une bonne maintenance), utiliser les méthodes suivantes:

- { commentaires: plusieurs lignes possibles }
- (* commentaires: plusieurs lignes possibles *)
- // commentaires: sur une seule ligne

Si je ne connais pas la façon de colorer la fenêtre, je vais chercher les propriétés de l'objet Tform grâce à l'aide en ligne. Il suffit de sélectionner "Form1" et d'appuyer sur le touche F1. Ensuite, cliquer sur "propriétés". La liste affiche "color": cliquer sur cette propriété. On trouve que "clRed" correspond au rouge.

Je vais donc compléter la procédure pour obtenir le listing suivant:

```
procedure Tform1.BtRougeClick(Sender: Tobject);
{colore en rouge le fond de la fenêtre}
begin
Form1.color:=ClRed;
end;
```

Remarques:

- quand je tape "Form1.", l'éditeur de code me propose la liste des propriétés, méthodes et variables de l'objet.
- le séparateur entre objet et méthodes est le point et l'instruction d'affectation est ":=" et non "=" qui est réservé aux tests d'égalité ou à la déclaration des types. Les lignes d'instruction se terminent par le point-virgule.

Enregistrer le projet, et vérifier son exécution.

8.3 Améliorations

Il est dommage de ne pas avoir de bouton pour remettre la couleur d'origine et un autre pour quitter le programme. On pourrait double-cliquer sur la fiche pour rétablir la couleur d'origine.

Sélectionner la fenêtre et double-cliquer à côté de l'événement "OnDblClick" dans l'inspecteur d'objet. Le code suivant est généré:

```
procedure Tform1.FormDblClick(Sender: Tobject);
begin

end;
```

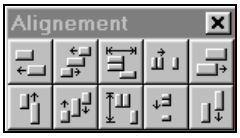
L'aide nous dit que "clWindow" pour la couleur de fond courante des fenêtres. La procédure devient donc:

```
procedure Tform1.FormDblClick(Sender: Tobject);
begin
Form1.color:=clWindow;
end;
```

Exécuter et tester.

La palette "suppléments" nous fournit un composant intéressant: TBitBtn. C'est un bouton comportant une image (glyphe) et ayant des fonctions préprogrammées (Kind). Mettre cette propriété "Kind" à "bkClose".

Aligner les deux boutons à gauche grâce à la palette d'alignement (menu "Voir").



On peut sélectionner plusieurs objets en cliquant-glissant pour tracer un rectangle qui les touche ou les englobe.

Un menu contextuel sur les objets permet également de les rendre de même taille ou de les changer d'échelle:

Présenter les deux boutons de même taille et alignés à droite et à gauche.

Enregistrer et exécuter.

Exercice:

Enlever le bouton "rouge" et placer trois curseurs dans des panneaux de couleur pour régler l'intensité des 3 couleurs de base: rouge, vert et bleu. Les intensités varient de 0 à 255. La fonction RGB (voir l'aide en ligne) est utile. Un seul gestionnaire d'événement suffira.

9 Eléments de programmation

9.1 Déclaration de variables

Une variable est un nom qui, dans du code, représente une adresse mémoire dont le contenu peut changer lorsque le code est exécuté. Vous devez déclarer une variable avant de l'utiliser. Les étapes requises sont les suivantes :

- Choix du nom d'une variable
- Choix du type d'une variable

9.1.1 Choix du nom d'une variable

Lorsque vous déclarez une variable, vous devez choisir un nom. Pour faciliter la lecture de votre code, il est préférable de choisir un nom parlant, qui vous rappelle la signification de la variable. Par exemple, Jour, Nom ou Total sont des noms plus évocateurs que X, Y ou Z.

En dehors du choix de noms évocateurs, il convient de garder à l'esprit un certain nombre de règles qui s'appliquent aux variables et à tous les autres identificateurs:

- Les identificateurs peuvent avoir une longueur de 63 caractères.
- S'ils dépassent cette longueur, le compilateur ne tient pas compte des caractères supplémentaires.
- Les identificateurs doivent commencer par une lettre ou par un trait de soulignement (_).
- Les caractères suivants peuvent être des lettres, des traits de soulignement ou les chiffres de 0 à 9.
- Ils ne peuvent contenir de caractères génériques comme $, %, *, etc.
- Par exemple, si vous utilisez des identificateurs appelés Faire@Midi ou Retour_en_%, vous aurez une erreur de syntaxe.
- Vous ne devez pas utiliser un mot réservé de Pascal objet pour désigner un identificateur: and, as, asm, array, begin, case, class, const, constructor, destructor, div, do, downto, else, end, except, exports, file, finally, for, function, goto, if, implementation, in, , inherited, inline, initialization, interface, is, label, library, mod, nil, not, object, of, or, packed, procedure, program, property, raise, record, repeat, set, shl, shr, string, then, to, try, type, unit, until, uses, var, while, with, xor
- Evitez d'utiliser des identificateurs déjà définis dans le langage Pascal objet pour désigner vos identificateurs. Par exemple, les mots Boolean et Integer sont des types de données prédéfinis.

Conventions d'écriture: nous allons adopter une séparation des significations par majuscules comme par exemple: TMonObjetPersonnelSecret. Interdiction formelle de mettre des espaces ou des accents.

9.1.2 Choix du type d'une variable

Lorsque vous déclarez une variable, vous devez déclarer son type. Le type d'une variable définit l'ensemble des valeurs qu'elle peut contenir.

En règle générale, vous déclarerez une variable là où vous en aurez besoin. Par exemple, si vous voulez utiliser une variable dans un gestionnaire d'événements, vous la déclarerez à l'intérieur de celui-ci. Il s'agit dans ce cas d'une variable locale. Si la variable doit être visible par plusieurs procédures ou fonctions, elle doit être déclarée avant la section implémentation: elle sera globale.

Les déclarations de variables doivent toujours être précédées du nom réservé var. La déclaration elle-même contient deux parties : celle de gauche est le nom de la nouvelle variable, celle de droite, son type. Ces deux parties sont séparées par le signe deux points (:). Cet exemple contient trois déclarations de variables : Valeur, Somme et Ligne :

```
var
 Valeur: Integer;
 Somme: Integer;
 Ligne: string;
```

Comme Valeur et Somme sont toutes deux du type Integer (entier), la déclaration de variables peut être présentée comme ci-dessous, avec une virgule pour séparer les deux variables du même type :

```
var
 Valeur, Somme: Integer;
 Ligne: string;
```

Dans l'exemple suivant, nous supposons qu'il existe une boîte de saisie appelée Edit1 et un bouton appelé BoutonAjouter dans la fiche. Trois variables (X, Y et Somme) sont déclarées comme étant de type Integer. Lorsque l'utilisateur clique sur BoutonAjouter, des valeurs Integer sont affectées à deux variables, X et Y, et ajoutées l'une à l'autre. Le résultat, Somme, apparaît dans la boîte de saisie :

```
procedure Tform1.BoutonAjouterClick(Sender: Tobject);
var
 X, Y, Somme: Integer;
begin
 X := 100;
 Y := 10;
 Somme := X + Y;
  Edit1.Text := IntToStr(Somme);
end;
```

La ligne ci-dessous affecte la valeur d'une expression à une variable :

```
Somme := X + Y;
```

Une expression consiste généralement en deux identificateurs liés l'un à l'autre par un opérateur. Pour plus d'informations sur les expressions, reportez-vous à l'entrée "expression" dans l'index de l'aide de la VCL. Comme la propriété Text d'une boîte de saisie est du type string et la variable Somme du type Integer, une ligne comme celle-ci ne peut pas être autorisée, car elle provoque une erreur d'incompatibilité de types à l'exécution ou à la compilation :

```
Edit1.Text := Somme;
```

Au lieu de cela, la dernière ligne du gestionnaire de l'événement utilise la fonction IntToStr de la bibliothèque d'exécution de Delphi pour convertir la valeur Somme en chaîne de caractères, puis elle affecte la valeur de la variable Somme à la propriété Edit1.Text.

Cette variante évite le risque d'affecter une valeur à Somme :

```
procedure Tform1.BoutonAjouterClick(Sender: Tobject);
var
 X, Y: Integer;
begin
 X := 100;
 Y := 10;
 Edit1.Text := IntToStr(X + Y);
end;
```

9.1.3 TYPES DE DONNEES

Plusieurs types de données sont intégrés au langage Pascal Objet. Vous pouvez créer des variables appartenant à n'importe lequel de ces types prédéfinis. Le tableau ci-dessous donne la liste (non exhaustive) des types de données prédéfinis :

ENTIERS

type	étendue	forme
Integer	–2147483648..2147483647	32 bits signé
Shortint	–128..127	8 bits signé
Smallint	–32768..32767	16 bits signé
Longint	–2147483648..2147483647	32 bits signé
Int64 (nouveau)	–2^63..2^63–1	64 bits signé
Byte	0..255	8 bits non signé
Word	0..65535	16 bits non signé
Longword ou cardinal	0..4294967295	32 bits non signé

Il est facile d'utiliser des entiers en base hexadécimale: faire précéder le nombre par le signe $.

DECIMAUX

type	étendue	chiffres significatifs	octets utilisés
Single	1.5 x 10^–45 .. 3.4 x 10^38	7–8	4
Double (ancien real)	5.0 x 10^–324 .. 1.7 x 10^308	15–16	8
Extended	3.6 x 10^–4951 .. 1.1 x 10^4932	19–20	10
Comp	–2^63+1 .. 2^63 –1	19–20	8
Currency	–922337203685477.5808.. 922337203685477.5807	19–20	8

BOOLEENS

Boolean	Peut contenir les valeurs True ou False. Nécessite un octet de mémoire.

CARACTERES

AnsiChar	Un caractère ANSI.
WideChar	Un caractère Unicode.
Char	Varie en fonction de l'implémentation. Dans cette version du Pascal Objet, Char est semblable à AnsiChar.

On peut utiliser le code ANSI du caractère en le faisant précéder par le signe # (ex: #10).

CHAINES

Type	Longueur maximum	Mémoire nécessaire	Utilisation
ShortString	255 caractères	de 2 à 256 octets	Compatibilité ascendante
AnsiString	~2^31 caractères	de 4 octets à 2Go	Caractère sur 8 bits (ANSI)
WideString	~2^30 caractères	de 4 octets à 2Go	Caractères Unicode; serveurs COM et interfaces
String	Varie en fonction de la directive de compilation $H. Par défaut, string est semblable à AnsiString		

POINTEURS

Un pointeur est une variable qui contient l'adresse mémoire de l'objet manipulé.

La déclaration se fait sous la forme: *var NomPointeur: ^TypeObjet;* ou par le mot-clé "*pointer"* qui représente un pointeur non typé (à transtyper lors avant le déférencement).

Utilisation:

- *new(NomPointeur)* réserve la mémoire correspondant à l'objet
- *NomPointeur^* permet de récupérer l'objet
- *dispose (NomPointeur)* détruit l'objet (pointeur à nil)

L'adresse d'un objet peut être récupérée par la fonction Addr ou @.

VARIANTS

Variant	Type de données pouvant contenir des valeurs de différents types, et pouvant convertir sa valeur en d'autres types de données.

Si vous ne voulez pas utiliser les types de données prédéfinis, vous pouvez définir vos propres types de données. Les types prédéfinis sont généralement suffisants pour la plupart des applications Delphi.

9.1.4 Déclaration de constantes

Alors que les variables contiennent des valeurs qui peuvent changer pendant l'exécution de votre application, les constantes contiennent une valeur immuable que vous leur attribuez au moment où vous les déclarez.

De la même manière que vous déclarez les variables dans une déclaration de variable, les constantes sont à déclarer dans une déclaration de constante. Une déclaration de constante doit commencer par le mot réservé const. Dans l'exemple ci-dessous, nous avons trois déclarations de constantes :

```
const
    Pi = 3.14159;
    Réponse = 342;
    Produit = 'Delphi';
```

Pi, Réponse, et Produit sont des noms de constantes qui décrivent la valeur de celles-ci. Les valeurs réelles des constantes, qui ne peuvent être modifiées au cours de l'exécution de l'application, sont 3.14159, 142 et "Delphi".

Il existe différents types de constantes, comme c'est le cas pour les variables. Par contre, à la différence de ces dernières, les constantes gardent le type de valeur déclaré dans la déclaration de constante. Par exemple, la constante Pi est de type Real parce que la valeur 3.14159 est un nombre réel, Réponse est de type Integer parce que 342 est un entier et Produit est de type string, parce que "Delphi" est une chaîne.

Il convient de noter que le signe égal (=), dans une déclaration de constante, diffère de l'opérateur d'affectation (:=) des instructions d'affectation. Dans une déclaration de constante, le signe égal indique que la partie gauche et la partie droite de la déclaration sont des valeurs égales.

La mauvaise utilisation de l'opérateur "égal" dans le code Pascal objet est une erreur assez courante. Lorsque le compilateur rencontre un signe égal au lieu d'un opérateur d'affectation, il suppose que la partie gauche de l'instruction est une constante ou qu'il y a comparaison de deux valeurs. Si ce n'est pas ce que vous vouliez, Delphi affiche un message d'erreur de syntaxe.

Si, inversement, vous avez fait l'erreur d'utiliser un opérateur d'affectation dans une déclaration de constante, il se produit une erreur de syntaxe et un message apparaît, vous informant que le compilateur attendait le signe égal.

Voici quelques exemples d'utilisation d'expressions constantes dans des déclarations de constantes :

```
const
    Min = 0;
    Max = 100;
    Centre = (Max - Min) div 2;
    Beta = Chr(225);
    NombreCar = Ord('Z') - Ord('A') + 1;
    Message = 'Pas assez de mémoire';
    ChaineErreur = ' Erreur : ' + Message + '. ';
    PosErreur = 80 - Length(ChaineErreur) div 2;
    Ln10 = 2.302585092994045684;
    Ln10R = 1 / Ln10;
    Numerique = ['0'..'9'];
    Alpha = ['A'..'Z', 'a'..'z'];
    AlphaNum = Alpha + Numerique;
```

9.1.5 Création de nouveaux types de données

Pascal objet est livré avec un certain nombre de types de données prédéfinis, mais vous pouvez en créer d'autres: énuméré, intervalle, tableau, ensemble, enregistrement, objet.

9.1.5.1 Type énuméré

Une déclaration de type énuméré donne la liste de toutes les valeurs que le type peut avoir. En voici quelques exemples :

```
type
TJours = (Lundi, Mardi, Mercredi, Jeudi, Vendredi, Samedi, Dimanche);
TCouleurPrimaire = (Rouge, Jaune, Bleu);
TService = (Finances, Personnel, Ingeniérie, Marketing, Informatique);
TChien = (Caniche, PittBull, Chihuahua, BergerAllemnad, Teckel);
```

Voici quelques variables de type énuméré :

```
var
JourSemaine : TJours;
Teinte : TCouleurPrimaire;
Service : TService;
Race : TChien;
```

Chaque valeur entre parenthèses dans une déclaration de type énuméré a une valeur de type entier déterminée par sa position dans la liste. Par exemple, Lundi, dans la déclaration de type TJours a la valeur 0, Mardi a la valeur 1, et ainsi de suite. Vous pourriez obtenir le même résultat en déclarant la variable JourSemaine comme étant de type Integer et en affectant ensuite une valeur Integer pour représenter chaque jour de la semaine. Si ce système donne de bons résultats pour des séries ordonnées et prévisibles, comme les jours de la semaine ou les mois de l'année, mais est beaucoup moins utile si l'ordre des valeurs représentées est arbitraire. Il n'est pas toujours facile de se souvenir de ce que représente un numéro. Par exemple, l'affectation:

```
Race := Chihuahua;
```

est beaucoup plus évocatrice que

```
Race := 2;
```

Quand vous listez une valeur dans un type énuméré, vous la déclarez comme identificateur. Si les déclarations de type énuméré et de variables précédentes étaient dans votre application, vous ne pourriez pas, de ce fait, déclarer une variable Finances, car cet identificateur existe déjà.

9.1.5.2 Type intervalle

Un type intervalle est un intervalle de valeur de type entier, booléen, caractère ou énuméré. Les intervalles sont utiles pour limiter le nombre de valeurs qu'une variable peut avoir.

Pour créer un intervalle, spécifiez les valeurs minimale et maximale en les séparant par deux points (..), par exemple : 1..100

Exemples:

```
type
TFourchetteBoussole = 0..360;
TLettresAdmises = 'A'..'F';
TSalaireMensuel = 10000..30000;
THeures = 0..23;
Ttemps = (Bruine, Giboulées, Pluie, Averse, Orage) {type énuméré}
TPluie = Bruine..Averse {intervalle de Ttemps}
```

9.1.5.3 Type tableau

Un tableau est un ensemble ordonné d'un type de données où chaque élément est spécifié par sa position par ordre numérique dans l'ensemble. Lors de la création du tableau, les éléments ne contiennent aucune valeur, mais vous pouvez les remplir avec des données et manipuler celles-ci à votre guise. Voici un exemple de déclaration de variable pour un tableau de type Double :

```
var
Verif : array[1..10] of Double;
```

Cette déclaration indique que la variable Verif désigne une liste de dix variables de type réel, chacune ayant un numéro (appelé numéro d'indice) compris entre 1 et 10.

Il est fait référence à chaque élément d'un tableau par le nom du tableau, suivi de son indice entre crochets ([ ]). Le tableau Verif contient donc les dix variables Verif[1], Verif[2], Verif[3], Verif[4] et ainsi de suite, jusqu'à Verif[10]. Vous pouvez utiliser n'importe laquelle de ces variables là où vous utiliseriez une variable Real ordinaire. En outre, la valeur d'indice ne doit pas nécessairement être une constante. Ce peut être n'importe quelle expression donnant un entier compris entre 1 et 10. Par exemple,

```
J := 5; Verif[J] := 0,0;
```

Ces instructions affectent la valeur 0,0 à la variable Verif[5]. Comme Verif est un tableau de type Double, la valeur que vous lui affectez doit être de type Real et contenir une virgule décimale. De plus, les valeurs Double devant commencer par un chiffre, vous ne pouvez pas écrire 0,0 sous la forme ,0.

Si vous voulez attribuer la valeur zéro à toutes les variables Verif du tableau, vous pouvez le faire à l'aide d'une boucle for. Comme un indice peut être une variable, il est plus facile d'utiliser une boucle for que d'affecter une valeur à chaque élément du tableau avec des instructions d'affectation distinctes. Cette boucle for affecte la valeur 0,0 aux dix variables :

```
for J := 1 to 10 do Verif(J) := 0,0;
```

Vous pouvez définir des tableaux en tant que types. Par exemple :

```
type
TVerif = array[1..100] of Double;
```

Vous pouvez ensuite déclarer des variables du type tableau. L'exemple ci-dessous déclare que Compte est une variable de type TVerif, soit un tableau de 100 nombres réels.

```
var
Compte : TVerif;
```

9.1.5.4 Tableaux multidimensionnels

Exemple:

```
type TTable = array[1..20, 1..20] of Double;
```

9.1.5.5 Types de chaînes

Par défaut, le compilateur utilise une chaîne longue pour la déclaration "string". La longueur de cette chaîne est limitée uniquement par la mémoire disponible qui est allouée de façon dynamique. Il n'y a donc plus lieu de déclarer la taille de la chaîne.

Exemple:

```
var
MaChaine: string;
```

Si l'on désire utiliser les anciennes chaînes courtes, on peut utiliser "shortstring". Dans ce cas, la longueur est limitée à 255 caractères.

9.1.5.6 Type ensemble

Un ensemble est un groupe d'éléments du même type : entier, booléen, caractère, énuméré ou intervalle. Les ensembles servent à vérifier si une valeur appartient à un ensemble particulier.

Exemple:

```
procedure TForm1.Button1Click(Sender : TObject);
type TVoyelles = set of Char;
var Voyelles : TVoyelles
begin
Voyelles := ['A','E','I','O','U','Y' ];
        if Edit1.Text[1] in Voyelles
        then Label2.Caption := 'Bravo !';
        else Label2.Caption := 'Veuillez réessayer...';
end;
```

9.1.5.7 Type enregistrement

Les enregistrements (ou structures) sont des ensembles de données auxquels votre application peut se référer globalement.

Exemple: le type enregistrement TEmploy déclaré comme suit :

```
type
TEmploy = record
Nom : string[20];
Pren : string[15];
Embauch : 1990..2050;
Salaire : Double;
Poste : string[20];
end;
```

Les enregistrements contiennent des champs contenant des valeurs de données. Chaque champ a un type de données. Les champs du type TEmploy sont Nom, Pren, Embauch, Salaire et Poste. Vous pouvez accéder à ces champs individuellement ou bien vous référer à l'enregistrement dans son ensemble. Il est également possible de déclarer des structures comportant des types variables en fonction de la valeur d'un champ.

Voici un exemple de déclaration de deux variables d'enregistrement :

```
var
NouvelEmploy, EmployPromu : TEmploy;
```

Le code peut se référer au champ Salaire d'un enregistrement NouvelEmploy, comme ci-dessous (le point sépare le nom du champ de celui de l'enregistrement):

```
NouvelEmploy.Salaire := 200000,00;
```

Mais votre code peut aussi manipuler l'enregistrement comme une entité à part entière :

```
EmployPromu := NouvelEmploy;
```

9.2 QUELQUES INSTRUCTIONS PASCAL

9.2.1 Affectation (:=)

Le signe utilisé est := en Pascal. Si l'on écrit A:=B, le contenu de la variable A est détruit et remplacé par le contenu de la variable B. La variable B est donc inchangée. Une erreur fréquente est la confusion avec le signe = seul. Si j'écris A=B, ce n'est pas une affectation, mais une comparaison. Le résultat sera vrai ("true") ou faux ("false").

Note: Le signe ; termine chaque instruction. S'il est omis, l'instruction se poursuit sur la ligne suivante. Les commentaires peuvent se mettre entre accolades, ou après un double-slash dans une même ligne.

9.2.2 Signes opératoires

+	Addition ou concaténation (ex: catena:=#66+'on'+'jour')
-	Soustraction ou inversion (unaire)
*	multiplication
/	division (résultat de type réel)
DIV	division entière
MOD	modulo (reste de la division entière)

9.2.3 Logique booléenne (true-false)

NOT	Inversion logique
AND	Et logique
OR	Ou logique
XOR	Ou exclusif

9.2.4 Opérateurs relationnels

=	Egal 💣 Ne pas confondre avec l'instruction d'affectation (:=)
>	Inférieur
<=	Inférieur ou égal
>	Supérieur
>=	Supérieur ou égal
<>	Différend
in	A l'intérieur (d'un ensemble)

9.2.5 Les tests

Ils permettent de prendre une décision en fonction de la valeur d'une variable.
Syntaxe*: if* condition *then* instruction *[else* instruction*]*

Exemple:

```
if age>65 then
        showmessage('retraité) // pas de ; car l'instruction n'est pas terminée
else
        begin // début de bloc
        showmessage('actif');
        MessageBeep(MB_ICONEXCLAMATION); // ou beep()
        end; // fin de bloc et fin d'instruction
```

Les instructions if peuvent être imbriquées.

9.2.6 Le sélecteur

Un sélecteur se comporte comme un rotacteur en électronique: il permet de faire un choix unique parmi un ensemble d'options. **Attention**, le sélecteur doit faire partie d'un **ensemble ordonné** (le système doit connaître le précédent et le suivant).

Syntaxe:

```
case sélecteur of
        cas1: instruction 1;
        cas2: instruction 2;
        (...)
        casn: instruction n
[else
        instruction]
end;
```

Exemple:

```
case Carac of
        '+': Op:='addition';
        '-': Op:='soustraction';
        '*': Op:='multiplication';
        '/': Op:='division';
else
        Op:='erreur';
end;
```

9.2.7 Les boucles

L'instruction if permet de faire des boucles, mais il en existe des "prêtes à l'emploi"

La procédure *Continue* provoque le passage du contrôle de l'exécution à l'itération suivante dans une instruction for, while ou repeat.

La procédure *Break* provoque l'interruption d'une boucle for, while ou repeat.

9.2.7.1 La boucle "pour" (for)

Elle est utilisée quand on connaît la valeur de départ, d'arrivée et de progression de l'indice de boucle.

Syntaxe: *for* variable=valeur_initiale [*down*] *to* valeur_finale *do* instruction;

Exemple: mise à zéro d'un tableau à deux dimensions de 100 cases:

```
for i:=0 to 9 do
        for j:=0 to 9 do
        Tableau[i,j]:=0;
```

Exercice: afficher une suite de 1 à 10 dans un mémo (à faire avec les 3 types de boucles).

9.2.7.2 La boucle "tant que" (while)

Cette boucle est maintenue par une condition. La sortie se fait dès que celle-ci cesse d'être vraie (true). Il n'est pas nécessaire de connaître par avance le nombre d'itérations

Syntaxe: *while* condition *do* [*begin*] instruction[s] [*end*] ;

Exemple:

```
while not fini do
begin
        travail1;
        travail2;
        ...
        travailn;
end;
```

Il est évident que la variable booléenne "fini" doit être réévaluée (il doit ^y avoir un moyen de savoir si le travail est terminé) au cours de la boucle, sinon elle ne prendra pas de fin.

Exercice: le même que précédemment.

9.2.7.3 La boucle "répéter" (repeat)

La boucle "répéter" est employée quand on souhaite faire d'emblée un parcours dans celle-ci. On ne connaît pas à priori le nombre de parcours de la boucle, mais il sera supérieur ou égal à 1. L'utilisation de *begin...end* est ici facultative.

Syntaxe: *repeat* instruction[s] *until* condition;

Exemple:

```
repeat
        prendre une pièce;
        la mettre dans le distributeur automatique;
until montant_introduit >= prix_article_choisi;
```

10 Quelques instructions

Voici quelques instructions utiles; il y en a d'autres à découvrir dans l'aide…

10.1 Arithmétiques

Abs	Renvoie la valeur absolue de l'argument
Ceil	Arrondit à l'entier supérieur
Exp	Fonction exponentielle
Flor	Arrondit à l'entier inférieur
Frac	Renvoie la partie fractionnelle de l'argument
Frexp	Renvoie la mantisse et l'exposant
Int	Renvoie la partie entière de l'argument
IntPower	Calcule la puissance entière d'une valeur de base
Ln	Renvoie le logarithme naturel de l'argument
Log10	Calcule le logarithme en base 10
Log2	Calcule le logarithme en base 2
LogN	Calcule le logarithme en base N
Max	Maximum de 2 valeurs
Min	Minimum de 2 valeurs
Pi	Renvoie 3.1415926535897932385
Power	Renvoie une élévation à la puissance
Round	Arrondi entier
Sqr	Renvoie le carré de l'argument
Sqrt	Renvoie la racine carrée de l'argument
Trunc	Partie entière

10.2 Date et heure

Date	Renvoie la date en cours.
DateTimeToStr	Convertit une valeur heure en chaîne.
DateTimeToString	Convertit une valeur de format heure en chaîne.
DateToStr	Convertit une valeur du format date en chaîne.
DayOfWeek	Renvoie le jour de la semaine.
DecodeDate	Décode la date spécifiée.
DecodeTime	Décode l'heure spécifiée.
EncodeDate	Renvoie les valeurs spécifiées au format date.
EncodeTime	Renvoie les valeurs spécifiées au format heure.
FormatDateTime	Formate une date et une heure en utilisant le format spécifié.
Now	Renvoie l'heure et la date en cours.
StrToDate	Convertit une chaîne en date.
StrToDateTime	Convertit une chaîne au format date/heure.
StrToTime	Convertit une chaîne au format heure.
Time	Renvoie l'heure en cours.
TimeToStr	Convertit un format heure en chaîne.

Toutes les routines de date et d'heure utilisent les variables de formatage Date/Heure.

10.3 Gestion fichiers

AssignFile	Fait correspondre un nom de fichier logique à un nom physique
ChDir	Change de dossier
CloseFile	Ferme le fichier (remplace "Close")
CreateDir	Crée un dossier
DeleteFile	Detruit un fichier
DiskFree	Renvoie l'espace disque disponible.
DiskSize	Renvoie la taille du disque spécifié.
FileClose	Ferme le fichier
FileDateToDateTime	Convertit la date du fichier
FileExists	Teste l'existence du fichier
FileGetAttr	Lit les propriétés du fichier
FileGetDate	Lit la date du fichier
FileOpen	Ouvre un fichier
FileRead	Lecture dans un fichier
FileSearch	Recherche un fichier
FileSeek	Change la position en cours dans le fichier.
FileSetAttr	Définit les attributs du fichier.
FileSetDate	Définit l'indicateur DOS de date et heure du fichier.
FileWrite	Ecrit dans un fichier spécifique.
FindClose	Termine une séquence FindFirst/FindNext.
FindFirst	Recherche un nom de fichier et un ensemble d'attributs spécifiés dans un répertoire.
FindNext	Renvoie la prochaine entrée correspondant au nom et aux attributs.
GetCurrentDir	Renvoie le dossier courant
GetDir	Renvoie le dossier courant sur le lecteur spécifié
RemoveDir	Supprime un dossier
RenameFile	Renomme un fichier
SetCurrentDir	Définit un dossier courant

10.4 Contrôle de flux

Abort	Permet de sortir d'un chemin d'exécution sans signaler d'erreur
Break	Sortie de boucle for, while ou repeat
Continue	Rebouclage dans une boucle for, while ou repeat
Exit	Sort immédiatement du bloc en cours.
Halt	Arrête l'exécution du programme et retourne au système d'exploitation: à éviter
RunError	Arrête l'exécution du programme.

10.5 Messages et dialogues

CreateMessageDialog	Crée un message de dialogue
InputQuery	Saisie de chaîne
LoginDialog	Saisie d'un nom et d'un mot de passe pour se connecter à une BDD
MessageDlg	Boite de dialogue
SelectDirectory	Choix d'un dossier
ShowMessage	Affiche une boîte de dialogue de message avec un bouton OK

10.6 Divers

Exclude	Exclut un élément d'un ensemble.
FillChar	Remplit un nombre spécifié d'octets contigus avec une valeur spécifié (de type Byte ou Char).
Hi	Renvoie l'octet de poids fort de l'argument.
Include	Inclut un élément dans un ensemble.
Lo	Renvoie l'octet de poids faible de l'argument
Move	Copie les octets d'une source dans une destination.
ParamCount	Renvoie le nombre de paramètres transmis au programme dans la ligne de commande.
ParamStr	Renvoie le paramètre spécifié de la ligne de commande.
Random	Renvoie un nombre aléatoire.
Randomize	Initialise le générateur intégré de nombres aléatoires avec une valeur aléatoire obtenue à partir de l'horloge système.
SizeOf	Renvoie le nombre d'octets occupés par l'argument.
Swap	Permute les octets de poids fort et de poids faible de l'argument.
UpCase	Convertit un caractère en majuscule.

10.7 Ordinaux

Dec	Décrémente une variable.
Inc	Incrémente une variable.
Odd	Teste si l'argument est un nombre impair.
Pred	Renvoie le prédécesseur de l'argument.
Succ	Renvoie le successeur de l'argument.

11 Méthodes: fonctions et procédures

Les méthodes sont des blocs de code débutant par une ligne de prototypage (déclaration de la méthode). Ensuite il faut déclarer les variables locales utilisées et implémenter le code entre les mots clés begin et end. On distingue deux types de méthodes: les fonctions et les procédures. Elles peuvent faire partie de la classe d'un objet ou être indépendantes dans le module en cours ou un autre module. On peut naturellement créer des modules "bibliothèques": c'est souvent le cas des DLL.

11.1 Fonctions

Une **fonction renvoie une donnée** dont le type est déclaré. Elle accepte généralement des paramètres. La structure est la suivante:

```
function NomFonction(para1:type;…):type;
  begin
    instructions;
  end;
```

Il est nécessaire de déclarer le type des paramètres ainsi que celui de la valeur retournée par la fonction. Dans le corps de la fonction, cette valeur retournée doit être définie soit en affectant une valeur au nom de la fonction ou en utilisant le mot-clé "result".

Nous allons créer un programme qui élève au cube la valeur saisie. Utilisons, pour changer, la fonction "InputBox" (voir dans l'aide en ligne) pour saisir une valeur et la fonction "ShowMessage" pour afficher le résultat. La fonction écrite dans la partie implémentation peut avoir le code suivant:

```
function Cube(valeur:extended):extended;
  begin
    Cube :=valeur*valeur*valeur;
  end;
```

Le compilateur travaillant du haut vers le bas, il est nécessaire que le code précédent se situe avant l'utilisation de la fonction. Il est possible de ne pas le faire, soit en déclarant la fonction (1° ligne du code: prototype) en fin de section "interface" (visibilité par les autres modules) ou en mettant ce prototype suivi de la directive "forward" au début de la section "implémentation"

Le listing complet est le suivant:

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls;

type
  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  private
    { Déclarations privées}
  public
    { Déclarations publiques}
  end;

var
  Form1: TForm1;

implementation
{$R *.DFM}

function Cube(valeur:extended):extended;forward;

procedure TForm1.Button1Click(Sender: TObject);
  var nombre,resultat: extended;
begin
  nombre:=StrToFloat(InputBox('Boîte de saisie', 'Taper un nombre',''));
  resultat:=Cube(nombre);
  ShowMessage('Le cube du nombre saisi est '+FloatToStr(resultat));
end;

function Cube(valeur:extended):extended;
  begin
    Result :=valeur*valeur*valeur;
  end;

end.
```

Essayer d'écrire la même fonction dans un nouveau module et de l'utiliser dans le module présent: attention à la clause uses.

1.1 Procédures

Les **procédures** sont utilisées pour effectuer des **actions**, mais peuvent également servir à renvoyer des valeurs. Comme les fonctions, elles peuvent accepter des paramètres transmis par valeur (option par défaut), par variables en lecture-écriture (précédées par le mot-clé "var") ou des constantes (précédés par le mot-clé "const"). La syntaxe en est la suivante:

```
procedure NomProcedure(arg1:type; arg2:type; ….argn:type);
  begin
    intructions;
  end;
```

Ecrire une procédure qui génère deux bips à une seconde d'intervalle:

```
procedure DeuxBips();
  begin
  beep();
  sleep(1000);
  beep();
  end;
```

L'appel à cette procédure peut être déclenchée par l'appui d'un bouton:

```
procedure TForm1.Button1Click(Sender: TObject);
  begin
  DeuxBips;
  end;
```

12 Les menus

12.1 Menu principal

La mise en place d'une barre de menus est simplifiée par le concepteur de menus. Placer sur la fiche un composant de type "TMainMenu". Double-cliquer sur lui pour accéder au concepteur de menus. Il suffit alors de taper les noms en validant avec la touche "entrée". Pour se déplacer, on peut utiliser la souris ou les flèches de direction.

Particularités:

Accès rapide	il suffit de placer le signe "&" dans le nom pour que la lettre suivante apparaisse en souligné
Séparation	taper le signe "-" à la place du nom
Touches de raccourci	Attribuer la propriété "shortcut" (attention aux doublons!)
Sous-menu	Taper Ctrl + flèche à droite
Déplacement d'éléments	par cliquer-glisser

On peut utiliser des modèles existants de menu ou créer des modèles personnels (voir menu contextuel).

Si l'on utilise des gestionnaires d'événements partagés, on peut les attribuer dans la page événements de l'inspecteur de propriétés.

Ensuite, fermer le concepteur de menus. Il est indispensable d'attribuer une procédure événementielle à chaque élément: ouvrir le menu et cliquer sur l'option. L'éditeur de code écrit tout seul l'entête de la procédure: il ne reste qu'à compléter.

Exercice: créer un projet de type bloc-notes (dans un nouveau répertoire "Notes"). Y mettre des menus (fichier, édition, ...). Utiliser les boîtes de dialogue préfabriquées (enregistrer, ouvrir).

12.2 Menus surgissants

Placer un composant de type TPopupMenu sur votre fiche. Appeler l'éditeur de menus par un double-clic sur lui. Compléter comme précédemment. Il faut ici affecter les événements avant de refermer le concepteur.

Utilisation: Affecter la propriété PopupMenu de l'un des composants de votre fiche avec le nom de ce menu.

13 Barres d'outils classiques

13.1 Mise en place de la barre

Placer un composant Tpanel sur la fiche. Effacer le contenu de la propriété "Caption". Choisir "alTop" pour la propriété "Align". Donner un nom significatif.

13.2 Mise en place des boutons

Il suffit de placer un composant Tspeedbutton sur la barre créée précédemment. La propriété "Glyph" permet de choisir une image au bouton. Des images existent dans le répertoire \Programme\Images\Buttons.

Il est facile de créer de nouvelles images grâce à l'éditeur d'images (menu outils). Choisir une taille 16X16 pour un bouton simple.

13.3 Info-bulles

Les info-bulles sont faciles à mettre en œuvre pour les boutons (ou autres composants). Mettre le texte à afficher dans la propriété "Hint" et positionner "ShowHint" à "true".

13.4 Affectation d'actions aux boutons

Généralement les gestionnaires d'événements existent déjà pour les menus (les boutons ne sont qu'un moyen pour procéder plus vite). Dans ce cas, il suffit d'affecter la procédure "OnClick" grâce à l'inspecteur de propriétés.

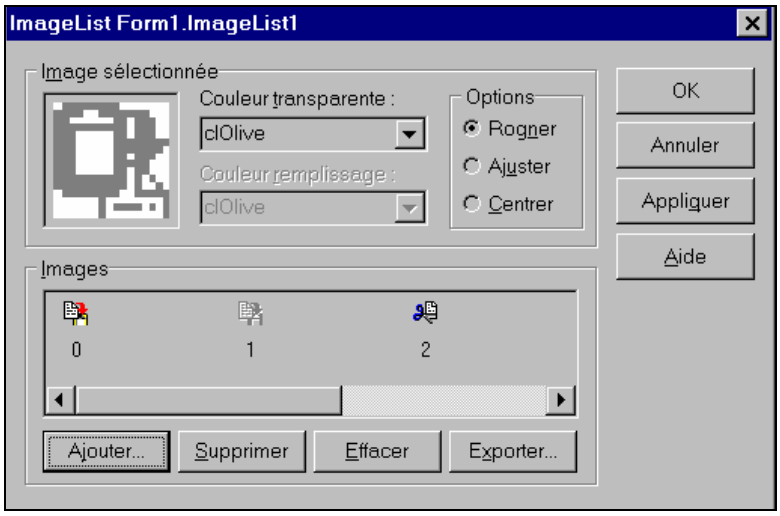
Pour écrire un gestionnaire, il suffit de double-cliquer sur le bouton.

Il est possible de grouper les boutons et de les rendre mutuellement exclusifs. Les boutons peuvent rester enfoncés et présenter une image différente dans ce cas.

Exercice: ajouter une barre d'outils au bloc-notes.

14 Barres d'outils "new look"

Placer un composant "ToolBar" et un composant "ImageList" sur la fiche. Double-cliquer sur ce dernier pour entrer dans l'éditeur d'images:



En utilisant ajouter, vous pouvez charger votre liste. Des bitmaps se trouvent dans le répertoire "C:\Program Files\Fichiers communs\Borland Shared\Buttons\". Ensuite il suffit de choisir "nouveau bouton" (ou séparateur) dans le menu contextuel du composant "Toolbar". La propriété "ImageIndex" du bouton définit l'image utilisée.

La propriété "Flat" de Toolbar permet d'avoir un effet de bouton surgissant comme dans certains logiciels récents bien connus.

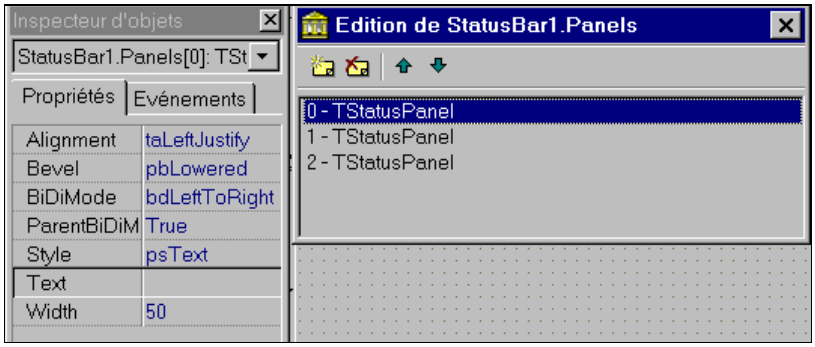
On peut aussi utiliser le composant "Coolbar" et placer des "Toolbar" garnis dans ses bandes redimensionnables (style internet).

15 Barre d'état

Le composant correspondant existe sur cette version de Delphi, il s'agit de TStatusBar. C'est une ligne de volets, généralement placée en bas d'une fiche, qui affiche des informations sur l'application en cours d'exécution. Chaque volet est représenté par un objet TStatusPanel énuméré dans la propriété Panels. La propriété SimplePanel peut être utilisée pour faire basculer, à l'exécution, la barre d'état entre l'affichage d'un seul volet et celui de plusieurs volets.

15.1 Mise en place

Placer un composant "TStatusBar" sur la fiche. Si l'on désire plusieurs volets, il faut attribuer la propriété "panels". L'éditeur de propriété se présente sous la forme suivante:



Choisir "Ajouter" pour augmenter le nombre de subdivisions. On peut accéder au contenu de chaque panneau en utilisant la propriété "text" de StatusBar1.Panels1[indice]. Les indices partent de la gauche avec la valeur zéro.

15.2 Exercice

Créer un nouveau projet. Mettre l'heure dans la seconde partie d'une barre d'état à 3 volets. Utiliser pour cela un timer système (voir l'aide en ligne pour l'utilisation). La fonction TIME donne l'heure (à convertir en texte).

Option: afficher "MAJ" dans la partie de gauche quand le verrouillage des majuscules est actif. Il faut intercepter la frappe des touches en mettant la propriété "KeyPrev" de la fiche à "true" (vrai). Le code de la touche est "VK_CAPITAL".

Voir la fonction "GetKeyState".

Liste des codes virtuels

CODE	TOUCHE
VK_LBUTTON	bouton gauche de souris
VK_RBUTTON	bouton droit de souris
VK_MBUTTON	bouton médian de souris (s'il existe!)
VK_BACK	retour arrière (backspace)
VK_TAB	tabulation
VK_RETURN	entrée
VK_PAUSE	pause
VK_CAPITAL	verrouillage majuscules
VK_ESCAPE	Echap.
VK_SPACE	espace
VK_PRIOR	page précédente
VK_NEXT	page suivante
VK_END	fin
VK_HOME	début (↖)
VK_LEFT	←
VK_UP	↑
VK_RIGHT	→
VK_DOWN	↓
VK_SNAPSHOT	impression écran
VK_INSERT	insertion
VK_DELETE	suppression
VK_0	zéro
...	
VK_9	9
VK_A	A
...	
VK_Z	Z
VK_NUMPAD0	zéro du pavé numérique
...	
VK_NUMPAD9	9 du pavé numérique
VK_MULTIPLY	multiplication
VK_ADD	addition
VK_SUBTRACT	soustraction
VK_DECIMAL	point décimal
VK_DIVIDE	division
VK_F1	F1
...	
VK_F12	F12
VK_NUMLOCK	verrouillage numérique
VK_SCROLL	arrêt défilement

16 Exercices proposés:

- créer un bloc notes amélioré, multi documents
- réaliser une calculatrice simple, puis à bande, enfin scientifique (puissance, factorielle, sinus, cosinus, tangente, binaire, hexadécimal, …)

17 Où trouver de l'aide?

- Aide en ligne
- Apprentissage en ligne de Delphi3 (S&SM): sur le CD de Delphi3
- Les manuels papier de Delphi4
- PASCAL (Philippe SPOLJAR chez Sybex poche)
- DELPHI Professionnel - Programmation système 32 bits (Dick LANTIM chez Eyrolles)
- DELPHI2 - Secrets d'Experts (Charles CALVERT chez S&SM)
- Outils de développement Delphi 1&2 (Gilles BETZ chez Sybex Mégapoche)
- DELPHI 3 (Dick LANTIM chez Eyrolles)
- Internet: www.inprise.com et les groupes de News

TABLE DES MATIERES