Introduction

Delphi représente une nouvelle façon de développer des applications sous Windows. Il associe la vitesse et la convivialité d'un environnement de développement visuel à la puissance d'un langage objet , au compilateur le plus rapide au monde et à une technologie de base de données de pointe.

Avec Delphi, vous avez maintenant la possibilité de construire des applications Client/Serveur en un temps record. Delphi permet de créer des fichiers .EXE indépendants, des fichiers .DLL (Bibliothèques de lien dynamique), des applications de base de données locales ou distantes, des applications OLE ainsi que des applications Client/Serveur. L'environnement de conception visuel de Delphi vous permet de créer des applications Windows 32 bits pour Windows 9x et Windows NT plus vite qu'avec n'importe quel outil de développement.

Tout au long de ces pages, vous allez apprendre à développer des applications Windows en Delphi. La simplicité du langage et le caractère visuel de l'environnement font de Delphi un des outils des plus conviviaux. En fait, la création d'une application en Delphi, consiste en grande partie dans la disposition des contrôles qui formeront l'interface. Votre application apparaît au fur et à mesure que vous ajoutez les objets, et telle qu'elle le sera à l'utilisateur. Delphi est, à ce propos, l'un des tous premiers langages de programmation à avoir intégré un environnement réellement **WYSIWYG** (What You See Is What You Get, ou tel écrit tel écran). Des projets « bonus » répartis tout au long de ce support vous invitent à mettre « la main à la pâte », pour créer des applications complètes et fonctionnelles.

Visual Basic est un langage *Basé* objet. D'autres langages sont dans ce cas. Ils se servent d'objets et de méthodes, mais ne répondent pas aux concepts fondamentaux des langages orientés objet qui sont l'encapsulation, l'héritage et le polymorphisme. Delphi est un véritable langage orienté objet. Il permet de combiner dans une classe des données et du code (Encapsulation), de créer de nouvelles classes à partir de classes parentes (héritage) et enfin d'altérer le comportement hérité (polymorphisme).

Les applications Windows apparaissent à l'écran formées d'éléments assez semblables. Par exemple, le bouton standard est représenté sous la forme d'un rectangle gris saillant comportant un texte spécifique en rapport avec sa fonction. Delphi reprend la fonctionnalité du bouton, la capacité de répondre à un clic de la souris et d'afficher du texte. Il l'emballe dans un objet appelé *composant*. Ce composant est stocké dans la bibliothèque de composants. Delphi est livré avec une importante bibliothèque qui contient tous les objets dont on a besoin pour créer un programme Win32 digne de ce nom.

La nature orientée d'objet de Delphi permet en cas de besoin d'ajouter des fonctionnalités ou de modifier le comportement d'un composant, en dérivant simplement un nouveau composant à partir de l'un de ceux qui se trouvent dans la bibliothèque.

Delphi se sert d'un compilateur et d'un lieur authentique qui produisent des exécutables en code machine pur. Utiliser un vrai compilateur présente encore l'avantage de pouvoir créer des DLL qui contiennent des composants provenant de la bibliothèque des composants. On pourra ensuite se servir de ces DLL pour étendre des

applications de Delphi, ou pour rendre service à des programmes développés avec des outils moins sophistiqués.

Enfin, Borland a pris conscience que la plupart des applications tournent autour des bases de données. Collecter des informations, les traiter et les restituer est le rôle de l'ordinateur. Comme les développeurs stockent en général les informations dans une base de données, Delphi propose des objets et des composants qui réduisent la quantité d'efforts nécessaires pour créer des applications orientés bases de données.

Attention

Ce support n'enseigne pas les bases de la programmation. Il apprend seulement à utiliser Delphi et suppose la connaissance des concepts fondamentaux de la programmation tels que variable, fonction et boucle.

I INTRODUCTION

A la fin de cette partie, vous serez capable de:

- Parler des caractéristiques de Delphi,
- Comparer Delphi avec d'autres environnements de programmation,
- Expliquer les principes de fonctionnement de Delphi.

I.1 Présentation

Delphi est un environnement de programmation permettant de développer des applications pour Windows 3.x, Windows 9x et Windows NT. Il incarne la suite logique de la famille Turbo Pascal avec ses nombreuses versions.

Le Turbo Pascal (DOS et Windows) ne subira apparemment plus de mises à jour. Delphi est un outil moderne, qui fait appel à une conception visuelle des applications, à la programmation objet, de plus, il prend en charge le maintien automatique d'une partie du code source.

Voici quelques unes des caractéristiques de Delphi:

- Programmation objet.
- Outils visuels bidirectionnels .
- Compilateur produisant du code natif .
- Traitement complet des exceptions .
- Possibilité de créer des exécutables et des DLL .
- Bibliothèque de composants extensible .
- Débogueur graphique intégré .
- Support de toutes les API de Windows: OLE2, DDE, VBX, OCX, ...

I.2 Delphi et les autres

La plupart des applications Windows sont encore écrites en C ou C++ pour des raisons essentiellement historiques. Microsoft pousse bien entendu le C++ du fait que l'environnement Windows lui-même conçu sous forme d'objets.

A la naissance de Windows, Microsoft donnait également la possibilité d'écrire des applications Windows en Pascal, mais probablement aucune application réelle et commerciale n'a jamais été écrite dans ce langage. Le premier changement majeur dans le développement sous Windows fut l'apparition de Visual Basic:

- Charmeur par sa relative facilité de programmation
- Pénalisé par un passé lourd de catastrophes dues à la moitié de son nom : *Basic*
- Décevant par ses performances.

Visual Basic a rendu d'immenses services à ceux qui ont su l'utiliser à bon escient, c'est-à-dire pour ce qu'il sait faire. Un certain nombre d'applications à large diffusion sont écrites à base de Visual Basic; par exemple *Visio 5*. Visual Basic a comblé une lacune, détourné quelques programmeurs C, mais surtout permis à d'autres programmeurs d'entrer en scène.

Alors pourquoi Delphi? La question a un sens car, à première vue, en comparant Delphi et Visual Basic, on penserait plutôt à du plagia. En réalité il n'en est rien. Par sa conception, son environnement de développement et ses performances, Delphi fait de plus en plus d'adeptes.

C'est sa courte histoire qui tend à le prouver, au travers de quelques constatations:

✍ De plus en plus, des journaux et revues publiant habituellement des programmes (souvent des utilitaires) écrits en C, se convertissent à Delphi. C'est le cas par exemple du très répandu PC Magazine.

✍ Les chiffres des ventes ont même réussi à sauver Borland (l'éditeur de Delphi), au-delà de toute attente réaliste.

✍ De l'avis quasi unanime Delphi surpasse largement Visual Basic, même et surtout on compare sa dernière version 32 bits et le nouveau Visual Basic 6.

✍ SUN, créateur de Java et HotJava, a fait appel à Borland pour créer des composants Delphi permettant de construire des applications Java sur Internet.

Et le C ? Mis à part des questions de langage (donc de syntaxe) est-il vraiment plus performant qu'un outil comme Delphi ? Il vaut mieux éviter tout affrontement. En tout cas, les compilateurs Turbo C++, Borland C++ produits par Borland ont toujours été parmi les plus performants du marché. Or il se trouve que Borland a utilisé le même noyau de compilateur pour son Delphi et pour son C++. Par exemple, le code généré est aussi optimisé pour l'un que pour l'autre. Partant de là, les performances doivent probablement être comparables. La vie d'un développeur Windows ne se résume plus qu'à un choix de langage.

Il y a bien d'autres environnement de développement à part les trois cités ci-dessus, mais leur faible diffusion les marginalise.

I.3 Principes de développement en Delphi

Delphi fait évidemment partie de la famille de la programmation destructive, comme ne peuvent que l'être les langages de développement modernes sous Windows. On ne peut plus se permettre d'attendre des années avant de découvrir (ou d'apprendre par cœur) que l'objet "barre de défilement verticale" possède telle ou telle propriété. Les propriétés, entre autres, des objets doivent être immédiatement et toujours visibles au programmeur. Pour construire l'interface d'une application, ce dernier place des objets sur une fiche "fenêtre" et les personnalise en modifiant éventuellement leurs propriétés et/ou en leur attachant des instructions liées à des événements donnés.

Bien que pour certaines application (souvent montrées comme exemples) il ne soit pas nécessaire d'écrire du code (ni de connaître le Pascal), il vaut mieux avoir une solide expérience de la programmation avant de se lancer dans un développement réel.

I.4 Delphi et Windows

Delphi permet de créer et de manipuler tout objet de Windows. Tous les objets et une grande partie de l'API de Windows sont encapsulés dans des composants Delphi. Les messages Windows sont redirigés par Delphi vers les objets auxquels ils sont destinés. Dans tous les cas, il est toujours possible de faire appel directement à l'API de Windows.

II ENVIRONNEMENT DE TRAVAIL

A la fin de cette partie, vous serez capable de:

- Connaître les composants de base de Delphi.
- Utiliser les différentes options des menus.
- Reconnaître les divers types de fichiers qui constituent un projet.
- Distinguer entre propriétés et événements concernant les composants Delphi.

II.1 L'interface de développement de Delphi

Examinons l'interface de développement de Delphi

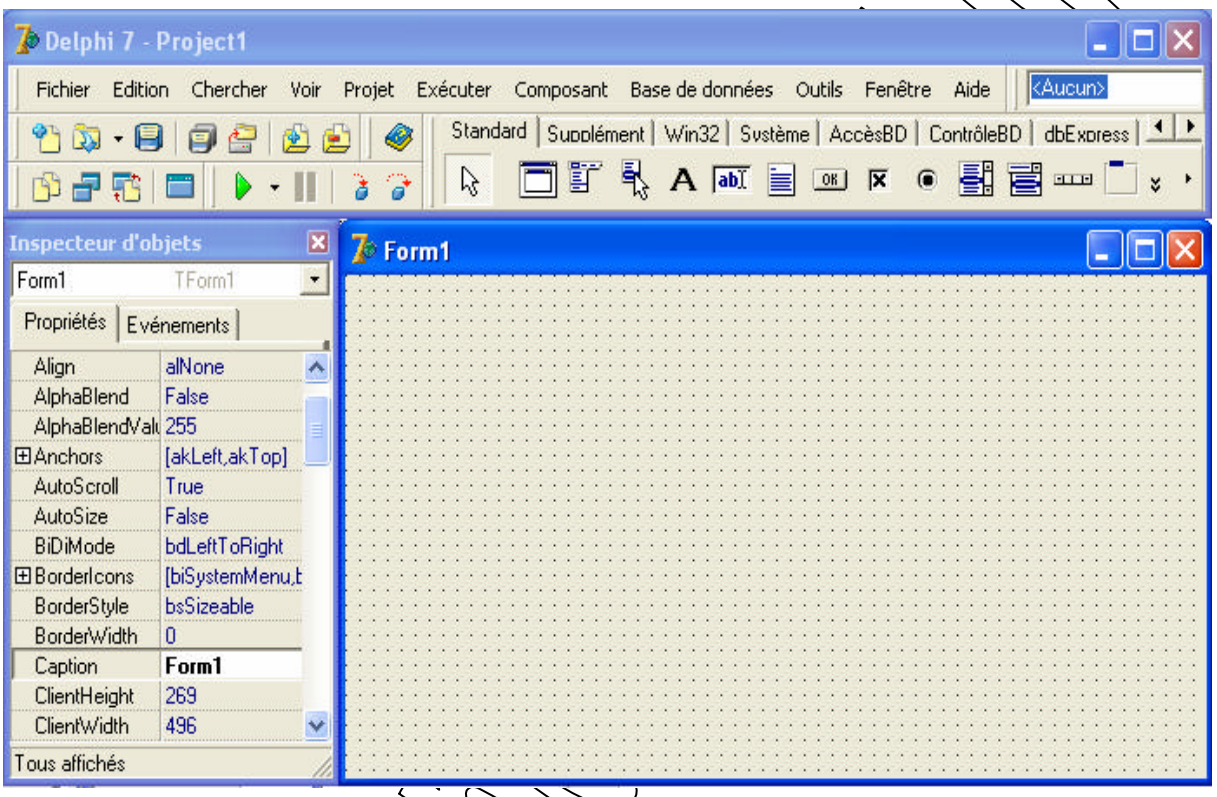


Figure II.1 : Interface de Delphi

L'interface de développement se divise en trois zones :

- Dans la partie supérieure de l'écran figure la fenêtre de programme, la barre d'outils ainsi que la palette des composants. Les fonctions concernant le projet sont situées à gauche dans la barre d'outils, à droite se trouve la palette des composants.
- Sous la fenêtre de programme à gauche, figure l'inspecteur d'objets. Il vous permet de définir l'aspect et le comportement de composants.
- A droite de l'inspecteur d'objets, Delphi affiche la première fenêtre de programme. Il attend que vous la définissiez. A chaque fenêtre correspond un texte source qui détermine son fonctionnement.

II.2 La barre de menu

Au dessous de la barre de titre et sur toute sa largeur s'étale un ruban de commandes appelé barre de menu. Chaque en tête de menu possède une lettre souligné qui le rend accessible avec le clavier (Alt et la lettre souligné du menu).

Le tableau suivant présente la structure des menus de Delphi et les commandes les plus importantes:

Menu	*Fonction*
Fichier	Ce menu contient les fonctions permettant de créer, ouvrir et enregistrer des fichiers.
Fichier/Nouveau	Créez ici les différents fichiers, fiches vides et les éléments de programmes.
Edition	Ce menu vous propose les commandes qui permettent d'exploiter le presse-papiers, il contient aussi quelques fonctions de définition des fenêtres.
Chercher	Ici figurent les fonctions de recherche et de remplacement.
Voir	Concerne les programmes de grandes tailles, pour en examiner la structure et recevoir des informations.
Projet	Créez ici les fichiers exécutables à partir des textes sources. Vous recevrez des informations sur le programme et pourrez définir certains paramètres et options.
Exécuter	Ce menu propose les fonctions pour lancer et exécuter en pas à pas les programmes.
Exécuter/Exécuter	Cette commande lance des applications, Delphi fonctionnant en tâche de fond.
Composants	Modifiez à l'aide des commandes de ce menu la palette des composants, intégrez des composants supplémentaires.
Base de données	Ce menu propose les assistants au développement de base de données.
Outils	Définissez ici les différents paramètres de Delphi, appelez des outils externes.
Aide	Accédez directement à l'aide de Delphi.

II.3 La barre d'outils

La barre d'outils permet d'accéder directement avec la souris à certaines commandes des menus.

Figure II.2: La barre d'outils

Elle contient les boutons permettant d'enregistrer et d'ouvrir des fichiers, de basculer d'une fiche ou d'une unité à une autre, d'ajouter/supprimer une fiche au/du projet, de rédiger le texte source, de basculer entre la fiche et l'unité qui lui est associée et d'exécuter pas à pas un programme. Les barres d'outils de Delphi sont personnalisables, ce qui signifie qu'on peut l'adapter à nos besoins en rajoutant ou en supprimant des boutons.

II.4 La palette des composants

La palette des composants est un élément extrêmement important. Les composants sont des éléments à partir desquels les programmes Delphi sont créés. Citons les menus, les zones de saisie et les boutons.

Delphi s'accompagne de composants. Ceux-ci figurent dans une palette, ce qui vous permet de les intégrer rapidement dans vos programmes.

La nature orientée objet de Delphi fait que, chaque composant étant un objet, on pourra s'on servir comme base pour un nouveau dont le comportement sera enrichi ou modifié. On pourra ensuite ajouter le nouveau composant à la bibliothèque des composants et le réutiliser. La commande Liste de composants du menu Voir permet d'accéder à tous les composants.

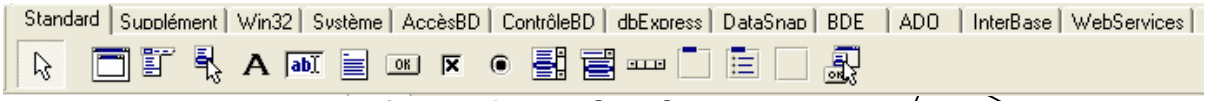


Figure II.3: La palette des composants

Les composants sont répartis en pages, classées par domaine d'application que nous découvrirons au fur et à mesure dans ces pages.

La configuration par défaut des pages de la palette des composants est la suivante:

- **Standard**: cette page contient des composants élémentaires qui étaient déjà présents dans les premières versions, comme TMainMenu, TButton, TListBox et TLabel.
- **Supplément**: cette page contient des composants plus spécialisés qui sont apparus dans des logiciels plus récents, comme TSpeedButton, TStringGrid, TImage. Le composant TSplitter mérite une attention particulière: il permet de concevoir une interface comportant des sections dont la taille est ajustable, comme c'est le cas pour la barre d'outils et la palette de composants dans la fenêtre principale de Delphi ou même l'Explorateur de Windows. TChart est aussi une nouveauté dans cette page. Il permet d'incorporer facilement dans vos programmes en Delphi des graphes de qualité professionnelle.
- **Système**: cette page contient des composants dont la fonctionnalité repose essentiellement sur le système d'exploitation. le composant TTimer en est l'exemple: il exécute une fonction à intervalles réguliers, dont la valeur est définie par l'utilisateur. c'est le système d'exploitation qui mesure les intervalles.
- **Win32**: cette page contient les nouveaux composants introduits par Windows 95 et les dernières versions de NT. Il s'agit des composants TTreeView et TListView.
- **Internet**: cette page contient des composants qui permettent de produire des applications pour Internet. Il s'agit, par exemple, des composants HTTPApp, HTTPProd et TcpClient.
- **AccèsBD**: cette page propose aux utilisateurs des composants qui leur facilitent l'usage des informations concernant les ensembles de données. Il s'agit par exemple de TTable et TSession. Ce ne sont pas des composants visuels dans la mesure où ils ne s'affichent pas au moment de l'exécution.
- **ContrôleBD**: cette page contient des composants appelés des contrôles orientés données. Ils fournissent une interface visuelle pour les composants non visuels de la page AccèsBD. Il s'agit par exemple des composants TDBGrid, TDBNavigator.
- **QReport**: cette page fournit une série de composants dont on se sert pour créer des états sans les complications de ReportSmith.
- **Dialogues**: cette page contient les composants qui encapsulent la fonctionnalité des boites de dialogues de Windows, comme TOpenDialog et TSaveDialog. Vous l'utiliserez pour vous conformer à l'esthétique standard de Windows.
- **ADO :** cette page vous permettent de vous connecter aux informations des bases de données via ADO **(ActiveX Data Objects)**

- **InterBase** : cette page vous permettent de vous connecter directement à une base de données *InterBase* sans utiliser de moteur tel que le *BDE* ou *ADO* (**Active Data Objects**).
- **BDE** : cette page vous permettent de vous connecter aux informations des bases de données via le BDE (**Borland Database Engine**).
- **Exemples**: cette page propose des exemples de composants personnalisés. Le composant TGauge qui s'y trouve n'est proposé qu'à des fins de comptabilité. La page Win32 contient une barre de progression qui se conforme à la nouvelle interface Windows.
- **ActiveX**: cette page contient un certain nombre de composants qui illustrent l'aptitude de Delphi à employer les contrôles OLE et ActiveX.

II.5 L'inspecteur d'objet

Il est l'un des trois outils les plus précieux dont vous disposez pour la conception de l'interface. les deux autres, le concepteur de fiches et l'éditeur de codes seront traités plus loin dans ce chapitre.

Deux onglets vous donnent accès aux propriétés et aux événements du composant. Elles déterminent son aspect et son comportement. Vous pouvez fixer ces valeurs et les modifier au stade de l'exécution.

Les propriétés sont des informations qui appartiennent à un objet et qui le décrivent. La couleur d'une voiture est une propriété de l'objet "voiture". Les objets et les composants de Delphi possèdent aussi des propriétés, comme la largeur du composant TButton ou le texte affiché par TLabel.

L'inspecteur d'objets contient une liste déroulante qui se trouve en haut de la fenêtre et qui dresse la liste des composants de la fiche que vous êtes en train de concevoir. Vous pouvez sélectionner un composant dans cette liste, ou vous servir de la souris pour le sélectionner sur la fiche.

Le reste de la fenêtre de l'inspecteur d'objets dresse, sur deux pages séparées, la liste des propriétés du composant actuellement sélectionné:

- **Propriétés**: dresse la liste des propriétés publiées (*published*) du composant, par exemple, la largeur ou la police employée pour afficher le texte.
- **Evénements**: dresse la liste d'un type particulier de propriétés publiées, les événements. Un événement est une propriété qui ne contient pas d'informations concernant le composant. Elle détient un lien vers un segment de code en *Pascal Object* qui s'exécutera lorsque l'événement sera déclenché.

Chaque page est elle même divisée en deux colonnes, la gauche contenant les noms des propriétés et la droite les valeurs des propriétés

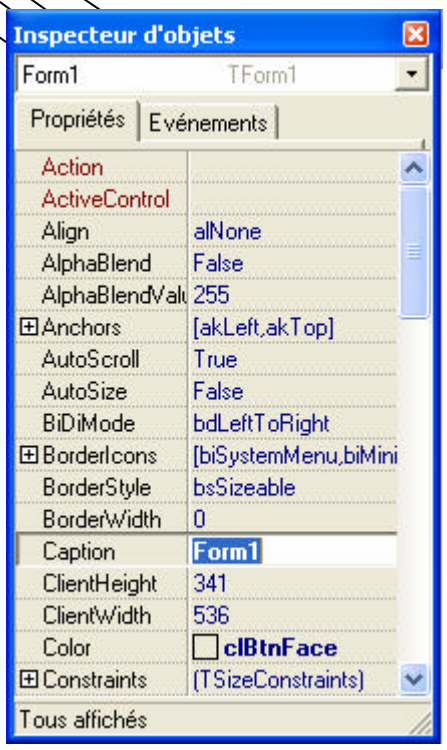


Figure II.4: l'inspecteur d'objets

II.5.1 Modifier les propriétés

Un composant digne de ce nom initialisera ses propriétés lorsqu'on le crée. Le composant se trouve donc toujours dans un état valide. Comment pourrait-on afficher un composant TButton dont la propriété couleur de fond ne serait pas définie?

L'inspecteur d'objets permet de changer ces paramètres initiaux. Les changements seront enregistrés dans le fichier DFM (pour *Delphi ForM*, ou fiche Delphi) de la fiche. Le compilateur s'en servira pour générer automatiquement du code qui affectera les nouvelles valeurs à la propriété au moment de l'exécution.

Il existe quatre types d'éditeurs de propriétés:

- **Texte**: la valeur de propriété est comme un texte qu'on édite. Ce type d'éditeur de propriétés sert pour les propriétés numériques comme Width et les propriétés de type chaîne de caractères comme Caption.
- **Enumération**: ce type de propriété peut prendre une seule valeur dans un ensemble prédéfini. L'inspecteur d'objets affiche une liste déroulante des valeurs possibles. La propriété Cursor est, par exemple, une valeur à énumération: un composant emploie tel curseur plutôt qu'un autre parmi ceux qui sont disponibles pour l'application.
- **Ensemble**: une propriété de ce type peut prendre plusieurs valeurs dans un ensemble prédéfini. L'inspecteur d'objets affiche ce type de propriétés dans une liste extensible de couple True/False. Il s'agit, par exemple, de la propriété BorderIcons de TForm: on peut inclure ou exclure toute icône susceptible d'apparaître sur la bordure de la fenêtre.
- **Objet**: une propriété de ce type est elle-même un objet, elle est affichée dans une liste extensible dont on pourra éditer chaque propriété. Certains objets possèdent leurs propres éditeurs de propriété. Ce sont des boites de dialogues qui permettent d'éditer toute la propriété d'un coup. On active ces propriétés personnalisées soit par un double-clic sur la propriété, soit en cliquant sur le bouton libellé par un ellipsis (...). La propriété Font fourni l'exemple des deux méthodes: on peut la développer dans une liste et elle possède un éditeur personnalisé.

II.5.2 Les propriétés événement

Les applications Windows sont commandées par les événements: les différentes parties de l'application s'exécutent en réponse à la situation du moment. On parle de **gestion d'événements**. La plupart des composants fournissent déjà une gestion d'événements élémentaires. Le clic sur un bouton déclenche un événement qu'on peut intercepter et utiliser pour exécuter un code en Pascal Object afin de fermer une fenêtre, enregistrer un fichier, supprimer un enregistrement ...etc.

On intercepte un événement en attribuant le nom d'une procédure en Pascal Object à la propriété d'événement. lorsque le composant détectera un événement, par exemple, un clic de la souris, il appellera la procédure ainsi référencée.

La propriété événement emploie un éditeur de propriété à énumération. L'événement ne peut exécuter qu'une seule procédure parmi l'ensemble des procédures présentes dans l'unité qui est associée à la fiche.

II.6 Les éditeurs

Delphi se sert d'un fichier DFM pour stocker les propriétés des composants. Le concepteur de fiches affiche ce fichier. Le code associé à la fiche, par exemple la définition de la fiche ou les procédures d'événements, est stocké dans une unité de Delphi ou dans un fichier PAS (pour Object PAScal ou Pascal Object). L'éditeur de code affiche ce fichier.

Delphi propose pour le développement deux éditeurs :

- L'éditeur de fiche qui permet de construire les fenêtre de l'application,
- L'éditeur de texte source comprenant l'ensemble des commandes qui constituent le programme.

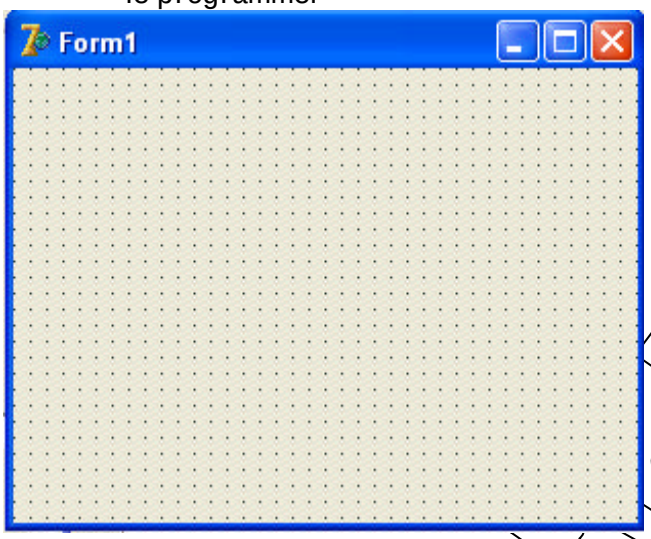


Figure II.5: Editeur de fiches

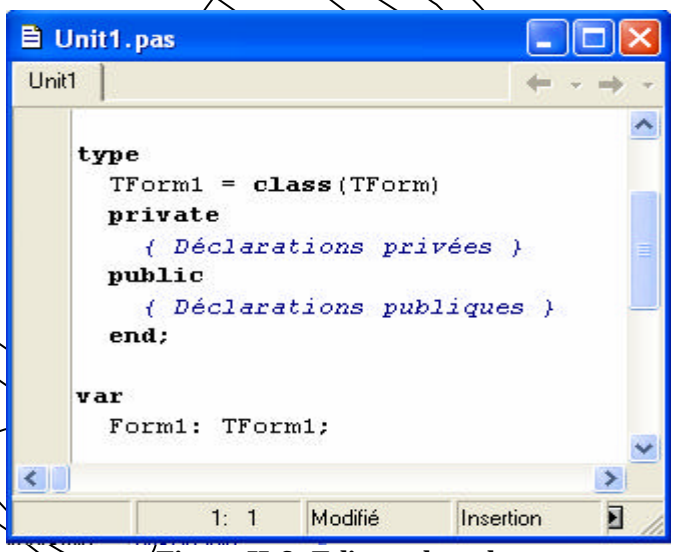


Figure II.6: Editeur de codes

II.6.1 Editeur de fiches

Les fiches sont les fenêtres des applications. L'éditeur de fiche vous permet de les construire à l'aide de la souris. Vous emploierez les composants de la palette et l'inspecteur d'objets.

Le concepteur de fiches n'est pas un programme distinct s'inscrivant dans l'EDI, mais ce terme désigne l'aperçu de la fiche au moment de la conception, par opposition, à celle qu'on aura au moment de l'exécution. Les composants non visuels comme TTable et TTimer apparaîtront dans le concepteur de fiche , et l'on pourra les sélectionner et les manipuler avec l'inspecteur d'objets. pendant la conception, il est possible d'afficher plusieurs fiches à la fois, une seule pouvant être active. En collaboration avec l'inspecteur d'objets, le concepteur de fiches permet d'effectuer les opération suivantes:

- Ajouter des composants à une fiche.
- Modifier les propriétés de la fiche et de ses composants.
- Relier les gestionnaires d'événements du composant au code Pascal Object que contient l'éditeur de code.

II.6.2 L'éditeur de texte source.

C'est avec cet éditeur que votre application commence à prendre corps. Avec le concepteur de fiches et l'inspecteur d'objets, vous pouvez donner libre cours à votre imagination, mais vous n'obtiendrez jamais qu'une interface.

Le texte source contient les commandes qui constituent le programme. Texte source et fiches sont transformés en un programme exécutable par l'environnement de programmation.

A chaque fiche correspond un fichier source, appelé Unité. Nous programmerons dans cette unité l'aspect et le comportement de la fiche et de chacun de ses composants. Les commandes du texte source sont traitées à l'exécution du programme. L'inspecteur d'objets ne permet de paramétrer les composants que pendant la conception.

On peut personnaliser l'éditeur de code en passant par le boîte de dialogue Options d'environnement (menu Outils).

On peut déjà constater que Delphi génère automatiquement le code correspondant à la fiche principale Form1. Dans Delphi, chaque fiche est décrite comme un objet (associé à une classe).

Delphi applique un principe fondamental lors de la création d'applications: ***Chaque fiche correspond à une unité.***

On peut voir sur la figure II.6 que Delphi a automatiquement déclaré la variable Form1. Cette variable définit un objet qui est une instance de la classe *T* form1.

Avant d'aller plus loin, il faut insister sur un point essentiel de l'utilisation de Delphi. Pour le confort de l'utilisateur, Delphi donne automatiquement un nom à chaque nouvel objet ou composant. Il s'agit d'un nom par défaut. Par exemple Form1, Button1, Button2, Listbox1... On peut bien entendu travailler avec ces noms, mais cette manière de faire est très fortement déconseillée. Donc: ***Il est impératif de donner un nom (explicite) à chaque nouvel objet***. Au lieu de Button1 on aura, par exemple, Quitter; au lieu de Listbox1 on aura, par exemple Couleurs. Les raisons pour se plier à cette discipline deviennent évidentes au fur et à mesure que l'on travaille avec Delphi.

II.7 Structures des fichiers du projet Delphi

Une application réalisée sous Delphi génère les fichiers ci-après :

Fichier	*Contenu de fichier*
*.dpr	Ce fichier contient le code source du fichier principal
*.dfm	Ce fichier contient les paramètres des fiches Delphi
*.pas	Extension des unités. Généralement elles appartiennent à une fiche dont elles définissent l'aspect et le comportement là l'exécution
*.dcu	Une unité compilé de cette extension contient les routines de l'unité de même nom et les données des fiches correspondantes
*.exe	Le fichier Exe contient le programme compilé, exécutable indépendamment de Delphi
*.res	Fichier des ressources du projet telles que les boutons et les images.

III NOTIONS DE BASE SUR LA POO

Le concept de l'orientation objet triomphe depuis quelques années déjà. il améliore la programmation structurée en allant au delà des fonctions et des procédures pour former un système encore plus souple et plus facile à étendre. Dès lors que l'on utilise Delphi, on ne peut éviter la programmation orientée objet (POO) car tous les composants, ainsi que diverses structures de base comme les exceptions, reposent sur cette technique.

La notion d'objet repose sur une entité physique (une classe), des propriétés, des méthodes et des événements.

Propriétés: Définissent les caractéristiques physique de l'objet.

Méthodes: Ensemble de procédures et fonctions intégrées à la structure de la classe.

Evénements: Ensembles d'actions auxquelles réagit l'objet. Ils sont exposées comme des méthodes protégées afin que les objets puissent appeler le gestionnaire d'événement approprié pendant son fonctionnement normal.

Les concepts fondamentaux de la POO sont : encapsulation, héritage et polymorphisme.

III.1 Encapsulation :

Les objets de la POO sont des entités concrètes à l'intérieur d'un programme, telles les zones de saisie, les fiches ou les messages d'erreur. Un objet présente certaines propriétés et peut effectuer certaines actions. Prenons par exemple une zone de saisie. L'une des ses propriétés est constituée par le texte affiché. Elle est par ailleurs capable de changer de contenu. Les propriétés sont représentées par des variables, les actions sont assurées par des fonction ou des procédures.

La séparation des variables qui mémorisent les propriétés et des routines qui les modifient est appelée Encapsulation. Elle sert à maintenir la flexibilité d'utilisation des objets tout en évitant les erreurs de mise en œuvre.

III.2 Héritage :

L'héritage consiste pour une classe à reprendre les méthodes et les variables d'une autre classe. Il n'est plus nécessaire de procéder à de nouvelles déclarations si elles sont déjà effectuées dans la classe dite de base.

L'héritage permet à des classes de reprendre méthodes et variables des classes ancêtres. Elles n'ont pas besoin d'être redéclarées et peuvent être mises en service comme leurs modèles.

Grâce à ce mécanisme, on peut commencer par définir des classes générales, d'où l'on dérive progressivement des classes plus spécialisées. Une classe donnée peut engendrer autant de classe filles que l'on désire, qui peuvent être à leur tour des classes de base pour d'autres classes.

Polymorphisme :

Le polymorphisme donne la possibilité d'utiliser des méthodes de même nom pour implémenter des instructions différentes.

Grâce au Polymorphisme, une méthode garde son nom tout en changeant de contenu. On peut ainsi invoquer de manière uniforme des méthodes appartenant à des

classes différentes. Les instructions exécutées ne sont pas les mêmes mais cet aspect des choses échappe à l'appelant.

En d'autres termes, on peut appeler une méthode sans connaître le détail des instructions exécutées qui s'adapteront à la classe concernée.

La méthode `Create` constitue un exemple de méthode Polymorphe. Elle est chargée d'instancier une classe et d'initialiser l'objet créé. Il est facile d'imaginer que ces opérations sont souvent très différentes d'une classe à une autre. Un objet fenêtre ne possède pas les même propriétés qu'un bouton. Malgré tout, l'un et l'autre sont créés par la méthode `Create`, qui exécute à chaque fois le jeu d'instructions approprié.

III.3 Droits d'accès :

Pour permettre au programmeur de spécifier quelles sont les variables et méthodes d'un objet qui sont ou non accessibles de l'extérieur, Pascal Objet dispose de quatre mots clés:

Mot clé	***Accès autorisé***
Public	Les variables et méthodes déclarées publics **(Public)** sont accessibles sans restrictions depuis l'extérieur.
Published	Les variables et méthodes dites publiées **(Published**) se comportent à l'exécution comme leurs équivalent publics. Mais elles sont déjà accessibles au moment de la conception (**exemple** : l'inspecteur d'objets).
Protected	Les variables et méthodes déclarées protégées **(Protected**) ne sont accessibles que de l'intérieur de l'objet et de ses dérivés.
Private	Les variables et méthodes déclarées privées **(Private**) sont accessibles de l'extérieur, mais uniquement à partir de l'unité qui les abrite.

Ces spécificateurs servent à fixer les droits d'accès souhaités.

IV LE LANGAGE PASCAL DE BASE

A la fin de cette partie, vous serez capable de:

- connaître la structure du langage Pascal (non objet);
- utiliser les divers types de données du Pascal;
- utiliser des instruction séquentielles, itératives et sélectives.

IV.1 La programmation structurée

Un ordinateur peut être assimilé à un système produisant des résultats à partir d'informations fournies et de "marches à suivre" permettant de les traiter. Les informations sont constituées par des données, et les méthodes de traitement par des algorithmes. Pour obtenir des résultats, la description des données et les algorithmes doivent être codés sous forme de programmes interprétables par l'ordinateur. En effet, le processeur de celui-ci ne peut exécuter qu'un nombre relativement restreint d'instructions élémentaires (le code machine).

Les programmes sont donc le résultat d'une succession d'étapes comprises entre la spécification informelle du problème et sa codification. Il y a ainsi entre ces deux pôles un "trou" qu'il s'agit de combler. Parmi les moyens ou les outils permettant d'y parvenir on peut citer notamment des environnements de production de logiciel (par exemple Delphi), des méthodes fournissant un encadrement au concepteur ou encore des langages de spécification permettant de préciser les étapes intermédiaires. Un autre aspect du rapprochement de la phase de codification vers la spécification du problème est constitué par le développement ou l'utilisation de langages de programmation permettant un niveau d'abstraction plus élevé.

Un des objectifs de la programmation structurée est la conception de logiciel fiable, efficace et d'une maintenance plus aisée. Il peut être atteint de manière asymptotique et par divers moyens. Un programme n'est pas juste ou faux; sa qualité est une notion globale, constituée par plusieurs éléments, dont nous allons étudier les plus importants.

La fiabilité est une propriété informelle et parfois difficile à cerner. Cette propriété peut être atteinte grâce à deux qualités du langage de programmation. D'abord, la facilité d'écriture doit permettre d'exprimer un programme de façon naturelle ou en termes du problème à résoudre. Le programmeur ne doit pas être dérangé par des détails ou des habitudes du langage, mais doit pouvoir se concentrer sur la solution recherchée. Les langages modernes de haut niveau tendent vers cet objectif. Ensuite, la lisibilité du programme doit permettre d'en saisir aisément la construction logique et de détecter plus facilement la présence d'erreurs. Dans cette optique, l'instruction Goto, par exemple, rend difficile la lecture du programme de façon descendante. Toutefois, dans certains cas, les objectifs énoncés au début de cette section peuvent être atteints dans de meilleures conditions par l'utilisation d'un Goto (bien placé et bien documenté) plutôt que par une construction structurée; sa présence est alors acceptable. De telles situations sont toutefois extrêmement rares.

Si l'efficacité était au début l'objectif principal de la conception d'un programme, actuellement cette notion a évolué pour englober non seulement des critères de vitesse d'exécution et de place mémoire, mais aussi l'effort requis par la maintenance du logiciel.

En effet, la nécessité de la maintenance impose au logiciel qu'il soit lisible et modifiable. Ces qualités sont souvent liées à des critères esthétiques. On peut néanmoins citer quelques facteurs qui facilitent les interventions dans un programme: un découpage approprié, une mise en page claire, un choix adéquat des noms d'objets utilisés, des commentaires cohérents placés judicieusement.

IV.2 Les variables

Une variable est une zone mémoire. Pour accéder à cette zone mémoire, nous utilisons dans le code que nous tapons, un mot pour l'identifier. Ce mot est le nom de la variable. Lors de la compilation, Delphi a besoin de savoir quelle quantité de mémoire, il doit réserver. Nous devons commencer à déclarer la variable (à l'aide du mot clé Var) en indiquant son type.

Exemples de déclarations :

```
var Num: integer;
var Nom : string;
```

Les deux variables ainsi déclarées ne sont pas du même type, la première permet de retenir un nombre entier (positif ou négatif) alors que la deuxième est faite pour retenir une chaîne de caractères (c'est-à-dire une suite de caractères lettres, chiffres ou signes…).

Pour mettre une valeur dans une variable (on dit affecter une valeur à la variable), il faut indiquer :

A gauche le nom de la variable
Suivi du symbole :=
A droite une valeur du bon type (sous forme simple, ou sous forme à calculer).

Exemples d'affectation :

```
Num := 123;
Matricule := -20549;
Resultat := 15+17*3;
Message := 'Etude de la programmation avec Delphi.';
```

IV.3 Où peut-on placer les déclarations?

Les déclarations peuvent être placées dans une procédure ou une fonction, avant le mot begin. Les affectations peuvent être placées dans une procédure ou une fonction, après le mot begin et avant le mot end; Nous verrons plus tard que d'autres places sont possibles.

Vous pouvez par exemple modifier la procédure Button1Click comme suit :

```
procedure TForm1.Button1Click(Sender: TObject);
var UneVariable: integer;
var UneAutre: string;
begin
     UneVariable:=53;
     UneAutre:='Bonjour, ceci est une chaîne.';
     ShowMessage(UneAutre);
end;
```

Remarques :

- le point-virgule, est nécessaire après chaque instruction pour la séparer de la suivante.
- La variable nommée UneVariable ne sert à rien, elle n'est jamais utilisée, aussi Delphi ignore tout ce qui concerne UneVariable lors de la compilation afin d'optimiser le code dans le fichier exécutable créé.

IV.4 Les types

IV.4.1 Généralités sur les types

Lorsque vous déclarez une variable, le compilateur a besoin de savoir quelle place il doit lui réserver en mémoire. En effet, il ne faut pas la même place pour retenir un nombre entier et une chaîne.

Avec Delphi, il faut 4 octets pour retenir un **Integer**, alors qu'un **String** nécessite 9 octets en plus des caractères composant la chaîne. De plus Delphi a besoin de connaître les opérations qu'il peut effectuer avec la variable. Il peut multiplier des entiers mais pas des chaînes !

Un type est une sorte de modèle de variable mais ce n'est pas une variable. Un type ne prend pas de place en mémoire, il permet seulement au compilateur de savoir quelle place il faut prévoir pour chaque variable déclarée avec ce type et d'interdire toute opération qui ne conviendrait pas pour ce type.

Par exemple

```
Type Str80 = string[80];
```

Signifie que nous venons de définir un type nommé Str80 qui nous permettra de retenir une chaîne de 80 caractères maximum. Aucune place mémoire n'est allouée à ce stade.

La déclaration de type peut être placée dans la partie **Interface** ou dans la partie **Implémentation** ou encore au début (avant le Begin) d'une procédure ou d'une fonction.

En ajoutant les lignes

```
Var S1 : Str80;
Var S2 : Str80;
```

Delphi réserve la mémoire suffisante pour retenir 80 caractères pour S1 et 80 caractères pour S2.

Les déclarations des variables peuvent être placées à plusieurs endroits mais pour l'instant, on se contentera de les placer au début d'une procédure ou d'une fonction avant le mot **Begin**.

La ligne :

```
S1:=123;
```

causera une erreur lors de la compilation.

Alors que la ligne :

```
S1:='Bonjour à tous';
```

est correcte.

L'instruction d'affectation d'une valeur à une variable peut être placée à plusieurs endroits mais pour l'instant, on se contentera de la placer entre le Begin et le End d'une procédure ou d'une fonction.

IV.4.2 Les types numériques

Les deux types numériques les plus utilisés sont : Integer et Real

Dans le premier on est limité à des nombres entiers de –2 147 483 648 à 2 147 483 647.

Dans le deuxième les nombres à virgule sont admis et la plage va de 3.6×10^{-4951} à 1.1×10^{4932} avec des valeurs positives ou négatives.

IV.4.3 Les types chaînes

Les chaînes courtes (255 caractères maximum) peuvent être du type ShortString. Depuis la version 2 de Delphi, le type `String` permet de manipuler des chaînes de toute taille.

La place allouée pour une variable de type `String` change automatiquement lorsque la chaîne change de taille.

IV.4.4 Le type boolean

Le type boolean ne permet que deux états. Une variable de type **Boolean** ne peut valoir que **True** ou **False**.

IV.4.5 Les types tableaux

Un tableau permet de retenir un ensemble de données de même type.

Il faut préciser après le mot Array le premier et le dernier indice du tableau.

```
Type TMonTableau = array[5..8] of integer;
```

Quand le compilateur rencontrera ces lignes, il n'allouera pas de mémoire. Remarquez le T permettant de se souvenir qu'il s'agit d'un type. C'est pratique, c'est une bonne habitude mais ce n'est pas obligatoire.

```
Var MonTableau : TMonTableau;
```

Quand le compilateur rencontrera cette ligne, il allouera la mémoire nécessaire. Les deux lignes de code précédentes pouvaient se réduire à une seule :

```
Var MonTableau : array[5..8] of integer;
```

Le tableau ainsi créé permet de retenir 4 nombres entiers. Voici des exemples de lignes de programme correctes :

```
MonTableau[5] := 758;
MonTableau[6] := MonTableau[5] * 2;
MonTableau[7] := -123;
MonTableau[8] :=MonTableau[5] - MonTableau[7];
Label1.Caption := IntToStr(MonTableau[8]);
```

La dernière ligne suppose que la fiche contient un label nommé Label1.

Exemples

```
Var NotesConcours : array[1..12] Of Real;
Var NotesPromo : array[1..100,1..12] Of Real;
```

La première variable tableau permettrait de retenir le résultat d'un candidat à un concours comportant 12 épreuves.

La deuxième variable tableau permettrait de retenir le résultat de 100 candidats à un concours comportant 12 épreuves.

Pour indiquer que le candidat numéro 25 a eu 13,5 à l'épreuve numéro 4, on pourra écrire :

```
NotesPromo[25,4] := 15.5;
```

Pour connaître le résultat du candidat numéro 25 à l'épreuve numéro 4, on pourra écrire :

```
Label1.Caption:='Le candidat a eu ' + IntToStr(NotesPromo[25,4]);
```

IV.4.6 Les types enregistrements

Ne pas confondre avec l'enregistrement d'un fichier sur le disque, il s'agit ici d'un regroupement de types plus simples.

```
Type TUnePersonne = Record
     Nom : string[25];
     Prenom : string[25];
     Age : ShortInt;
End;
```

Quand le compilateur rencontrera ces lignes, il n'allouera pas de mémoire.

```
Var UnePersonne:TUnePersonne;
```

Quand le compilateur rencontrera cette ligne, il allouera dans la mémoire une zone d'au moins 51 octets.

```
UnePersonne.Nom := Boualam;
UnePersonne.Prenom := samir;
UnePersonne.Age := 29;
```

Quand le compilateur rencontrera ces lignes, il remplira les zones mémoires précédemment allouées.

Pourquoi pas un tableau d'enregistrements ?

```
Type TTableauPersonnes = array[1..50] of TUnePersonne;
Var TableauPersonnes : TTableauPersonnes;
```

Pour indiquer par exemple que la troisième personne s'appelle Boualam Ali et qu'elle a 28 ans, on pourra taper :

```
TableauPersonnes[3].Nom := BOUALAM;
TableauPersonnes[3].Prenom := Ali;
TableauPersonnes[3].Age := 28;
```

Pour copier cette personne à la première place du tableau, on pourra au choix faire :

```
TableauPersonnes[1].Nom := TableauPersonnes[3].Nom;
TableauPersonnes[1].Prenom := TableauPersonnes[3].Prenom;
TableauPersonnes[1].Age := TableauPersonnes[3].Age;
```

Ou plus simplement :

```
TableauPersonnes[1] := TableauPersonnes[3];
```

IV.5 Tests

IV.5.1 If then

```
if expression qui peut être vraie ou fausse then
begin
// ce qu'il faut faire si vrai
end;
// suite faite dans tous les cas
```

Si, par exemple, vous devez gérer le stock d'un magasin, vous serez amené à contrôler le nombre d'articles afin de décider à quel moment il faudra passer la commande. La partie du programme pourrait contenir :

```
if NbrSacsCimentEnStock  < 50 then
begin
EnvoyerCommande('Sacs de ciment',120);
end;
```

Bien sûr ces lignes supposent que la variable NbrSacsCimentEnStock soit définie et que sa valeur reflète effectivement le stock. Elles supposent également que la procédure EnvoyerCommande existe et qu'elle demande un paramètre chaîne puis un paramètre entier.

Prenons un exemple qui utilise le hasard. Nous allons lancer un dé par exemple 1000 fois et nous allons comptabiliser le nombre de fois que l'on obtient le 6. Nous devrions en général obtenir aux environs de 167, mais rien n'interdit en théorie d'obtenir 1000 fois le 6 ou de ne jamais le voir !

Dans une nouvelle application, placez un label et un bouton sur la fiche. En cliquant sur le bouton, une boucle va simuler les 1000 lancers du dé, le label nous permettra de voir le résultat. Le test sera fait à l'intérieur de la boucle (entre le begin et le end correspondant au for).

Essayez de faire ce programme puis comparer avec ce qui suit.

```
unit Unit1;
interface
uses
Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
Dialogs, StdCtrls;
type
TForm1 = class(TForm)
LabelResultat: TLabel;
ButtonLancer: TButton;
procedure ButtonLancerClick(Sender: TObject);
private
{ Déclarations privées }
public
{ Déclarations publiques }
end;
var
Form1: TForm1;
implementation
{$R *.DFM}
procedure TForm1.ButtonLancerClick(Sender: TObject);
var i:integer;
var Nbr6:integer;
begin {début de la procédure}
randomize;
Nbr6:=0;
for i := 1 to 1000 do
begin {début de la boucle}
if random(6)+1 = 6 then
begin {début du test}
     inc(Nbr6);
end; {fin du test}
end; {fin de la boucle}
LabelResultat.Caption:='Nombre de 6 : ' + IntToStr(Nbr6);
end; {fin de la procédure}
end.
```

Il est nécessaire de mettre 0 dans la variable Nbr6 avant de commencer la boucle, en effet, à l'entrée dans une procédure ou une fonction, les variables déclarées localement ont une valeur quelconque.

Essayez de supprimer la ligne Nbr6 := 0 et vous obtiendrez des résultats certainement farfelus.

Essayez de ne pas déclarer la variable Nbr6 dans la procédure mais de la déclarer dans la partie classe (entre le mot private et le mot public). Ne mettez pas la ligne Nbr6 : = 0 ; .

Fig IV.1

Cliquer sur le bouton : « **Lancer 1000 fois** »

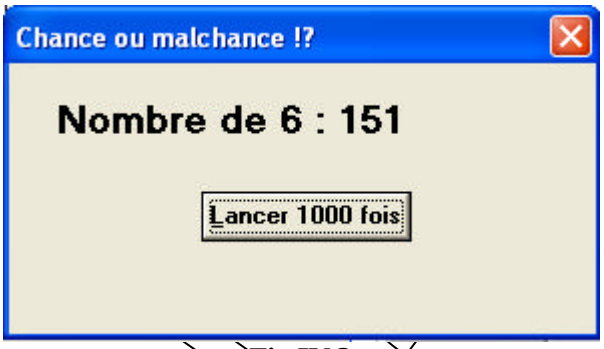


Fig IV.2

Vous obtenez

La première fois que vous cliquez sur le bouton, la variable Nbr6 commence à 0, vous obtenez un nombre de 6 raisonnable, la deuxième fois, il y a cumul avec les lancers de la première fois… Si vous arrêtez et lancez à nouveau le programme, Nbr6 recommence à 0.

Une variable déclarée dans la partie classe est mise automatiquement à 0 pour les variables numériques, à False pour les variables booléennes et à chaîne vide pour les variables chaînes.

IV.5.2 If then else

Il arrive assez souvent qu'une action doive être faite si le test est vrai et une autre si le test est faux.

```
if expression qui peut être vraie ou fausse then
begin
// ce qu'il faut faire si vrai
end
else
begin
// ce qu'il faut faire si faux
end;
// suite faite dans tous les cas
```

En supposant que Moy ait été définie comme real et qu'elle contienne le résultat d'un examen, une partie du programme pourrait être :

```
if Moy >= 10 then
begin
Label1.Caption := 'Examen réussi';
end
else
begin
Label1.Caption := 'Echec';
end;
```

Pour indiquer "supérieur ou égal" on utilise > suivi de =.
L'expression Moy >= 10 peut être vraie (True) ou fausse (False)

IV.6 Une boucle

Une boucle permet de répéter une partie d'un programme un certain nombre de fois. La syntaxe de la boucle est :

For nom de variable entière := première valeur **To** dernière valeur **Do**
Begin
ce qu'il faut faire plusieurs fois
End;

Pour mettre ceci en évidence sur un exemple, je vous propose de créer un deuxième bouton, (je suppose que vous l'appelez (propriété Name) "ButtonBoucle" et que vous mettez "&Essai de boucle" dans sa propriété Caption. Le « Et commercial » permet de souligner la lettre qu'il précède pour qu'elle sert de touche de raccourci clavier en combinaison avec la touche Alt (Alt + e). Faites un double clic sur le bouton et complétez la procédure obtenue comme suit :

```
Procedure TForm1.ButtonBoucleClick(Sender: TObject);
Var i,S : Integer;
Var  Resultat : String;
begin
s:=0;
for i:=1 to 10 do
begin
S :=S+i
End;
Resultat :=IntToStr (S);
ShowMessage ('La somme des dix premières valeurs est ' + Resultat);
end;
```

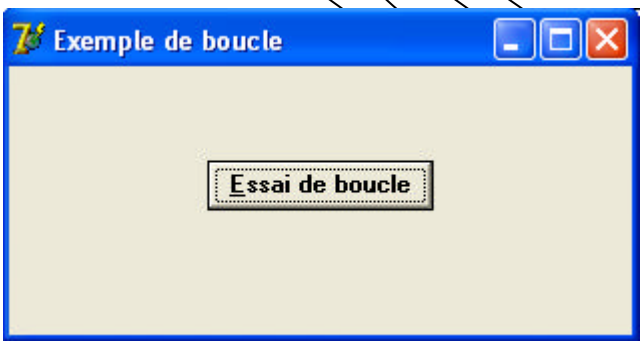


Fig IV.3

Fig IV.4

Cliquer su le bouton Essai de boucle ou vous obtiendrez :
Appuyer au clavier sur Alt E

En exécutant le programme et en cliquant sur le bouton "Essai de boucle", vous obtenez un message qui affiche la somme des dix premières valeurs (de 1 à 10).
La fonction IntToStr permet de convertir le nombre placé dans la variable S en une chaîne. Lorsque la variable S contient par exemple le nombre 3, la variable Resultat reçoit la chaîne '3'. Les deux chaînes "La somme des dix premières valeurs est '' et Resultat sont mises l'une au bout de l'autre à l'aide du signe +. On dit que les chaînes ont été concaténées.

IV.7 Débogage

Placez le curseur sur la ligne commençant par For et tapez sur F5 (on peut aussi faire "Ajouter point d'arrêt" à l'aide du menu Exécuter). Exécutez le programme et cliquez sur le bouton "Essai de boucle". Le programme s'arrête sur la ligne contenant le point d'arrêt. Utilisez F8 ou F7 pour avancer en "Pas à Pas".

Pour observer le contenu d'une variable, il suffit de placer (sans cliquer) le pointeur de la souris sur le nom de la variable et d'attendre une seconde. Avec toutes les versions, placez le curseur sur la variable et faites CTRL + F7 (ou utilisez "Evaluer/modifier" dans le menu Exécuter). Pour continuer l'exécution du programme à la vitesse normale, vous pouvez taper sur F9.
Remarque :

Si vous obtenez le message "La variable ... est inaccessible ici du fait de l'optimisation" vous pouvez dans "Projet" "Option" "Compilateur" supprimer la coche dans "Optimisation". L'exécutable obtenu sera un peu moins performant, mais vous pourrez voir toutes les variables dès la prochaine compilation.

IV.8 Comment supprimer un composant

Pour supprimer par exemple Button1, vous allez commencer par vider toutes les méthodes liées aux événements de Button1. Dans notre cas, seul l'événement "OnClick" déclenche une procédure. Supprimez les lignes situées entre la ligne "Procedure ... " et la ligne "Begin" ainsi que les lignes entre Begin et End;
Votre procédure doit devenir :

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

end;
```

Tapez sur F12 pour voir la fiche et supprimez le bouton "Button1"
A la prochaine sauvegarde ou à la prochaine compilation, les lignes inutiles seront automatiquement supprimées.

V QUELQUES CONSEILS

A la fin de cette partie, vous serez capable de:

- ✍ utiliser Delphi avec un peu plus d'assurance,
- ✍ imprimer depuis un programme Delphi,
- ✍ construire une application constituée de plusieurs fiches.

V.1 Questions fondamentales

Quand on aborde Delphi on est souvent arrêté par des problèmes de moindre importance, mais qui demandent du temps pour les résoudre. Nous allons en étudier quelques uns dont la résolution n'est pas évidente, étant entendu que d'autres seront étudiés au fur et à mesure dans les pages suivantes.

Comment trouver de l'aide en cas de problèmes ?

Le système d'aide de Delphi est extrêmement bien conçu, bien qu'il soit (ou parce qu'il est) volumineux. Toutefois on peut s'y perdre et avoir du mal à trouver les renseignements cherchés. Pour mieux comprendre son utilisation nous allons voir quelques conseils.

Il est important, dans un environnement tel que Delphi, d'utiliser correctement et efficacement l'aide en ligne. Les problèmes rencontrés peuvent être divers:

- ? on est complètement perdu.
- ? on ne sait pas comment exprimer ce qu'on cherche.
- ? on tourne en rond.
- ? etc. ...

Passons en revue quelques exemple d'appels à l'aide:

On aimerait connaître la syntaxe d'une instruction Pascal

- ? placer le curseur sur un mot, par exemple if
- ? appuyer sur <F1> et attendre le résultat.

On aimerait savoir s'il existe une procédure qui donne la date du jour

- ? lancer l'aide de Delphi et choisir "Référence de la bibliothèque Delphi" .
- ? rechercher, par exemple, le mot "date" ce qui devrait résoudre le problème.

On aimerait savoir comment ajouter des ligne dans un composant mémo

- ? pour obtenir de l'aide concernant un objet de Delphi, placez-le d'abord sur une fiche, sélectionnez-le, puis appuyer sur <F1>.

généralement l'aide donne une explication succincte sur l'utilisation de cet objet, mais donne également la liste des propriétés, méthodes et événements associés à cet objet. de fil en aiguille on trouve généralement les renseignements cherchés.

Comment terminer une application ?

Il faut savoir que Delphi gère un objet "Application" qui représente justement l'application en cours. cet objet possède entre autres une méthode "Terminate" qui termine l'exécution de l'application.

Comment créer et utiliser d'autres fiches que la fiche principale ?

- ? créez la fiche principale "fiche1" (avec par exemple un bouton pour appeler une seconde fiche).
- ? créez la seconde fiche "fiche2".
- ? dans la partie **Implementation** de la fiche1 placez la clause: `uses unit2;` où unit2 est l'unité associée à la fiche2

pour ouvrir la fiche2 depuis la fiche1, tapez:

```
Form2.show;
```

pour fermer la fche2, tapez:

```
Form2.close;
```

V.2 Comment imprimer ?

Il y a plusieurs manière d'imprimer avec Delphi

V.2.1 Impression du contenu d'une fiche

Si on place un bouton **Print** dans une fiche il est possible d'imprimer le contenu de la fiche de la manière suivante:

```
procedure Tform1.PrintClick (Sender: Tobject);
begin
Print;
end;
```

V.2.2 Utilisation du printdialog

Voici un exemple d'utilisation du dialogue standard d'impression:

```
procedure Tform1.PrintClick (Sender: Tobject);
var i : integer;
begin
if PD1.execute then begin
Print;
if (PD1.Copies > 1) then
for i := 1 to PD1.Copies - 1 do
Print;
end;
end;
```

V.2.3 Impression par ma méthode "Brute"

Lorsqu'il s'agit d'imprimer du texte sans trop de raffinements, avec une seule fonte, on peut utiliser une méthode simple, dont voici un exemple:

```
procedure Tform1.ImprimerClick(Sender: TObject);
var PrintFile: system.text;
i : integer;
begin
if PD1.execute then begin
assignPrn (PrintFile);
Rewrite (PrintFile);
Printer.Canvas.Font := Liste.Font;
for i := 0 to Liste.Items.count - 1 do
Writeln (PrintFile, Liste.Items[i]);
System.CloseFile (PrintFile);
end;
end;
```

Il faut cependant ajouter l'unité Printers dans la clause "uses".

V.3 Comment introduire un délai d'attente ?

Dans un environnement comme Windows, et de surcroît Windows 9x ou Windows NT, il n'est pas judicieux d'introduire une temporisation comme en mode "DOS" avec une instruction du genre **delay (nnn)**. En effet, il faut éviter de monopoliser le processeur dans un environnement multitâches. D'autres moyens permettent d'obtenir le résultat désiré. Dans Delphi 1.0 on peut utiliser un **timer** avec la propriété **interval** ayant pour valeur le délai désiré. Une fois l'événement **OnTimer** déclenché, il suffit de désactiver le **timer**.

Delphi permet de faire appel à la fonction sleep. Pour obtenir un délai de 3 secondes on écrira: `Sleep (3000);`

V.4 Comment laisser la main à Windows de temps à autre ?

Il est parfois important de laisser Windows répondre à des messages provenant de l'application en cours ou d'autres applications concurrentes. C'est le cas, par exemple, lors de longues boucles de calculs ou lorsque l'on veut laisser réagir l'application en cours. Dans ces cas on peut insérer l'instruction: **Application.ProcessMessages;**

Ceci permet, par exemple dans une boucle sans fin, de pouvoir arrêter normalement l'application. En effet, la demande d'arrêt de l'application se traduit par l'envoi d'un message. Pour qu'il soit pris en compte, il faut laisser Windows prendre la main.

V.5 Application comportant plusieurs fiches

Généralement, lorsque l'on maîtrise l'appel et l'affichage des fiches, on a tendance à créer des applications comportant beaucoup de fiches. Ce n'est pas toujours une bonne idée. Pour s'en convaincre, il suffit de regarder autour de soi. Beaucoup d'applications (Word, Excel, programme de dessin divers...) sont de type MDI (avec une fiche principale et une ou plusieurs fenêtres filles, toutes du même type). Il est assez rare de voir des applications qui ouvrent de multiples fenêtres. *Mais alors, comment faire ?*

Plusieurs techniques sont possibles, permettant de réduire le nombre de fiches. On peut:

- utiliser une seule fiche et changer sont contenu selon les besoins;
- faire abondamment appel aux boîtes de dialogue depuis une fiche principale.
- utiliser les fiches multipages à onglets;
- changer de métier.

Si toutefois, on est contraint à l'utilisation de plusieurs fiches, il faut connaître quelques principes.

Supposons que depuis Form1 (dans unit1) on veuille ouvrir Form2 (dans unit2); il convient de placer la clause *uses unit2 ;* dans la partie **implementation** de l'unité unit1. En cas d'oubli, Delphi propose de placer cette clause à votre place par le message suivant:

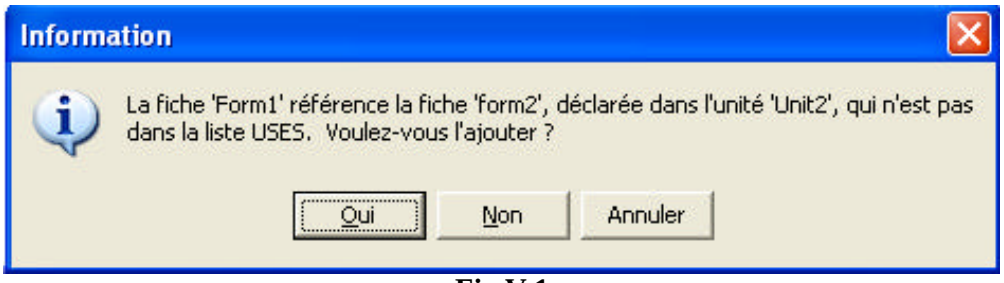


Fig V.1

Après avoir accepté voici la clause **uses** que Delphi ajoute à l'unité de la Form1:

```
unit Unit1;
interface
…
implementation
uses Unit2;
```

Voyons maintenant ce que signifie exactement afficher une fenêtre. Quand on commence une nouvelle application (avec une fiche vide) le projet se présente sous la forme suivante:

```
program Project1;
uses Forms,
Unit1 in 'Unit1.pas' {Form1};
{$R *.RES}
begin
Application.Initialize;
Application.CreateForm(TForm1, Form1);
Application.Run;
end.
```

On peut alors ajouter une seconde fiche au projet à l'aide de l'option Nouvelle fiche du menu Fichier. Le projet devient alors:

```
program Project1;
uses Forms,
Unit1 in 'Unit1.pas' {Form1},
Unit2 in 'Unit2.pas' {Form2};
{$R *.RES}
begin
Application.Initialize;
Application.CreateForm(TForm1, Form1);
Application.CreateForm(TForm2, Form2);
Application.Run;
end.
```

Par défaut, Delphi crée automatiquement les deux fiches. Mais créer ne signifie pas montrer. Une seule fiche (la première) est affichée lors du lancement du programme. La seconde est crée, mais pas affichée. On peut l'afficher et la cacher à l'aide des méthodes Show, Hide et Close. Show montre la fiche. Hide cache la fiche (met la propriété visible à false). Close ferme la fiche.

VI PREMIERS PAS

A la fin de cette partie, vous serez capable de:

- accéder au code source d'une application,
- construire une application très simple.

VI.1 Adjonction d'objets sur une fiche

Nous allons suivre ces bons conseils et commencer à placer des objets sur notre fiche (que nous appellerons Exemple).

VI.2 1ère étape: mise en place d'un bouton

Le moyen le plus simple pour placer un objet (ou contrôle, ou encore composant) sur une fiche est de le repérer dans la liste proposée par Delphi (partie supérieure de l'écran) et de double-cliquer sur son icône. Un objet de ce type est alors inséré au centre de la fiche. Un autre moyen pour placer un objet est le suivant: après avoir cliqué sur son icône, il faut cliquer sur la fiche et déplacer la souris; l'objet prend alors les dimensions imposées.

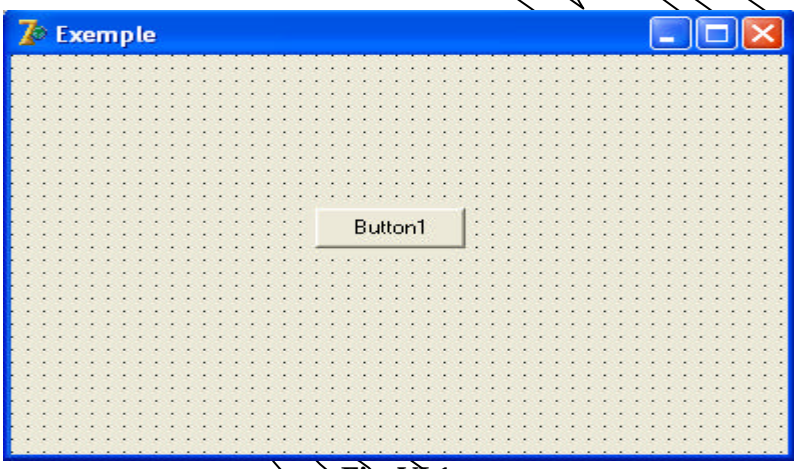


Fig. VI.1

Voici comment le code correspondant à la fiche est automatiquement modifié par Delphi

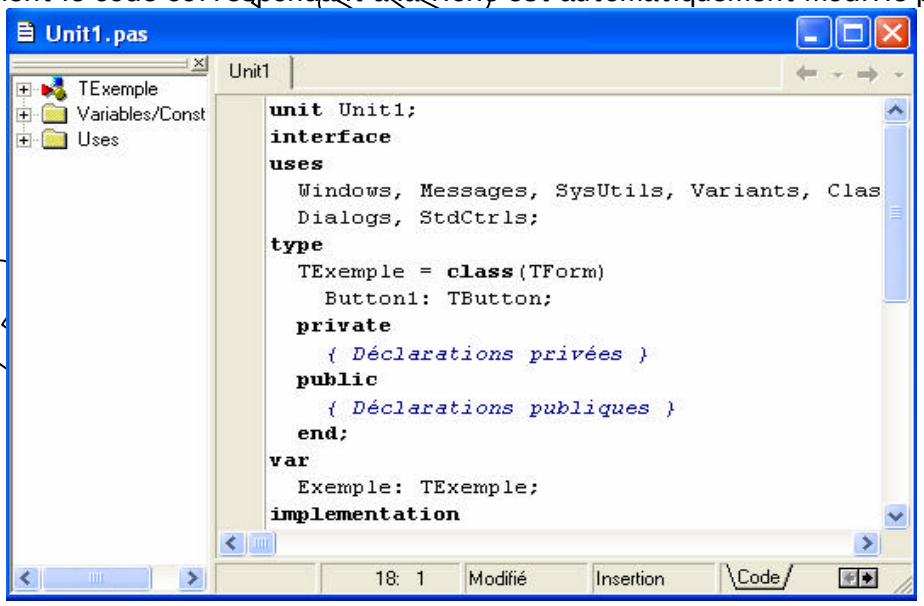


Fig. VI.2

On peut constater que:

Lorsque l'on change le nom de la fiche de Form1 en Exemple (dans les propriétés de l'inspecteur d'objets), Delphi transpose ce changement dans le code source. Une fiche possède aussi une propriété appelée Caption (titre ou étiquette). Cette propriété prend par défaut la valeur du nom de l'objet. Toutefois il est possible de modifier ce titre. C'est Caption et non Name qui apparaît comme titre de la fiche.

Delphi a placé un nouveau champ dans la classe Texemple: Button1 qui est de type (prédéfini) Tbutton. Il correspond à notre nouveau bouton.

VI.3 Sauvegarde du projet

Delphi appelle Projet une application Windows qu'il gère. Avant de sauvegarder tous les fichiers d'un projet sur disque il est conseillé de créer un répertoire permettant de les accueillir. La sauvegarde s'obtient à l'aide de l'option Fichier / Enregistrer le projet sous... Si aucune sauvegarde n'a encore été effectuée, Delphi propose d'abord d'enregistrer la **unit** sous le nom UNIT1.PAS. Il vaut mieux, comme indiqué plus haut, changer le nom que Delphi propose par défaut; par exemple, indiquons EXU1.pas. Puis Delphi demande le nom du projet et propose PROJET1.DPR. Là nous spécifions, par exemple, EXE1.DPR. Dans la figure qui suit, on voit comment Delphi a répercuté ces choix dans le code source. Afin de ne pas avoir de doutes sur les différents noms de fichiers, il convient de bien comprendre comment Delphi les gère.

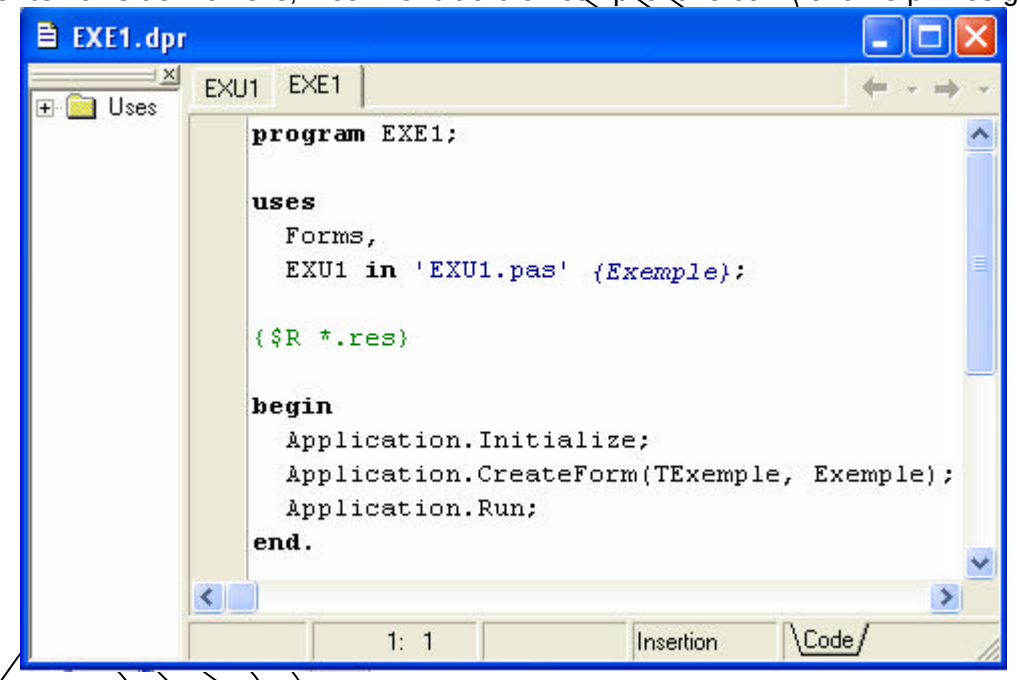


Fig. VI.3

Jusqu'à présent nous avons vu une `unit`, mais pas de programme principal. Où se cache-t-il ? On peut le voir à l'aide de la commande Voir le Source du projet du menu Projet. Sur la figure VI.3 on voit une fenêtre avec deux onglets: un pour le programme principal (projet) et un pour la `unit`. Si le projet fait appel à plus d'`unit`, il y aurait plus d'onglets.

On constate que EX1 est le nom du projet (EX1.DPR), mais aussi le nom du programme. De plus, dans la clause uses on voit la ligne:

```
exu1 in 'exu1.pas' {Exemple};
```

où:

- exu1 est le nom interne de la unit.
- exu1.pas est le nom du fichier dans lequel se trouve la unit (pas forcément le même)
- Exemple est le nom de la fiche associée à cette unit.

On peut être déçu par la simplicité du programme principal. En fait, il ne sera jamais beaucoup plus imposant que cet exemple.

VI.4 Pourquoi tant de fichiers ?

Un vrai projet complexe écrit en Delphi peut compter, du moins dans la phase de développement, plusieurs dizaines de fichiers. On comprend donc pourquoi il est important de bien savoir à quoi ils correspondent et où il convient de les sauvegarder. Dans la plupart des exemples simples de ce cours, les fichiers générés sont les suivants:

Type	Création	Nécessaire pour compiler	Description
DPR	Développement	Oui	Projet (programme principal)
PAS	Développement	Oui	Source des units
DCU	Compilation	Non	Units compilées
EXE	Compilation	Non	Code exécutable de l'application
DFM	Développement	Oui	Contient la description des propriétés d'une fiche et des contrôles qu'elle contient
RES	Développement	Oui	Fichier binaire associé au projet. Contient l'icône par défaut. Plusieurs fichiers RES peuvent être ajoutés au projet
~PA	développement	non	Copie de sauvegarde du PAS correspondant
~DP	développement	non	Copie de sauvegarde du DPR correspondant
~DF	développement	non	Copie de sauvegarde du DFM correspondant

Fig. VI.4

D'autres type de fichiers peuvent être créés, utilisés ou gérés par Delphi:

Type	Création	Nécessaire pour compiler	Description
BMP, ICO	développement	non (mais peuvent être nécessaires à l'exécution)	Bitmaps et icons divers
DOF	développement	oui si des options sont spécifiées	Contient les valeurs courantes des options du projet
DSK	développement	non	Contient des informations concernant l'environnement de développement (position des fenêtres, éditeur, etc...)
DSM	compilation	non	Contient les informations associées au Browser

fig. VI.5

VI.5 2ème étape: utilisation des composants

Revenons à notre premier programme et ajoutons-lui quelques fonctionnalités, par exemple:

- le bouton déjà placé va servir à arrêter le programme.
- un nouveau bouton "Dessin" appellera une seconde fiche sur laquelle on placera une image de type BMP. Ce bouton sera rendu invisible.
- un nouveau champ de type Edit dont le contenu pourra rendre le bouton "Dessin" visible.

Voici comment se présente la nouvelle fiche principale:

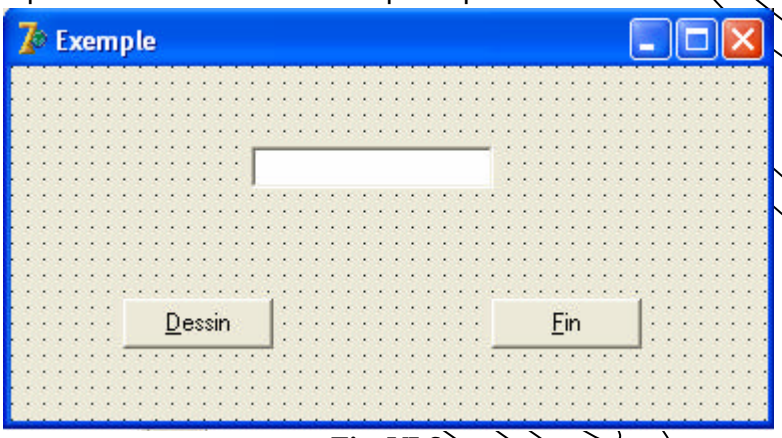


Fig. VI.6

Nous allons faire réagir notre programme à 3 événements:

- Lorsque le contenu du champ Edit est "démarrer" le bouton "Dessin" doit devenir visible.
- lorsque l'on clique sur le bouton "Fin" le programme doit se terminer.
- Lorsque l'on clique sur le bouton "Dessin" le programme doit ouvrir une seconde fiche.

A ce point il est très important, voir même vital, de comprendre le principe du raisonnement qui mène à la construction d'un programme. Il faut toujours se poser la question suivante: *Je veux qu'il se passe quoi, lorsque quel événement se produit sur quel objet ?*

Répondre à ces trois questions est obligatoire. Si un doute subsiste concernant -ne serait-ce qu'une de ces trois questions-, alors le principe même de la réalisation d'applications Windows avec Delphi est fortement compromis.

Pour chacun de nos trois objets nous allons répondre à ces trois questions:

- je veux qu'il se passe "rendre le bouton Dessin visible" quand l'événement "changement du contenu et contenu vaut ' démarrer ' " se produit sur l'objet "Edit".
- je veux qu'il se passe "arrêt du programme" quand l'événement "cliquer" se produit sur l'objet "bouton Fin".
- je veux qu'il se passe "ouvrir une seconde fiche" quand l'événement "cliquer" se produit sur l'objet "bouton Dessin".

En effet, il y a généralement une réponse lorsqu'un événement survient sur un objet donné. Voyons maintenant comment implémenter ces réponses.

Nous décrivons ici une des diverses méthodes permettant de placer des instructions de programme (du code) "derrière" un objet en réponse à un événement. Commencez par cliquer sur le champ Edit pour le sélectionner.

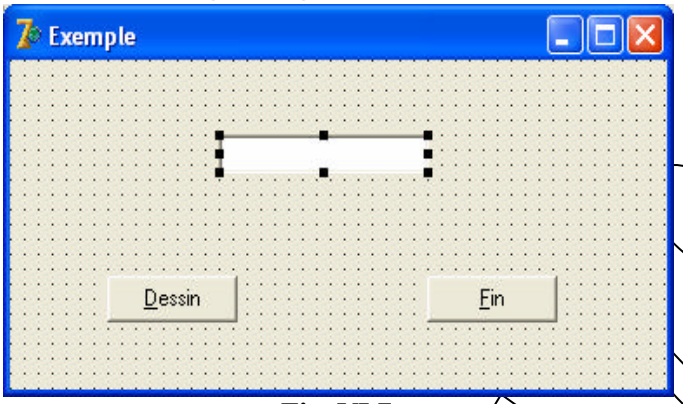


Fig. VI.7

Dans l'inspecteur d'objets, choisissez l'onglet "Evénements" et double-cliquez dans la case blanche en face de l'événement "OnChange". Cela signifie que l'on veut écrire du code qui sera exécuté chaque fois que le contenu de cet objet change:

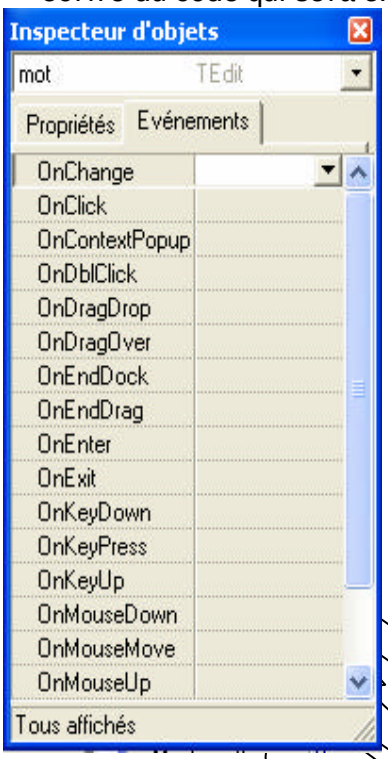


Fig. VI.8

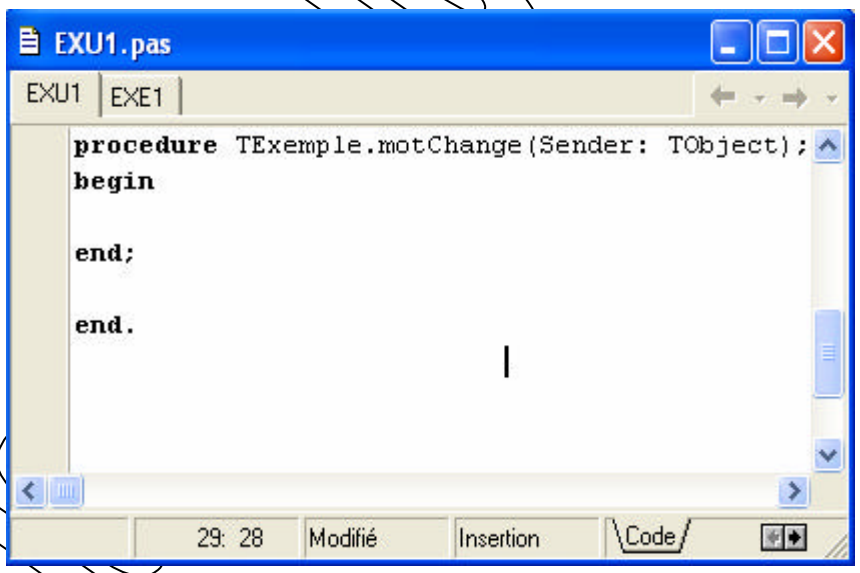


Fig. VI.9

Voici ce qui apparaît alors:

Nous allons placer le code suivant à l'endroit où Delphi a positionné le curseur. A chaque fois que le contenu du champ Edit change, la procédure Texemple.motChange est appelée. Quand le contenu correspond à ce que nous avons pré-défini pour rendre le bouton Dessin visible, la propriété "visible" de l'objet "Dessin" est mise à "vrai". Il est donc possible, et même courant, de modifier des propriétés d'objets en réponse à des événements.

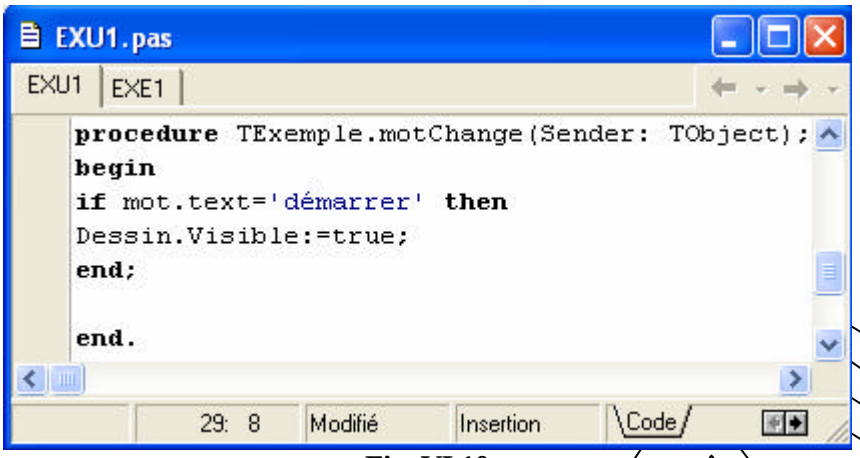


Fig. VI.10

Procédez de la même manière avec le bouton "Fin". Voici le code correspondant:

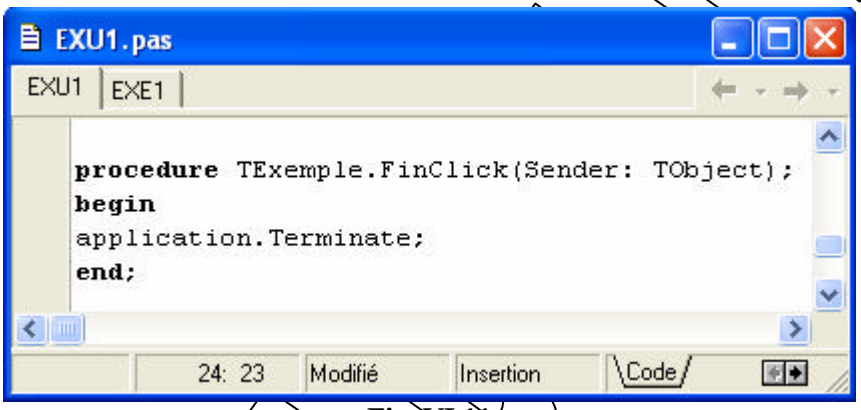


Fig. VI.11

Avant de passer à la suite il faut créer une nouvelle fiche (menu Fichier/Nouvelle fiche) dont le nom sera "Expo". Dans cette fiche placez un contrôle de type image dont on garde le nom ("Image1") et un bouton de type BitBtn à partir de la palette Supplément et mettez sa propriété Kind à BkClose pour que la fiche se ferme quand on clique dessus et cela sans écrire la moindre ligne de code. Ce type de bouton à la faculté de contenir un texte (Caption) et une image. Voici cette fiche en construction:

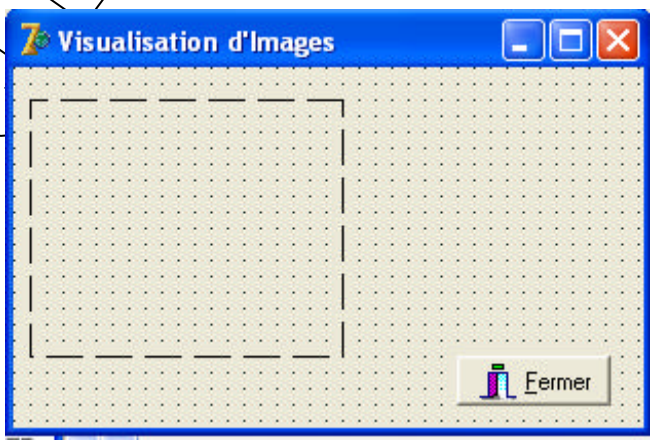


Fig. VI.12

Revenez sur la première fiche en placez le code suivant dans l'événement "OnClick" du bouton "Dessin". Ce code se résume à Expo.show où Expo est le nom de la seconde fiche et show est la méthode permettant de charger et d'afficher une fiche.

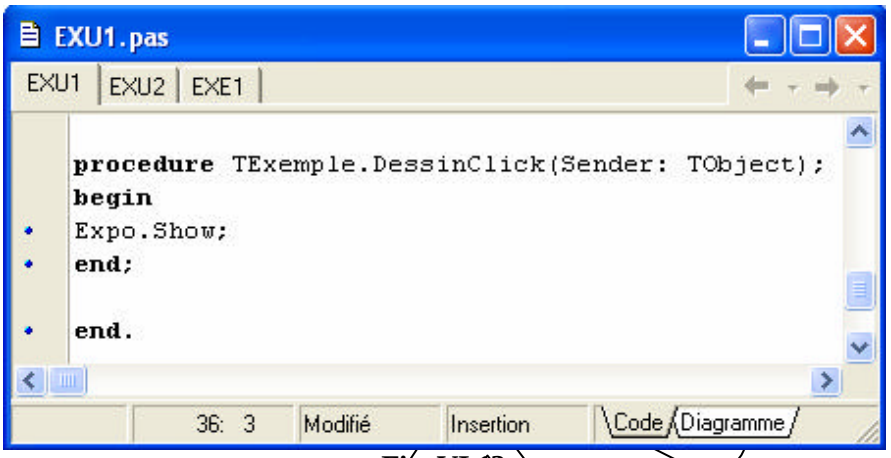


Fig. VI.13

VI.6 Quelques précisions:

lors de la sauvegarde du projet, Delphi demande quel nom il doit donner au fichier contenant l'unité rattachée à la fiche "Expo". Pour des raisons de cohérence (la première fiche étant dans "exu1.pas") nous choisissons "exu2.pas";

lors de la compilation du projet (menu Projet/Compiler) Delphi affiche un message d'erreur. En effet, depuis la première fiche on fait référence à la deuxième fiche, mais cette dernière (donc son unité) est inconnue de la première fiche (donc unité). Dans une grande bonté d'âme Delphi propose d'ajouter la ligne uses exu2; dans l'unité de la première fiche, ce qui permet de supprimer l'erreur.

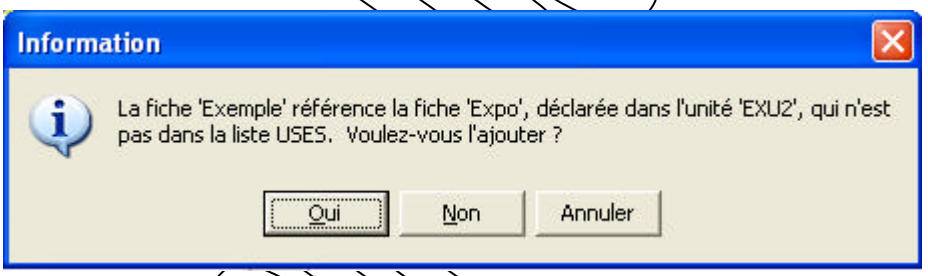


Fig. VI.14

A ce stade vous pouvez déjà exécuter le programme, appeler la seconde fiche, la fermer, etc. Toutefois, la seconde fiche ne contient pas d'image visible. Nous allons voir 3 possibilités de placer une image dans cette fiche.

Première solution: placer une image dans la propriété "Picture" de l'objet "Image1" en cliquant sur les 3 points à côté de "(vide)" puis cliquer sur le bouton Charger et sélectionnez à partir d'un dossier une image.

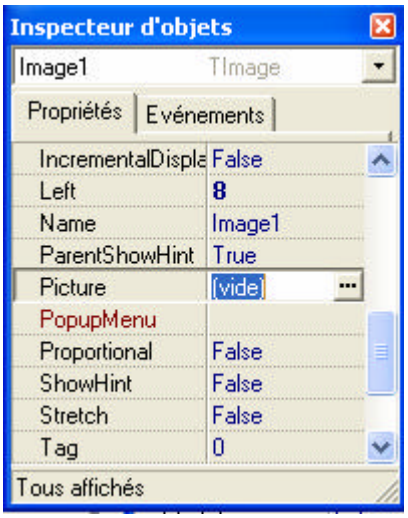


Fig. VI.15

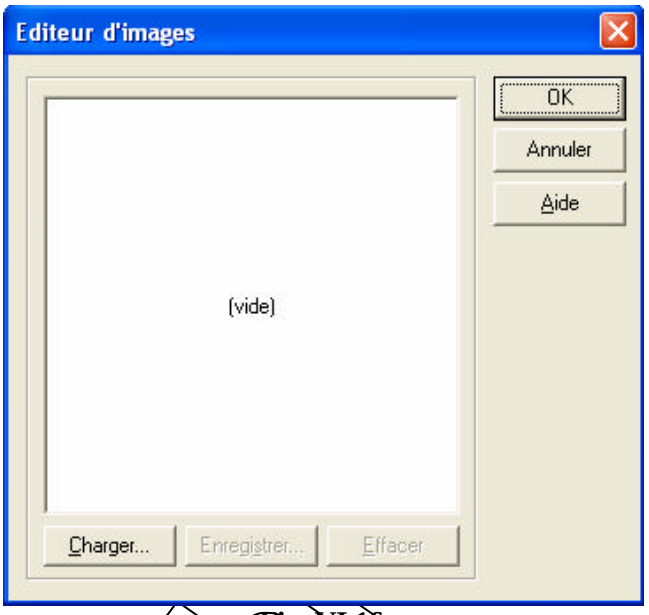


Fig. VI.16

Voici comment se présente la fiche après avoir choisi une image

Fig. VI.17

Deuxième solution: charger une image depuis le code lorsque la fiche "Expo" est activée.

Par exemple, placez le code suivant dans l'événement "OnActivate" de la fiche "Expo": En effet, la propriété "picture" possède une méthode "loadfromfile" permettant de charger une image. Taper entre parenthèse et entre cote le nom du fichier, du lecteur et du chemin.

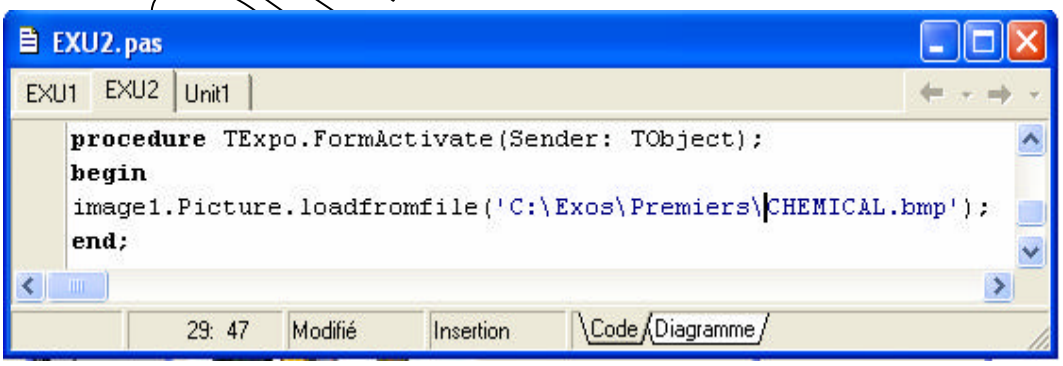


Fig. VI.18

Troisième solution: proposer à l'utilisateur de choisir lui-même l'image qu'il veut voir. Pour cela, on utilise un des composants de dialogue standard proposés par Windows.

Supprimez le code figurant sur la figure VI.18. Ajouter un bouton dont le nom est "Charger" sur la seconde fiche. Ajouter sur cette même fiche un OpenDialog à partir de la palette de composants que l'on appellera "Ouvrir". Voici comment se présente la fiche

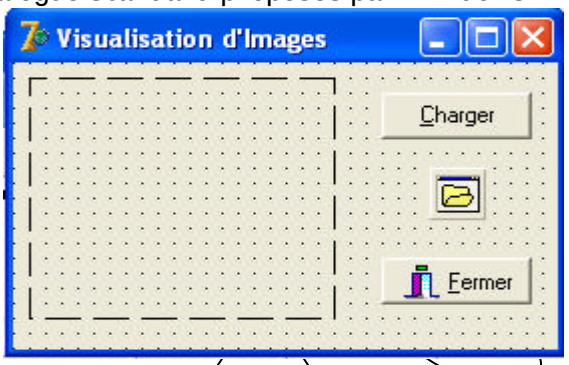


Fig. VI.19

placer le code suivant dans l'événement "OnClick" du bouton "Charger":

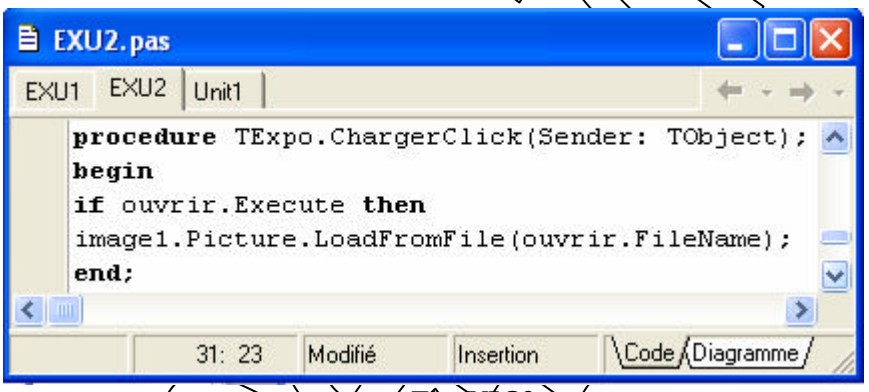


Fig. VI.20

Lors de l'exécution du programme, une boite de dialogue apparaît quand on clique sur le bouton "Charger", sélectionner le lecteur, le dossier puis le fichier de l'image ensuite cliquer sur le bouton Ouvrir. Dans le code que nous avons écrit (figure VI.20), "Ouvrir" est l'objet "dialogue standard d'ouverture de fichier", "**execute**" est une des méthodes de ce dialogue; elle permet de l'exécuter ou de le montrer. "**Filename**" est une propriété de ce dialogue contenant le nom du fichier choisi. Il est fortement conseillé de tester le résultat de la méthode "**execute**" avant d'utiliser "**Ouvrir.filename**", car si l'utilisateur abandonne le dialogue ("cancel") la propriété "**filename**" n'est pas utilisable. Il est bien entendu possible de spécifier le répertoire de départ, le ou les extensions de fichiers acceptées par défaut, etc. Ces différentes améliorations sont laissées comme exercice.

VI.7 Conclusion

Le petit exemple construit ci-dessus a pour but de:

- montrer quels sont les principes de base à utiliser pour construire une application avec Delphi.
- montrer qu'avec quelques lignes de code il est possible de construire une application fonctionnelle, sûre et d'un aspect propre
- donner envie de réfléchir sur de nouveaux problèmes à résoudre à l'aide de Delphi

Rappelons encore que :

Delphi permet d'obtenir des programmes dont les performances sont au moins aussi bonnes que celles obtenues avec le meilleur compilateur C, car derrière Delphi il y a du C pour ce qui est de la génération du code et de l'optimisation.

VII EXEMPLE D'APPLICATION

VII.1 Comment préparer un projet ?

L'exemple traité dans ce chapitre vous permettra de suivre pas à pas le processus standard de création d'un projet à commencer par la fiche principale, les menus jusqu'à appels de fiches et de procédures.

VII.2 Résumé du projet

Nous développerons une petite application qui permet de réaliser les opérations de base: Addition, produit, Soustraction et Division. Les valeurs sont introduites dans deux zones de textes différentes, puis on choisit à l'aide d'une commande de menu ou un bouton de barre d'outils l'opération à réaliser. On verra également comment afficher des textes d'aide pour guider l'utilisateur. Ces textes d'aide vont s'afficher comme info-bulle et dans la barre d'état également (à la manière Office). Ensuite, on verra comment réaliser l'appel des fiches (A propos…). Et enfin, on s'intéressera aux *Exceptions*: Comprendre la signification des exceptions, les appliquer dans des programmes et développer ses propres traitement d'exceptions.

VII.3 Créer la fiche principale

A son lancement, Delphi crée un projet contenant une fiche vide qu'il appelle Form1 . Vous pouvez le prendre pour votre premier projet ou en créer un nouveau par la Commande **Fichier / Nouvelle application**. Dans une application, la fiche principale comporte généralement une barre de menus et une barre d'outils qui permettent d'assurer les appels de fiches ou de procédures.

Commençant par donner un nom plus original à la fiche. Pour cela, sélectionnez la propriété Caption de l'inspecteur d'objet, cette propriété définit le titre de la fiche. Saisissez alors comme titre : **Mon premier programme delphi**. Remarquez que le titre de la fenêtre change au fur et à mesure de la saisie.

Fig 6.1

Pour qu'à l'exécution, la fiche principale de l'application occupe le plein écran, choisissez la valeur WsMaximized pour la propriété WindowState de la fiche.

VII.4 Ajouter des composants

Maintenant vous allez commencer à *habiller* votre fiche pour en faire un masque de saisie. Dans l'onglet **Standard**, cliquer sur le composant **Label** et déposer le sur la fiche. Il s'appelle par défaut Label1. A l'aide de la propriété **Caption** du label, donner un nom significatif. Dans notre cas c'est : &Valeur**.** On fait précéder le nom par **&** pour le rendre accessible avec le clavier (Alt+V).

Rajouter deux autres labels que vous placeriez verticalement avec le premier label. Nommez les respectivement : **&**Argument et **&**Résultat. Placez à coté de ces trois labels et verticalement aussi, trois composants **Edit** . Changer les contenus de la propriété **Text** des composants Edit pour qu'ils affichent zéro quand on exécute le programme. Pour qu'ils aient les même dimensions, utiliser les propriétés Height (Hauteur) et Width (largeur). Afin de les aligner verticalement, utiliser la même valeur pour la propriété Left**.** Ou bien utiliser la **palette d'alignement** du menu **Voir** pour bien ligner les composants (horizontalement et verticalement)

Placer sur la fiche deux composants **UpDown** que vous trouverez dans la palette Win32. changer leurs propriétés **Associate** afin de les lier à Edit1 et Edit2. Les propriétés Min et Max permettent de fixer le plafond et le plancher qu'on peut atteindre à l'aide des UpDown. La propriété **Increment** permet de spécifier le pas d'incrémentation ou de décrémentation du UpDown.

Remarque

Pour que les raccourcis claviers fonctionnent entre les labels et les Edit respectifs qui leurs sont associés, relier chaque label à un Edit en utilisant la propriété FocusControl du composant Label.

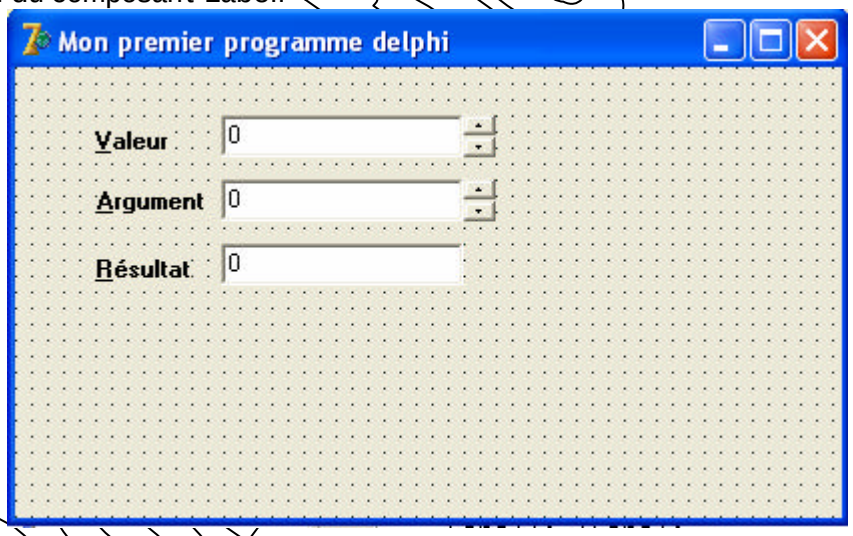


Fig 6.2

Astuces

Pour placer plusieurs copies du même composant, combiner la touche Shift au clic de souris au moment de la sélection du composant. Il suffit ensuite de déposer sur la fiche par différent clics le nombre de copies souhaitées. Pour libérer la souris, cliquer sur le premier composant (le plus à gauche) sous forme d'une flèche.

Pour sélectionner plusieurs composants de la fiche en même temps, maintenez la touche Shift enfoncée et cliquer sur les différents composants.

VII.5 Création d'un menu

Dans l'onglet **Standard**, cliquer sur le bouton **MainMenu** puis déposer le sur la fiche elle même. Double-cliquer sur ce composant et commencer à construire votre menu en définissant les libellés des commandes avec la propriété **Caption**. Commencer d'abord par créer trois en tête de menus, dont les propriétés Caption respectives sont : **&**Opérations, &Edition et **&**Aide.

Dans le premier menu (Opérations), définissez les entrées suivantes :

Caption	ShortCut
&Addition	Ctrl+A
&Produit	Ctrl+P
&Soustraction	Ctrl+S
&Division	Ctrl+D
-	
&Fermer	Ctrl+F4

Dans le deuxième menu (Edition) on va mettre les commandes : Couper (Ctrl+x), Copier (Ctrl+C) et Coller (Ctrl+V).

Dans le troisième menu (Aide) qui va servir à l'appel d'une autre fiche, on va placer une seule commande : **A propos…**

La propriété ShortCut permet de créer des raccourcis claviers aux commandes des menus.

Le signe moins (-) sert à créer un trait séparateur entre un groupe de commandes du menu.

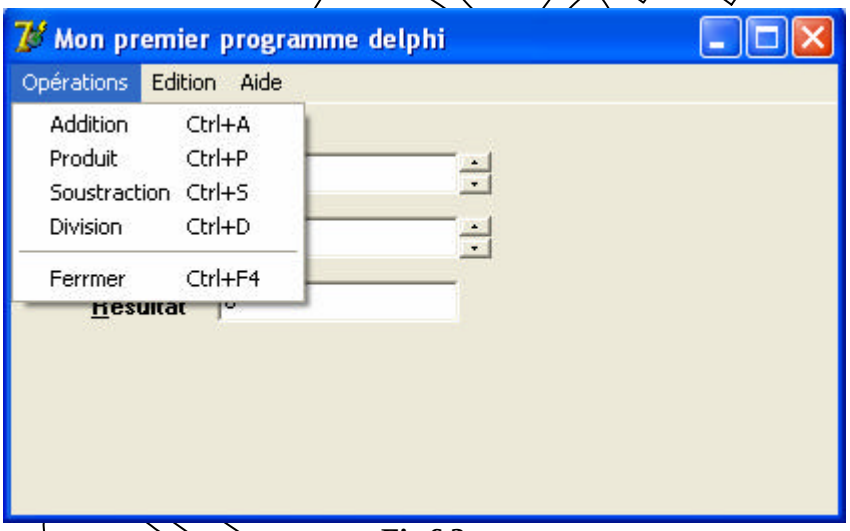


Fig 6.3

Double cliquer sur la commande **Fermer** (à partir du concepteur des menus). Vous êtes alors dans l'unité correspondante de la fiche, à l'intérieur d'un corps de procédure générée automatiquement par le moteur de Delphi. Taper la commande Close ;

```
Procedure TForm1.Button1Click (Sender: TObject);
begin
Close
end;
```

la commande **Close** permet de fermer le programme. Revenez sur la fiche en appuyant au clavier sur F12.

Double cliquer sur **Addition** et rajouter la ligne de code suivante :

```
edit3.text:= floattostr(strtofloat(edit1.text) + strtofloat(edit2.text));
```

Double cliquer sur **Produit** et rajouter la ligne de code suivante :

```
edit3.text:= floattostr(strtofloat(edit1.text) * strtofloat(edit2.text));
```

Double cliquer sur **Soustraction** et rajouter la ligne de code suivante :

```
edit3.text:= floattostr(strtofloat(edit1.text) - strtofloat(edit2.text));
```

Double cliquer sur **Division** et rajouter la ligne de code suivante :

```
edit3.text:= floattostr(strtofloat(edit1.text) / strtofloat(edit2.text));
```

Remarque: On est obligé de convertir les valeurs saisies dans les différents composants Edit afin de pouvoir réaliser des opérations de calculs avec ces données parce qu'elles sont, par défaut, de type texte (Caractère). Si cette conversion ne s'est pas faite, l'opération d'addition se solderait par un résultat de concaténation de deux chaînes de caractères (Ex: 5+3 donnerait comme résultat 53).

VII.6 Des commandes sans programmation

Dans Delphi, il est possible d'introduire des commandes dans des menus sans avoir à écrire la moindre ligne de code. En effet, Delphi est doté d'un composant appelé **ActionList** qui propose un jeu de commandes qu'on retrouve dans la plupart d'applications professionnelles.

Exemple : Les commandes du menu Edition : couper, Copier et Coller.

Commencer d'abord par l'ajout d'un composant **ActionList** à partir de la palette **Standard**, ensuite double cliquer dessus. Dans la boite de dialogue qui apparaît cliquer sur **Nouvelle action standard**.

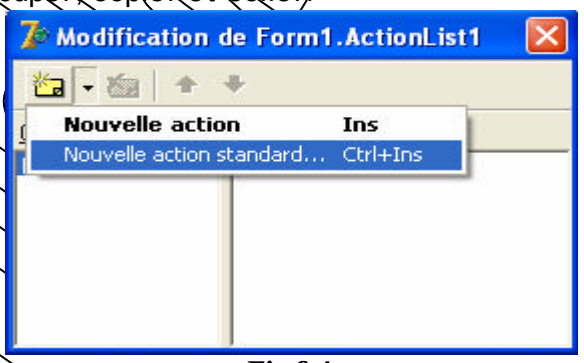


Fig 6.4

Une autre boite de dialogue apparaît, elle contient le jeu de commandes proposé par le **ActionList**. Sélectionner les actions que vous voulez ajouter, puis confirmer par OK. Dans notre cas ce sont : TeditCut, TeditCopy, TeditPaste.

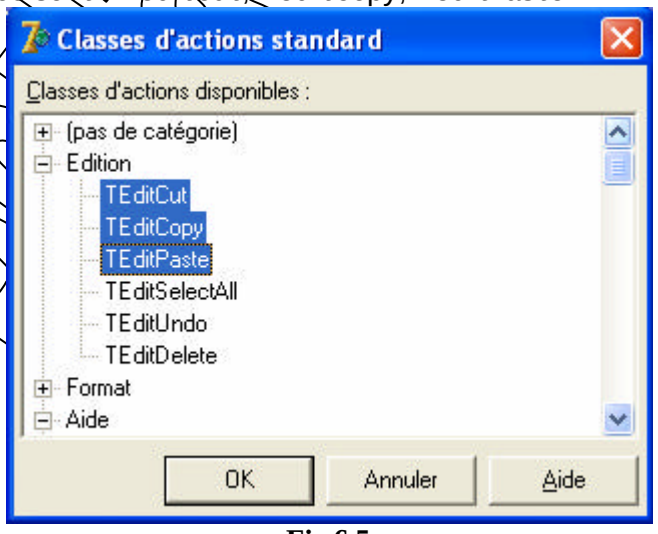


Fig 6.5

De retour à la fiche, double cliquer sur le composant du menu, puis sélectionner la commande Couper du menu Edition, ensuite à l'aide de la propriété Action de la commande, lier la à TeditCut. Procéder de la même manière pour Copier (TEditCopy) et Coller (TEditPaste).

VII.7 Création d'un menu contextuel

Ce type de menu existe depuis longtemps, mais il a acquis une signification particulière avec Windows95 : Lorsqu'on appuie sur le bouton droit de la souris, un menu contextuel s'affiche, il propose de nombreuses fonctions utiles.

Comme son nom l'indique, ce type de menu se réfère à un objet précis. Si vous appuyez sur le bouton droit de la souris alors que le curseur est dans une zone de saisie, un menu s'affiche, différent de celui qui s'affichera si le curseur était dans une zone de liste. Le menu n'est donc pas accessible à partir de la totalité de l'application.

Pour créer un menu contextuel, commencer d'abord par le dépôt sur la fiche d'un composant PopupMenu. Double cliquer dessus, et rajouter les même commandes incluses dans les menu Opérations et Edition.

Pour ajouter du code aux commandes du menu contextuel, il suffit d'associer les codes des commandes du menu **Opérations** aux commandes de ce menu et cela on utilisant l'événement **OnClick** de chaque commande du menu contextuel. Pour les commandes Couper, Copier et Coller utiliser la propriété **Action** de chaque commande.

Pour associer ce menu contextuel à un objet, sélectionner cette objet et affecter à sa propriété PopupMenu le nom du menu. Dans notre cas, sélectionner la fiche principale et mettez la propriété PopupMenu à PopupMenu1 du moment qu'on a créer qu'un seul menu contextuel.

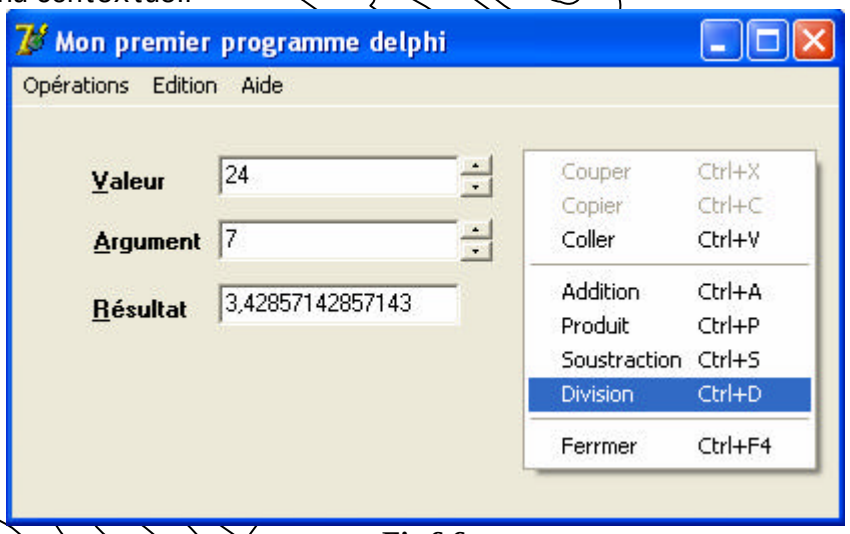


Fig 6.6

VII.8 Création d'une barre d'outils

A partir de la palette **Win32**, déposer sur la fiche un composant **Toolbar**. Par défaut, ce composant s'appelle ToolBar1 et s'affiche en haut de la fiche car sa propriété **Align** Vaut **AlTop par** défaut.

Cliquer avec le bouton droit de la souris sur le composant ToolBar et choisir dans le menu contextuel qui s'affiche **Nouveau bouton** ou **Nouveau séparateur**. Ajouter de cette manière quatre boutons pour les opérations de calcul puis un séparateur ensuite trois boutons pour les opérations du presse papiers Couper, Copier et Coller.

Toujours à partir de la palette Win32, déposer sur la fiche un composant **ImageList**. Ce composant permet de stocker la liste des images qu'on va placer sur chacun des boutons de la barre d'outils. cliquer avec le bouton droit sur **ImageList** et choisir la commande **Editeur de liste d'images.**

Dans la boite de dialogue qui apparaît, cliquer sur le bouton **Ajouter** puis choisir à partir d'un dossier vos images. Ces images sont indexées à partir de zéro. Le premier bouton de la barre d'outils recevra l'image zéro, le deuxième recevra l'image 1 et ainsi de suite. Fermer le boite de dialogue.

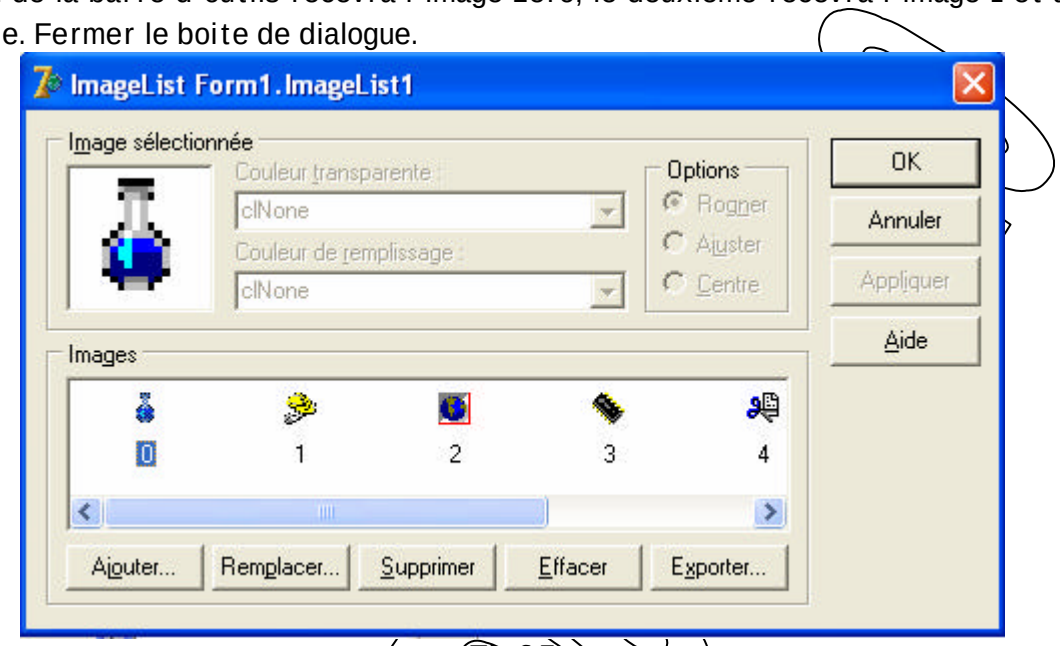


Fig 6.7

Pour affecter les images ainsi sélectionnées aux boutons de la barre d'outils, sélectionnez le composant Toolbar et mettez sa propriété **images** à **ImageList1**.

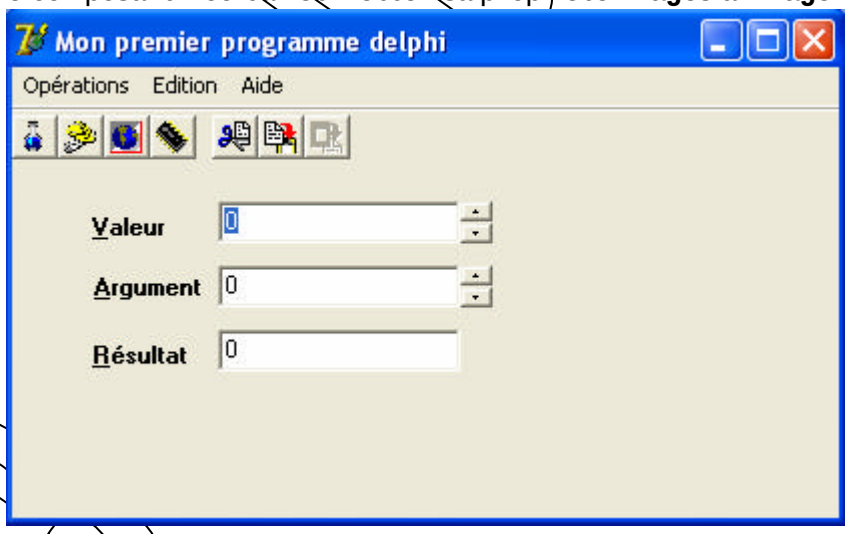


Fig 6.8

Pour associer du code aux boutons de la barre d'outils, sélectionner les boutons un à un puis dans l'événement OnClick lier chaque bouton à la commande de menu équivalente. De même, pour les commandes Couper, Copier et Coller utiliser la propriété **Action** de chaque bouton pour le lier à une action du **ActionList**.

VII.9 Des conseils pour guider le programme

Les textes d'aide qui accompagnent chacun des composants s'appellent Hint en anglais et Conseils (ou suggestions) en français. Ils s'affichent dans une boite à coté du composant (info-bulle) ou dans la ligne d'état du programme.

Remarques : Les règles préconisées par Microsoft pour la rédaction de des conseils sont les suivantes:

- L'aide sur une commande explique précisément et simplement les fonctions dont les utilisateurs disposent.
- Elle commence toujours par un verbe au présent.
- Si vous souhaitez que des composants affichent des info-bulles, sélectionner le composant en question (un bouton de barre d'outils par exemple) et affectez à sa propriété ShowHint la valeur True. Puis dans la propriété Hint taper le texte de l'info-bulle.

Par contre, pour afficher des textes d'aide sur la barre d'état, commencer d'abord par le dépôt d'un composant StatusBar qu'on trouve dans la palette Win32**.** Ce composant s'aligne sur le bord inférieur de la fiche parce que la propriété **Align**, qui définit l'alignement du composant dans la fiche est égale à **alBottom par** défaut.

Affecter la valeur True à propriété AutoHint de la barre d'état

Définissez la propriété SimplePanel à True afin de préciser que la ligne d'état ne se constitue que d'une seule section. La ligne d'état n'affiche alors que le texte stocké dans la propriété SimpleText. Si la propriété SimplePanel est à False, la ligne d'état se constitue de plusieurs sections.

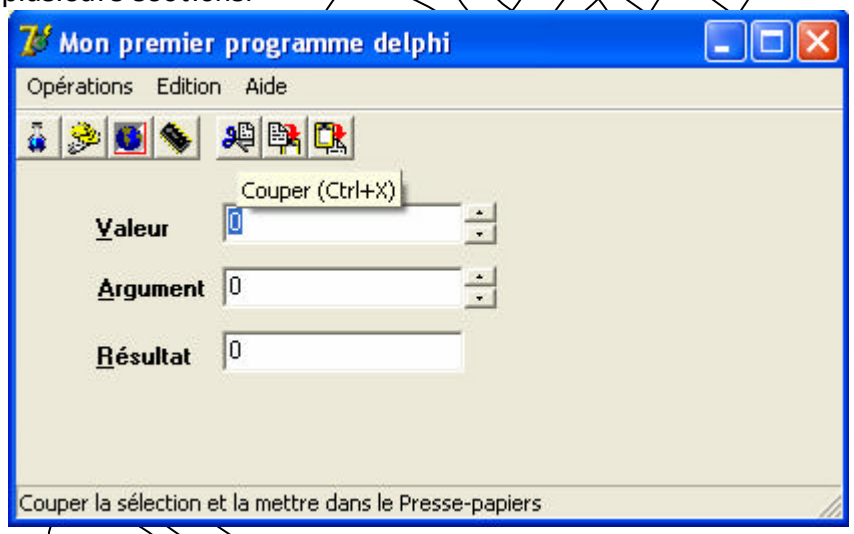


Fig 6.9

***Astuce* :** Pour simplifier le travail dans l'environnement de programmation, appuyer sur la touche F11 pour appeler l'inspecteur d'objets et modifier les propriétés d'un composant. Pour basculer entre l'éditeur de texte source et l'éditeur de fiches appuyer sur F12.

Revenez sur la fiche principale par F12 et sélectionner les composants lesquels vous voudriez qu'ils affichent des messages d'aides. pour ce faire, sélectionner un composant quelconque, mettez sa propriété ShowHint à True et taper le texte d'aide dans la propriété Hint.

Remarque

Le séparateur "|" qui s'obtient au clavier par la combinaison **Alt Gr + 6** permet de séparer le texte de l'info-bulle de celui à afficher sur la ligne d'état.

Si le conseil ne comprend pas de séparateur, la totalité du texte est considérée comme un conseil court (info-bulle) et comme conseil long (texte de la ligne d'état) et donc il sera affiché et comme info-bulle et comme texte de la ligne d'état. En revanche, si vous débutez le texte par le séparateur, seul le texte de la ligne d'état s'affiche. Et si par contre, vous tapez un texte que vous terminez par le séparateur, il ne s'affiche alors que le texte de l'info-bulle.

VII.10 Ajout d'une fiche d'information

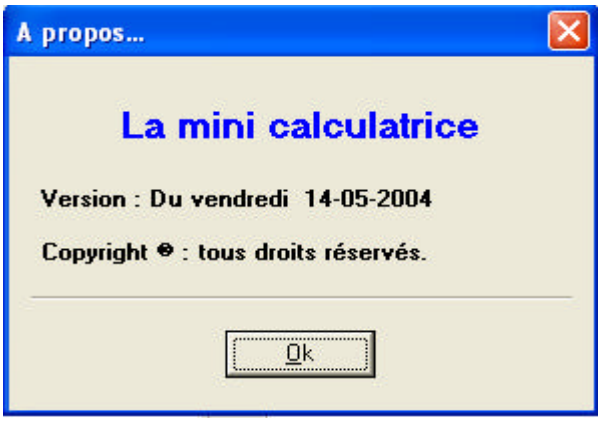
Créer une nouvelle fiche (nommée par défaut **Form2**) en cliquant sur l'icône Nouvelle Fiche situé sur la barre d'outils, ou bien choisir dans le menu Fichier la commande Nouvelle Fiche.

- Affectez à la propriété Caption de cette fiche le nom : A propos…
- Affectez à la propriété Height (Hauteur) la valeur 220.
- Affectez à la propriété Width (Largeur) la valeur 300.
- Affectez à la propriété BorderStyle la valeur bsDialog, cette propriété permet de rendre la taille de la fiche non modifiable, c'est à dire qu'on ne peut ni la réduire, ni l'agrandir (Boite de dialogue standard).
- Affectez à la propriété Position la valeur PoScreenCenter afin qu'elle s'affiche au centre de l'écran. En effet, la propriété Position permet de choisir l'emplacement de la fiche lors de l'appel.
- Déposer sur cette fiche trois composants label dont les propriétés Caption sont respectivement : La mini calculatrice, **Version** : Du vendredi 14-05-2004, **Copyright** ® : tous droits réservés.
- Utiliser la code ASCII Alt+169 pour le obtenir le symbole du copyright ®
- Rajouter un composant Button dont les propriétés sont les suivantes:

Propriété	**Valeur**
Caption	OK
ModalResult	mrOk

- Rajouter entre le troisième label et le bouton, un composant Bevel à partir de la palette Supplément pour créer un trait séparateur entre les labels et le bouton. Les propriété du Bevel sont :**Height** est égale à 3, **Width** est égale à 270 et **Shape** est égale à bsTopLine.

Vous obtiendrez une fiche qui ressemble à la **Fig 6.10**



Maintenant que la fiche **A propos** est terminée, on doit réaliser l'appel à celle ci à partir de la fiche principale. Lorsqu'on décide de l'affichage, on peut choisir s'il est modal ou non.

VII.11Les fiches modales et non modales

Pendant l'exécution, il est possible, sous Windows, de commuter entre plusieurs applications. Dans certains programmes, il est en outre possible de passer d'une fenêtre à une autre. Ces fenêtres sont non modales.

En revanche, si dans une application une fiche modale est active, elle intercepte la totalité des signaux provenant du clavier ou de la souris : une autre fenêtre ne peut être utilisée que lorsque la fenêtre modale est désactivée. C'est le cas des boites de dialogue par exemple.

Ainsi, on s'assure que l'utilisateur ne travaille pas dans une fenêtre quelconque alors que vous affichez des informations importantes dans une certaine fenêtre, ou que vous attendez une réponse.

Revenez sur la fiche principale en cliquant dans la barre d'outils sur le bouton Voir une fiche. Double-cliquer sur le composant MainMenu puis cliquer sur le menu Aide et enfin double cliquer sur la commande **A propos…** Vous devez définir l'action à déclencher à savoir l'appel (affichage) de la fiche. Tapez :

Dans le cas d'un affichage modal

```
Form2.ShowModal ;
```

Dans le cas d'un affichage non modal

```
Form2.Show ;
```

Dans notre cas, on va opter pour un affichage modal (Comme toutes les boites de dialogues A propos…). La fiche à propos possède la méthode ShowModal qui permet de l'afficher. ShowModal appelle une fiche en mode modal. Celle ci possède une propriété ModalResult, qui défini son état. La propriété à la valeur mrNone lorsque la fiche est active. Si elle reçoit la valeur mrOk ou mrCancel, la fiche est fermée. La valeur de la propriété ModalResult est retournée à ShowModal en tant que résultat de fonction.

Avant d'exécuter le programme, ajouter la clause **Uses** au dessous de la ligne **Implementation**, et taper ensuite le nom de l'unité de la fiche **A propos…** . Afin d'établir une liaison entre l'unité de la fiche principale et celle de la fenêtre A propos… . Comme suit:

```
implementation
Uses Unit2;
{$R *.DFM}
```

Où **Unit2** est le nom de l'unité associée à la fiche **A propos…** et sous lequel elle est enregistrée. Si cette unité a été enregistré sous un autre nom, il faut spécifier ce nom au lieu du nom par défaut.

Exécuter le programme en cliquant sur l'icône Exécuter située sur la barre d'outils ou bien Appuyer sur la touche **F9** au clavier. A ce stade vous êtes en mesure de faire appel à une fiche et de la fermer au besoin.

VII.12 Les exceptions

Vous avez sans doute déjà rencontré des erreurs de compilation et des erreurs logiques. Dans ce qui suit on va s'intéresser aux erreurs d'exécution. Ces erreurs résultent de l'échec d'une fonction ou d'une méthode opérant dans la bibliothèque d'exécution, la bibliothèque des composants visuels ou même dans le système d'exploitation. Le code susceptible de déclencher de telles erreurs doit envisager toutes les éventualités de traitement.

Une exception est un changement dans les conditions de traitement qui provoque l'interruption ou même l'arrêt de l'application. Une exception peut survenir n'importe où dans l'application. Elle peut avoir plusieurs sources qui sont toutes en interaction plus ou moins directe avec le code.

VII.12.1 Interception des erreurs

Les erreurs dans les programmes ont plusieurs sources: les utilisateurs peuvent fournir des données illégales pour les opérations envisagées, le programme essaye d'écrire sur une disquette protégée, ou encore d'accéder à une zone déjà attribuée de la mémoire, d'effectuer une division par zéro … La liste est quasiment infinie.

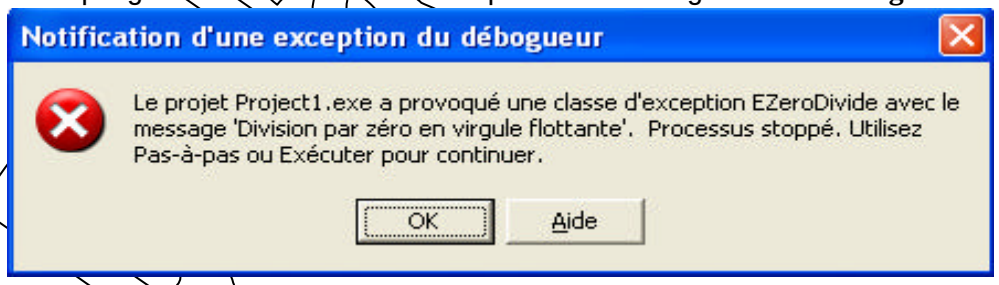
Les programmes devraient intercepter toutes les erreurs possibles, de façon qu'elles n'influent pas négativement sur le déroulement du programme. Les utilisateurs doivent être informés des causes possibles d'erreur et il convient d'assurer qu'une erreur ne bloque pas le programme.

Un message d'erreur s'appelle Exception. Le traitement des exceptions dans un programme s'appelle: Structured Exception Handling (SEH).

Lorsqu'une erreur se produit dans un programme , une exception est émise. Elle contient des informations sur la cause de l'erreur. Le traitement des exceptions exploite ces informations pour que le programme réagisse en accord. Delphi vous permet d'intercepter les exceptions et de les traiter.

VII.12.2 Lorsque l'exception paraît

Illustrons le traitement des exceptions par un exemple. Dans l'exemple précédent, il y avait une opération de division. Supposer que l'utilisateur tente de faire une division par zéro (0), c'est à dire que dans la zone de saisie **Argument**, il tape la valeur zéro. Une exception est générée. En effet, on ne peut pas effectuer cette opération. Le programme est donc interrompu et un message est émis : **Fig 6.11**



Nous avons affaire à une exception de la classe EZeroDivide. Le nom de l'exception indique qu'une erreur s'est produite lors de l'opération de division. L'exécution du programme est interrompue mais elle n'est pas terminée. Appuyer une deuxième sur la touche F9 pour continuer l'exécution, et examiner sa fenêtre de messages.

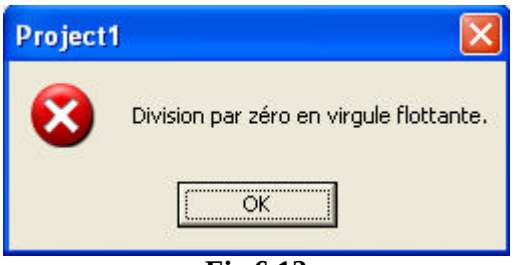


Fig 6.12

Les programmeurs ne tolèrent pas de tels messages d'erreur! nous devons traiter nous mêmes les mauvaises saisies des utilisateurs et rédiger un message compris par tous. C'est exactement le sens dans lequel vous pouvez exploiter le traitement des exceptions sous Delphi. Rédigez des blocs d'instruction appelés uniquement en cas d'erreur. votre programme aura tout le loisir de réagir aux erreurs.

VII.12.3 Traitement de l'exception

Modifions l'instruction de division partout dans le programme. Il doit vérifier si la donnée saisie dans **Argument** est différent de zéro (0). Si c'est le cas, l'opération de division se réalise et un résultat s'affiche. dans le cas contraire, notre programme émettra un message d'erreur.

L'élément décisif dans le texte source est le bloc **Try … Except.** Le mot réservé **Try** introduit le bloc. Le programme teste les exceptions des instructions qui suivent **Try**. **Except** ferme le bloc d'instructions surveillé. Si dans l'une de ces instructions une exception se produit, nous quittons immédiatement le bloc. L'exécution se poursuit après Except. A cet endroit figurent des instructions appelées lorsqu'une erreur se produit. Si aucune exception n'est signalée, cette zone est sautée.

```
Try
edit3.text:= floattostr(strtofloat(edit1.text)/strtofloat(edit2.text))
Except
Application.MessageBox ('Revoyez vos domaines de définition', 'Division par
zéro', mb_ok)
End;
```

Dans la section Try, le programme tente de faire un opération de division entre les données saisies dans Edit1 (Valeur) et Edit2 (Argument). Si cela n'entraîne aucune erreur, l'exécution se poursuit, et le résultat de l'opération est affichée dans Edit3 (Résultat). Si par contre l'opération de division n'est pas possible, elle déclenche une exception. Le programme se branche à la section Except et émet un message d'erreur.

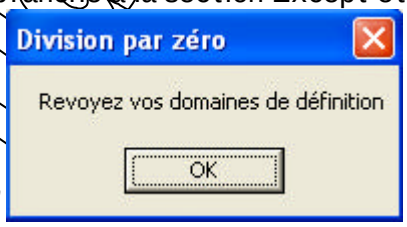


Fig 6.13

VII.13Conclusion

Vous venez de découvrir quelques composants fondamentaux vous permettant de concevoir une interface très simple. Vous avez traité leurs propriétés (caractéristiques) et leurs comportements pendant l'exécution du programme, c'est à dire leurs réactions à des événements (actions) donnés (tel qu'un double click ou une sortie de la fenêtre) en effectuant une opération bien définie. A travers cet exemple vous aurez compris que la gestion des événements n'est rien d'autre qu'un appel de procédure pris en charge par le gestionnaire d'événement dont vous vous contentez de préciser les réactions.

VIII SECOND EXEMPLE D'APPLICATION: UNE CALCULATRICE

A la fin de cette partie, vous serez capable de:

- Pouvoir attribuer une méthode à plusieurs composants,
- Maîtriser le paramètre "Sender",
- Travailler avec les composants d'une fiche,

VIII.1 Description de l'application

Dans ce chapitre nous allons écrire une application ayant comme point de départ une calculatrice simple. Nous compléterons graduellement la calculatrice en lui ajoutant des fonctionnalités jusqu'à la rendre relativement sophistiquée.

- Créez une nouvelle application et nommez par exemple votre projet Calculette.
- Placez sur la fiche un Label qui simulera l'affichage à cristaux liquides des calculettes.
- Placez 10 boutons correspondant aux dix chiffres, ajoutez les boutons pour les quatre opérations, un ou deux boutons pour les corrections, un pour le changement de signe, un pour la virgule et évidemment une touche égal.
- Il est possible de créer un bouton de la bonne taille et de le copier. Il reste alors à choisir pour chaque bouton, un nom (name) et son libellé (caption).
- Modifiez certaines propriétés du label : Autosize à false, Alignment à taRightJustify

Dans la suite les composants placés seront appelés : LabelAffichage.
BoutonZero, BoutonUn, BoutonDeux, BoutonTrois, BoutonQuatre, BoutonCinq, BoutonSix, BoutonSept, BoutonHuit, BoutonNeuf, BoutonPlus, BoutonMoins, BoutonMultiplier, BoutonDiviser, BoutonPlusMoins, BoutonVirgule, BoutonCorrection, BoutonEgal.

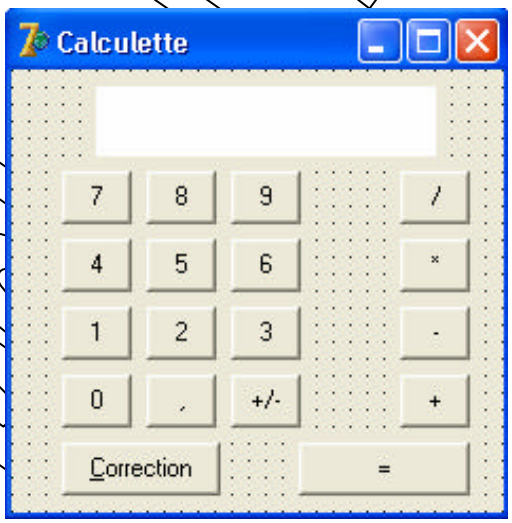


Fig 7.1

VIII.2 Les dix chiffres

A chaque fois que l'utilisateur choisira un chiffre, nous devons modifier la zone d'affichage en conséquence. Une solution simple consiste à créer dix méthodes (procédures) pour répondre à l'événement "OnClick" de chaque bouton.

```
procedure TForm1.BoutonSeptClick(Sender: TObject);
begin
LabelAffichage.Caption := LabelAffichage.Caption + '7';
end;
```

Il est cependant possible d'utiliser la même procédure pour répondre à l'événement "OnClick" de chaque bouton. Mais comment savoir alors s'il faut ajouter 1, 2 ou 3… ? C'est en utilisant la variable **Sender**. En effet, lorsqu'un événement est généré par un objet, la méthode associée reçoit en paramètre la variable **Sender** qui est un pointeur sur l'objet. Le pointeur permet d'accéder aux propriétés de l'objet sur lequel il pointe. Afin de préciser que l'objet est un bouton, nous devons indiquer au compilateur à l'aide du mot **as** que **Sender** est un pointeur sur un bouton (et non sur un objet quelconque). Un objet est de type TObject alors qu'un bouton est de type TButton.

```
procedure TForm1.BoutonChiffre(Sender: TObject);
begin
LabelAffichage.Caption := (Sender as TButton).Caption;
end
```

En fait, nous devons disposer d'une variable qui nous permettra de savoir si l'affichage doit être nettoyé ou s'il faut ajouter le chiffre à la fin. Enfin nous devons éviter les zéros au début du nombre.

Placez une variable de type booléen (boolean) dans la partie private de la classe, on pourra par exemple la nommer NouveauNombre.

```
procedure TForm1.BoutonChiffre(Sender: TObject);
begin
if NouveauNombre then
begin
     LabelAffichage.Caption := '';
     NouveauNombre := false;
end;
if LabelAffichage.Caption = '0' then
begin
   LabelAffichage.Caption := (Sender as TButton).Caption;
end
else
begin
   LabelAffichage.Caption := LabelAffichage.Caption + (Sender as
TButton).Caption;
end;
end;
```

Afin de pouvoir être connectée à un événement "OnClick", cette procédure devra être déclarée dans la classe, **avant** le mot **private.**

Pour connecter cette procédure à l'événement "OnClick" de chacun des 10 boutons chiffres, vous devez, dans le volet "Evénements" de l'inspecteur d'objet, choisir "BoutonChiffre" dans la liste des procédures compatibles avec cet événement. Vous ne devez donc pas faire un double clic pour créer une nouvelle procédure, vous devez au contraire choisir parmi celles qui existent.

VIII.3 Les touches opérateurs

Lorsque l'utilisateur choisit une des quatre touches correspondant aux quatre opérateurs, nous devons :

- Effectuer l'opération en cours s'il y en a une.
- Retenir le nombre actuellement affiché.
- Retenir qu'il faudra commencer un nouveau nombre.
- Retenir l'opération demandée.

Pour l'instant contentons-nous de créer une procédure vide "Calcule" qui se chargera plus tard d'effectuer l'opération en cours.

Nommons par exemple « Pile » une variable de type Extended (semblable au type real mais avec une meilleure précision) qui permettra de retenir le nombre affiché. Voir l'aide en tapant sur F1 alors que le curseur est sur le mot Extended. Codons par exemple avec le nombre 1 l'opérateur +, 2 l'opérateur -, 3 l'opérateur multiplier et 4 l'opérateur diviser.

Nommons par exemple Operateur (sans accent) une variable entière retenant ce code. Voici la méthode pour le bouton « plus ». Faites de même pour les trois autres opérateurs.

```
private
  NouveauNombre :boolean; Pile:extended;Operateur:integer;
```

```
procedure TForm1.Calcule;
begin
// cette procédure n'est pas encore réalisée
end;
procedure TForm1.BoutonPlusClick(Sender: TObject);
begin
     Pile := StrToFloat(LabelAffichage.Caption);
     NouveauNombre := true;
     Operateur := 1;
end;
```

StrToFloat permet de convertir une chaîne en un nombre décimal avec une précision importante.

VIII.4 La touche "égal"

Il s'agit d'effectuer le calcul et d'initialiser la variable NouveauNombre. Le calcul consiste à effectuer l'opération entre le nombre retenu dans la variable Pile et le nombre qui est à l'affichage en tenant compte de l'opération demandée. C'est le rôle de la procédure Calcule que nous allons terminer maintenant :

```
procedure TForm1.Calcule;
begin
if Operateur = 1 then
begin
LabelAffichage.Caption :=
FloatToStr(Pile + StrToFloat(LabelAffichage.Caption));
end;
if Operateur = 2 then
begin
LabelAffichage.Caption :=
FloatToStr(Pile - StrToFloat(LabelAffichage.Caption));
end;
if Operateur = 3 then
begin
LabelAffichage.Caption :=
FloatToStr(Pile * StrToFloat(LabelAffichage.Caption));
end;
if Operateur = 4 then
begin
LabelAffichage.Caption :=
FloatToStr(Pile / StrToFloat(LabelAffichage.Caption));
end;
Operateur := 0;
end;
procedure TForm1.BoutonEgalClick(Sender: TObject);
begin
Calcule;
NouveauNombre := true;
end;
```

VIII.5 La touche Correction

Si l'utilisateur clique sur la touche de correction, nous devons oublier le nombre dans la variable Pile, oublier l'opérateur et mettre 0 à l'affichage:

```
procedure TForm1.BoutonCorrectionClick(Sender: TObject);
begin
     Pile := 0;
     Operateur := 0;
     LabelAffichage.Caption := '0';
end;
```

VIII.6 La touche virgule

Une **virgule** ou un **point** ? Cela dépend de la configuration de Windows.

Dans le cas d'une virgule :

```
procedure TForm1.BoutonVirguleClick(Sender: TObject);
begin
if NouveauNombre then
begin
     LabelAffichage.Caption := '0';
     NouveauNombre := false;
end;
LabelAffichage.Caption := LabelAffichage.Caption + ',';
end;
```

VIII.7 La touche +/-

Si l'affichage commence par le signe moins, nous devons l'enlever, sinon nous devons le mettre. Il nous faut le moyen de comparer le premier caractère avec le symbole -. Nous pouvons utiliser la fonction **Copy**. Dans l'aide vous trouverez :

```
function Copy(S: string; Index, Count: Integer): string;
```

Dans notre cas **Copy**(LabelAffichage.Caption, 1, 1) correspond au premier caractère du Caption de LabelAffichage ou à la chaîne vide si le label est vide.

Pour supprimer le premier caractère nous pouvons également utiliser Copy avec 2 pour Index et un grand nombre pour Count afin de prendre tous les caractères restants.

```
procedure TForm1.BoutonPlusMoinsClick(Sender: TObject);
begin
if LabelAffichage.Caption <> '0' then
begin
if Copy(LabelAffichage.Caption, 1, 1)='-' then
begin
LabelAffichage.Caption := Copy(LabelAffichage.Caption, 2, 255);
end
else
begin
LabelAffichage.Caption := '-' + LabelAffichage.Caption;
end;
end;
end;
```

VIII.8 Le programme n'est pas parfait

En effet :

- Nous ne traitons pas le cas de la division par zéro.
- L'utilisateur peut taper un nombre trop grand.
- Il peut également taper plusieurs virgules dans un nombre.
- L'utilisateur aimerait certainement pouvoir taper les chiffres au clavier.

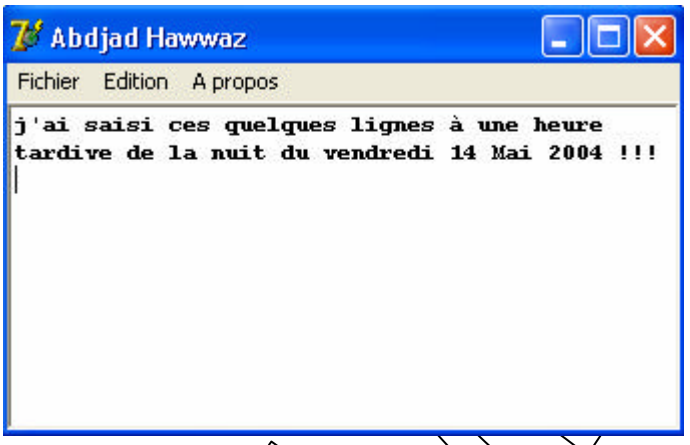
Pour cela :

- La division par zéro peut être traitée avec **try** et **except**.
- Nous pouvons limiter le nombre de caractères à l'aide de la fonction **length** qui permet de connaître le nombre de caractères d'une chaîne. Si ce nombre est trop grand, on n'effectue pas l'ajout du chiffre. L'utilisateur pourra toutefois multiplier plusieurs fois un très grand nombre par lui-même et finir par obtenir un message d'erreur.
- Pour tester si une virgule existe déjà, on peut utiliser la fonction **pos** qui retourne la position d'une sous-chaîne dans une chaîne (et si la sous-chaîne n'existe pas, la fonction retourne 0).
- Pour accepter la frappe des chiffres au clavier, il faut ajouter des méthodes pour répondre aux événements OnKeyPress, OnKeyDown de la fiche. On pourra aussi mettre à false la propriété TabStop de chaque bouton et mettre la propriété KeyPreview de la fiche à true.

IX UN PETIT ÉDITEUR DE TEXTES

Ce chapitre permet de montrer l'utilisation :

- du composant Mémo.
- des menus.
- des messages.
- des boîtes de dialogue.



IX.1 Composant Memo

Créez un nouveau projet, placez un composant "Memo" (il s'appelle par défaut Memo1 et est de type TMemo), enregistrez votre unité et votre projet puis exécutez votre programme.

Vous constatez que le composant Memo permet de taper du texte, de sélectionner, de copier (avec Ctrl C), de coller avec (Ctrl V), de couper (avec Ctrl X), de défaire (Crtl Z) ...

Si vous modifiez la propriété "Align" du composant Memo1 en choisissant "alClient" toute la fenêtre est occupée par ce composant. A l'exécution, si l'utilisateur change la dimension de la fenêtre, la dimension du composant change automatiquement en même temps.

Un fichier texte est en général montré avec une police non proportionnelle, on choisira "Courier New" pour le composant Memo1.

IX.2 Ajoutons un menu

Placez un composant "MainMenu" n'importe où sur la forme, puis faites un double clic dessus.

Ajoutez les menus et sous menus habituels :

&Fichier	&Edition	& A propos
&Nouveau &Ouvrir &Enregistrer Enregistrer &sous - &Quitter	&Défaire - &Couper Co&pier C&oller	&A propos.

Vous remarquez que le symbole & a été utilisé, il permet de définir la lettre qui sera soulignée et qui permettra donc à l'utilisateur de choisir rapidement dans le menu à l'aide du clavier. Le signe moins (-) permet de créer une ligne séparatrice.

Vous pouvez fermer la fenêtre d'édition du menu. Pour corriger ce menu par la suite, faites un double clic sur le composant placé sur votre forme et utilisez éventuellement les touches <Enter> <Inser> <Suppr>.

Si vous exécutez ce programme maintenant, vous verrez le menu mais il est encore inefficace.

IX.3 Une variable pour se souvenir si le texte est modifié

Lorsque l'utilisateur est sur le point de perdre son document, il est habituel de lui demander s'il souhaite l'enregistrer avant. Il ne faut pas demander systématiquement mais seulement si le document a été modifié. Il nous faut donc une variable booléenne pour retenir si le document a été modifié.

Créez par exemple la variable : `Modif: boolean ;` dans la partie `Private` de TForm1. Cette variable sera automatiquement mise à `false` au démarrage du programme. Elle devra être mise à `True` à chaque fois que l'utilisateur change le contenu du Memo1 (utilisez l'événement `OnChange` de Memo1).
Elle devra être mise à `false` :

- Après un enregistrement réussi,
- Lorsque l'utilisateur vient d'ouvrir un texte
- Et lorsqu'il choisit de commencer un nouveau texte.

IX.4 Permettre le choix du fichier à ouvrir

la variable `NomFich`. Placez cette variable dans la partie `Private` de TForm1 :

```
NomFich:string;
```

Nous allons placer n'importe où sur la forme un composant dialogue **OpenDialog**.

Pour appeler la fenêtre habituelle permettant le choix du fichier à ouvrir, il suffira de faire : `OpenDialog1.execute;`
ou mieux :

```
if  OpenDialog1.execute then
begin
{ ce qui doit être fait si l'utilisateur a validé son choix }
end;
```

Au retour, si l'utilisateur a validé son choix, le nom du fichier est placé dans `OpenDialog.FileName`

```
procedure TForm1.Ouvrir1Click(Sender: TObject);
begin
if  OpenDialog1.execute then
begin
     Memo1.Lines.LoadFromFile(OpenDialog1.FileName);
     NomFich:=OpenDialog1.FileName;
     Modif:=false;
end;
end;
```

Pour aider l'utilisateur à trouver les fichiers textes, modifiez la propriété `Filter` de `OpenDialog1`:

Fichiers textes (*.txt)	*.txt
Tous (*.*)	*.*

Voir dans l'aide la procédure LoadFromFile.

IX.5 Menu Enregistrer sous

Placez n'importe où sur la forme un composant `SaveDialog`. Pour enregistrer toutes les lignes de Memo1, on pourra utiliser la méthode `SaveToFile` définie par Delphi.

En mode conception, cliquez sur le menu `Fichier` puis sur la commande `Enregistrer` pour créer la procédure attachée à l'événement clic de ce sous menu. Complétez cette procédure comme suit :

```
procedure TForm1.Enregistrer1Click(Sender: TObject);
begin
if SaveDialog1.execute then
begin
     Memo1.Lines.SaveToFile(SaveDialog1.FileName);
     NomFich:=SaveDialog1.FileName;
     Modif:=false;
end;
end;
```

Modifiez la propriété `Filter` de `SaveDialog1` comme précédemment.

Améliorons la procédure du menu `Enregistrer`. Si `NomFich` est vide, c'est que l'utilisateur n'a pas encore choisi de nom pour son fichier. Nous devons en fait, faire comme s'il avait choisi `Enregistrer sous`.

```
procedure TForm1.Enregistrersous1Click(Sender: TObject);
begin
if  NomFich=' ' then
Enregistrersous1Click(nil)
else
begin
Memo1.Lines.SaveToFile(NomFich);
Modif:=false;
end;
end;
```

La procédure `Enregistrersous1Click` nécessite un paramètre de type `TObject`, mais comme ce paramètre n'est pas utilisé dans la procédure, n'importe quel paramètre convient pour peu que son type soit accepté. `Nil` est un pointeur qui pointe sur rien.

IX.6 Une fonction Fermer

Dans plusieurs cas, nous devons demander à l'utilisateur s'il accepte de perdre le document en cours. Il sera donc pratique de créer une fonction qui aura pour rôle de vider le composant Memo1 sauf si l'utilisateur décide d'abandonner la fermeture au dernier moment.

Principes

Tant que Memo1 n'est pas vide et que l'utilisateur ne désire pas abandonner nous devons, si le texte a été modifié, poser la question "Faut-il enregistrer le texte avant de le perdre ?".

Si l'utilisateur répond oui, nous appelons la procédure d'enregistrement. On sait que la procédure d'enregistrement a réussi en regardant la variable `Modif`. Si `Modif` est `false`, nous pouvons vider Memo1 et quitter la fonction `Fermer` avec succès.

Si l'utilisateur répond non, nous pouvons vider Memo1, mettre `Modif` à `false` et quitter la fonction `Fermer` avec succès.

Si l'utilisateur abandonne, nous ne vidons pas Memo1 mais nous quittons tout de même la fonction `Fermer`. Dans ce cas la fonction `Fermer` retourne `false` pour montrer que la fermeture a échoué.

Aides

Le texte est vide si Memo1 n'a aucune ligne. Le nombre de lignes est donné par :

```
Memo1.Lines.Count
```

On peut aussi utiliser la propriété Text de Memo1. Text contient la totalité du texte contenu dans le composant Mémo, chaque chaîne autre que la dernière est terminée par les codes 13 et 10 correspondant au passage à la ligne (retour chariot).
Pour indiquer la valeur de retour d'une fonction nous utiliserons result. Dans notre cas nous pouvons quitter cette fonction avec

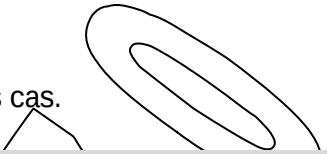
```
result:=true;
```

ou

```
result:=false;
```

Utilisez MessageDlg pour poser la question.

Utilisez l'instruction Case of pour envisager les trois cas.



```
Function TForm1.Fermer:boolean;
var Abandon:boolean;
begin
Abandon:=false;
while (Memo1.Lines.Count<>0) and not Abandon do
begin
if Modif then
begin
case MessageDlg('Enregistrer ?',mtConfirmation,[mbYes,mbNo,mbCancel],0) of
mrYes:
begin Enregistrer1Click(nil);
if not Modif then Memo1.Lines.Clear;
end;
mrNo:
begin
Memo1.Lines.Clear;
Modif:=false;
end;
mrCancel:Abandon:=true;
end;{case}
end
else
begin
Memo1.Lines.Clear;
end;
end;
if Memo1.Lines.Count=0 then
begin
result:=true;
NomFich:=' ';
end
else
begin
result:=false;
end;
end;
```

IX.7 Fichier Nouveau

Il suffit d'appeler la fonction Fermer sans même se préoccuper de sa valeur de retour.

IX.8 Menu Quitter

Le mot close est suffisant pour quitter.

```
procedure TForm1.Quitter1Click(Sender: TObject);
begin
Close;
end;
```

Lorsque la procédure close est appelée, elle génère un événement OnCloseQuery. Si nous attachons une procédure à cet événement, il nous sera encore possible de ne pas quitter.

Une idée serait de remplacer Close par : `If Fermer then close;`

Mais cette façon de faire ne convient pas. En effet, Fichier/Quitter n'est pas la seule façon de quitter un programme Windows.

IX.9 Evénement OnCloseQuery

Cet événement se produit lorsque l'utilisateur est sur le point de quitter. Pour quitter, l'utilisateur peut utiliser Fichier/Quitter, la procédure Close est alors appelée, mais il peut aussi cliquer sur la case Quitter (en haut à droite) ou utiliser le menu système (en haut à gauche) ou faire Alt F4…

Dans tous les cas l'événement OnCloseQuery se produit. Il est encore possible de ne pas quitter.

Voici la procédure attachée à cet événement :

```
procedure TForm1.FormCloseQuery(Sender: TObject; var CanClose: Boolean);
begin.

end;
```

En mettant dans cette procédure `CanClose:=false;` le programme ne s'arrête plus. Si vous voulez faire l'essai, il est prudent d'enregistrer votre projet avant. Vous pouvez toutefois arrêter votre programme en mettant un point d'arrêt sur la ligne située après la ligne contenant CanClose, faites Fichier et Quitter. A l'aide de la fenêtre Evaluation / modification (Ctrl F7 lorsque le curseur est sur le mot CanClose)

Mettez à True la variable CanClose. Tapez F9 pour terminer correctement votre programme. En fait, CanClose ne devra être mise à False que si la fonction Fermer échoue.

```
procedure TForm1.FormCloseQuery(Sender: TObject; var CanClose: Boolean);
begin
if not Fermer then CanClose := false;
end;
```

IX.10 Le menu Edition

Comme le composant Memo1 sait déjà "Défaire", "Copier", "Couper" et "Coller", il suffit de lui envoyer un message pour demander ce travail. En regardant dans l'aide on trouve :

```
LRESULT SendMessage(
HWND hWnd, // handle of destination window
UINT Msg, // message to send
WPARAM wParam, // first message parameter
LPARAM lParam // second message parameter
);
```

Il s'agit de la déclaration en langage C de la fonction SendMessage. Parmi les quatre paramètres seuls les deux premiers nous intéressent ici.

hWnd est le handle de l'objet à qui nous allons envoyer le message. Le handle est un numéro (un integer) qui permet d'accéder à un objet.

Lorsqu'un objet est créé, il est placé en mémoire. Windows se réserve le droit à tout moment de déplacer les objets dans sa mémoire. Il n'est donc pas question, pour

l'utilisateur d'accéder directement à un objet comme on pouvait le faire avec les "vieux" programmes dos. Le handle est attribué par Windows lors de la création de l'objet et ne changera pas pendant toute la durée de vie cet objet.

Le handle de Memo1 est obtenu par : `Memo1.Handle`

Msg est un numéro (un integer) de message. Pour avoir une liste des messages utilisés par Windows vous pouvez regarder le fichier Message.pas par exemple avec le bloc-notes. Si Delphi a été installé dans le répertoire habituel sur le disque C, voici le chemin à utiliser pour ouvrir ce fichier :

`c:\Program Files\Borland\Delphi7\Source\Rtl\Win\Messages.pas`

Vous trouverez par exemple que WM_UNDO correspond à $304.

Pour défaire la dernière action faite par l'utilisateur dans Memo1, nous pouvons utiliser : `SendMessage(Memo1.Handle,WM_UNDO,0,0);`

Memo1 recevra le message WM_UNDO et exécutera la procédure correspondante.

Edition Copier, Couper, Coller :

Il est possible d'utiliser également SendMessage avec :

`WM_COPY, WM_CUT et WM_PASTE.`

Il est cependant plus facile d'utiliser les méthodes

```
CopyToClipboard;
CutToClipboard;
PasteFromClipboard;
```

Vous trouverez ces méthodes dans l'aide de TMemo.

IX.11 A propos

Beaucoup d'applications Windows possèdent une boîte de dialogue souvent nommée "A propos", qui donne le nom du programme, sa version, le nom de l'auteur...

Faites "Fichier" et "Nouvelle Fiche" (avec Delphi vous pouvez aussi faire "Fichier" "Nouveau" "Fiches" et "Boîte à propos"). Nommez cette forme par exemple Bap et enregistrez l'unité en l'appelant par exemple Ubap Améliorez cette fiche...

Il faut au moins un bouton "Ok" ou "Fermer" qui aura pour rôle de fermer cette fenêtre facilement. Pour l'appeler à partir du menu "A propos" de la forme principale, vous pouvez attacher la procédure suivante au sous menu "A propos".

```
Bap.Showmodal;
```

L'utilisateur devra quitter la boîte "A propos" pour pouvoir continuer à utiliser le mini éditeur.

IX.12 Idées d'améliorations

Si l'utilisateur ne met pas d'extension, ajoutez l'extension txt. La fonction suivante ajoute .txt si nécessaire :

```
function TForm1.AjouteEventuellementExtension(NF:string;Ext:string):string;
begin
if UpperCase(ExtractFileExt(NF))='' then
      result:=NF + Ext
else
      result:=NF;
end;
```

Par exemples

```
AjouteEventuellementExtension('Bonjour','.txt') retournera 'Bonjour.txt'
AjouteEventuellementExtension('Bonjour.abc','.txt') retournera 'Bonjour.abc'
AjouteEventuellementExtension('Bonjour.','.txt') retournera 'Bonjour.'
```

On utilisera cette fonction avant de faire appel à SaveToFile

- La variable Modif n'est pas nécessaire pour un composant Mémo car ce composant possède déjà la variable Modified.
- Dans le menu, utilisez la propriété "ShortCut" pour Enregistrer, Défaire, Copier, Couper et Coller en choisissant respectivement Ctrl+S, Ctrl+Z, Ctrl+C, Ctrl+X, Ctrl+V.
- Ajoutez un sous menu Rechercher dans le menu Edition. Vous pouvez profiter du composant FindDialog. Pos donne la position d'une sous chaîne dans une chaîne.
- Remplacez Memo1 par un composant RichEdit et il vous sera facile d'imprimer.
- Ajoutez un composant StatusBar pour y mettre le numéro de ligne et de colonne.

Sommaire

INTRODUCTION **1**

I INTRODUCTION **3**

I.1 PRÉSENTATION 3
I.2 DELPHI ET LES AUTRES 3
I.3 PRINCIPES DE DÉVELOPPEMENT EN DELPHI 4
I.4 DELPHI ET WINDOWS 4

II ENVIRONNEMENT DE TRAVAIL **5**

II.1 L'INTERFACE DE DÉVELOPPEMENT DE DELPHI 5
II.2 LA BARRE DE MENU 5
II.3 LA BARRE D'OUTILS 6
II.4 LA PALETTE DES COMPOSANTS 6
II.5 L'INSPECTEUR D'OBJET 8
II.5.1 Modifier les propriétés *9*
II.5.2 Les propriétés événement *9*
II.6 LES ÉDITEURS 10
II.6.1 Editeur de fiches *10*
II.6.2 L'éditeur de texte source *10*
II.7 STRUCTURES DES FICHIERS DU PROJET DELPHI 11

III NOTIONS DE BASE SUR LA POO **12**

III.1 ENCAPSULATION : 12
III.2 HÉRITAGE : 12
III.3 DROITS D'ACCÈS : 13
III.4 LA PROGRAMMATION STRUCTURÉE 14
III.5 LES VARIABLES 15
III.6 OÙ PEUT-ON PLACER LES DÉCLARATIONS? 15
III.7 LES TYPES 16
III.7.1 Généralités sur les types *16*
III.7.2 Les types numériques *16*
III.7.3 Les types chaînes *17*
III.7.4 Le type boolean *17*
III.7.5 Les types tableaux *17*
III.7.6 Les types enregistrements *18*
III.8 TESTS 18
III.8.1 If then *18*
III.8.2 If then else *20*
III.9 UNE BOUCLE 21
III.10 DÉBOGAGE 22
III.11 COMMENT SUPPRIMER UN COMPOSANT 22

IV QUELQUES CONSEILS **23**

IV.1 QUESTIONS FONDAMENTALES 23
IV.2 COMMENT IMPRIMER ? 24
IV.2.1 Impression du contenu d'une fiche *24*
IV.2.2 Utilisation du printdia log *24*
IV.2.3 Impression par ma méthode "Brute" *24*
IV.3 COMMENT INTRODUIRE UN DÉLAI D'ATTENTE ? 25

IV.4 COMMENT LAISSER LA MAIN À WINDOWS DE TEMPS À AUTRE ? 25
IV.5 APPLICATION COMPORTANT PLUSIEURS FICHES 25

V PREMIERS PAS 27

V.1 ADJONCTION D'OBJETS SUR UNE FICHE 27
V.2 1ÈRE ÉTAPE: MISE EN PLACE D'UN BOUTON 27
V.3 SAUVEGARDE DU PROJET 28
V.4 POURQUOI TANT DE FICHIERS ? 29
V.5 2ÈME ÉTAPE: UTILISATION DES COMPOSANTS 30
V.6 QUELQUES PRÉCISIONS: 33
V.7 CONCLUSION 35

VI EXEMPLE D'APPLICATION 36

VI.1 COMMENT PRÉPARER UN PROJET ? 36
VI.2 RÉSUMÉ DU PROJET 36
VI.3 CRÉER LA FICHE PRINCIPALE 36
VI.4 AJOUTER DES COMPOSANTS 37
VI.5 CRÉATION D'UN MENU 38
VI.6 DES COMMANDES SANS PROGRAMMATION 39
VI.7 CRÉATION D'UN MENU CONTEXTUEL 40
VI.8 CRÉATION D'UNE BARRE D'OUTILS 40
VI.9 DES CONSEILS POUR GUIDER LE PROGRAMME 42
VI.10 AJOUT D'UNE FICHE D'INFORMATION 43
VI.11 LES FICHES MODALES ET NON MODALES 44
VI.12 LES EXCEPTIONS 45
VI.12.1 Interception des erreurs 45
VI.12.2 Lorsque l'exception paraît 45
VI.12.3 Traitement de l'exception 46
VI.13 CONCLUSION 46

VII SECOND EXEMPLE D'APPLICATION: UNE CALCULATRICE 47

VII.1 DESCRIPTION DE L'APPLICATION 47
VII.2 LES DIX CHIFFRES 47
VII.3 LES TOUCHES OPÉRATEURS 48
VII.4 LA TOUCHE "ÉGAL" 49
VII.5 LA TOUCHE CORRECTION 50
VII.6 LA TOUCHE VIRGULE 50
VII.7 LA TOUCHE +/- 50
VII.8 LE PROGRAMME N'EST PAS PARFAIT 51

VIII UN PETIT ÉDITEUR DE TEXTES 52

VIII.1 COMPOSANT MEMO 52
VIII.2 AJOUTONS UN MENU 52
VIII.3 UNE VARIABLE POUR SE SOUVENIR SI LE TEXTE EST MODIFIÉ 53
VIII.4 PERMETTRE LE CHOIX DU FICHIER À OUVRIR 53
VIII.5 MENU ENREGISTRER SOUS 53
VIII.6 UNE FONCTION FERMER 54
VIII.7 FICHIER NOUVEAU 55
VIII.8 MENU QUITTER 55
VIII.9 EVÉNEMENT ONCLOSEQUERY 56
VIII.10 LE MENU EDITION 56
VIII.11 A PROPOS 57
VIII.12 IDÉES D'AMÉLIORATIONS 57

SOMMAIRE 59