DELPHI

Les outils classiques de développement BASIC, PASCAL,C permettent d'écrire des programmes qui , au moins à l'origine , tournaient sous DOS. Pour introduire les données et lire les résultats l'utilisateur se trouvait le plus souvent devant un écran en mode texte. Rien n'empêchait bien sûr le programmeur d'écrire un logiciel graphique pour présenter un écran en mode fenêtre similaire avec celui auquel nous a habitués Windows, mais l'écriture d'un tel outil graphique est ardue et exige de larges connaissances en programmation.

Les utilisateurs aujourd'hui habitués à l'environnement Windows (ou LINUX) ne supportent plus un écran purement texte, plus grave les programmes écrits sous DOS risquent de ne plus fonctionner sur un ordinateur récent.

VISUAL BASIC fut le premier outil permettant d'écrire rapidement des programmes affichant des écrans de type Windows et tournant dans cet environnement. Il fournit a l'utilisateur des blocs d'instructions directement implantables pour gérer un écran graphique. Le programme proprement dit est dans ce cas écrit en BASIC. Le VISUAL C++ plus récent fournit le même service mais exige une programmation en C++. Nous décrirons ici DELPHI développé par la société BORLAND qui s'appuie sur une programmation en PASCAL. C'est un outil largement utilisé dans l'industrie , performant, d'emploi aisé .Le programmeur qui utilise chaque jour le langage C lui préférera sans doute le VISUAL C++. mais la philosophie des deux outils est assez semblable et il est sans doute assez facile de passer de l'un à l'autre .

L'écran sous Windows est divisé en fenêtres (d'où le nom) dans lesquelles on peut introduire les données et lire les résultats. DELPHI fournit à l'utilisateurs des sous programmes tous faits permettant de visualiser à l'écran de telles fenêtres. On peut demander au programme de faire beaucoup plus ; communiquer par liaison série, permettre l'écriture ou la lecture de fichiers sur tout support magnétique ou optique (CD, DVD), autoriser une entrée sous forme de dessins, lire des fichiers WAV ou des CD audio. DELPHI fourni la aussi tous les outils nécessaires évitant à l'utilisateur de rentrer dans la complexe programmation sous Windows .Nous ne passerons pas en revue toutes ces possibilités, lorsque vous aurez compris le 'truc' vous pourrez vous plonger dans la documentation fort nombreuse aussi bien en librairie que sur le web.

Il existe en effet une très abondante littérature sur ce programme ainsi que des cours,(voir plus loin une bibliographie) mais le plus souvent les auteurs semblent avoir oublié les difficultés qu'ils ont sans doute rencontrées au début et veulent en dire trop. Comme c'est toujours le cas il existe pour réaliser une tâche donnée de multiples méthodes, mais il faut dans un cours d'introduction à but pédagogique n'en citer au début qu'une seule , laissant au lecteur le soin de compléter plus tard sa formation lorsqu'il aura assimilé le mécanisme du logiciel.

Nous ne décrirons pas la procédure d'installation de DELPHI à partir du CD ou d'un fichier téléchargé, ,elle ne présente pas de difficulté, il suffit de suivre les indications apparaissant à l'écran. Les exemples suivants ont été écrits sous DELPHI 7 ,ils sont compatibles avec les autres versions du logiciel (au-delà de la version 4 sans doute). Actuellement BORLAND commercialise la version 10 .
Les versions 6 et 7 personnelles sont téléchargeables gratuitement sur le web (plus de 100Mo) et fournies sur CD avec certains ouvrages.

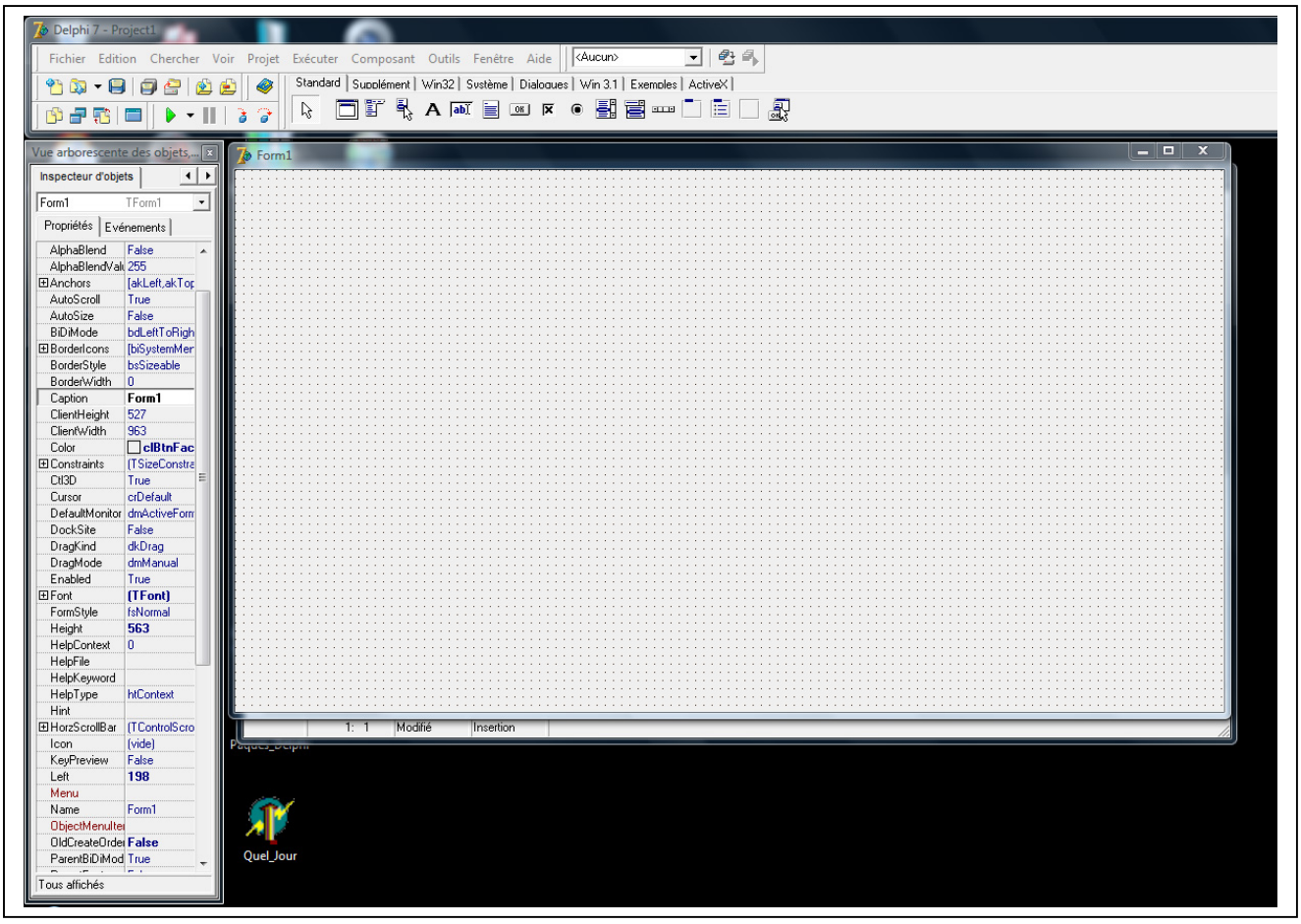
Le langage de programmation utilisé par Delphi est le PASCAl ou plus exactement un PASCAL Objet ,ce texte n'est pas un cours de PASCAL, le lecteur est supposé connaître l'essentiel du PASCAL classique , des informations concernant la partie Objet sont introduites au fur et à mesure des besoins. Un document très complet peut être extrait du CD officiel de Delphi c'est le guide du développeur., un gros pavé de plus de 1500 pages mais essentiel.

Documentation :

http://fbeaulieu.developpez.com/guide/ Un cours très complet, parfois même un peu trop, sur Pascal objet et Delphi
http://dialog.ac-reims.fr/web/Documents/delphi.pdf Une introduction avec exemples intéressants. (écrit au temps de delphi4)
http://delphi.developpez.COM/telecharger/gratuit/delphi7-perso/ Téléchargement de la version 7 de Dlelphi , sur le site on trouvera aussi Delphi6.
Michel Pelletier Delphi 5 Le Tout en poche Campus press Pas toujours très pédagogique mais des exemples intéressants bien détaillés.
L'intro Développement DELPHI 6 Thomas Binzinger Campus Press 2002 Un très bon livre
Le guide du développeur Sur le CD de Delphi ou sur le web (plus de 1500 pages) : http://noisetteprod.developpez.com/delphi/doc/pdfd7
http://pagesperso-orange.fr/patrice.rabiller exemples de sujets donnés à des élèves de seconde.

L'Environnement DELPHI

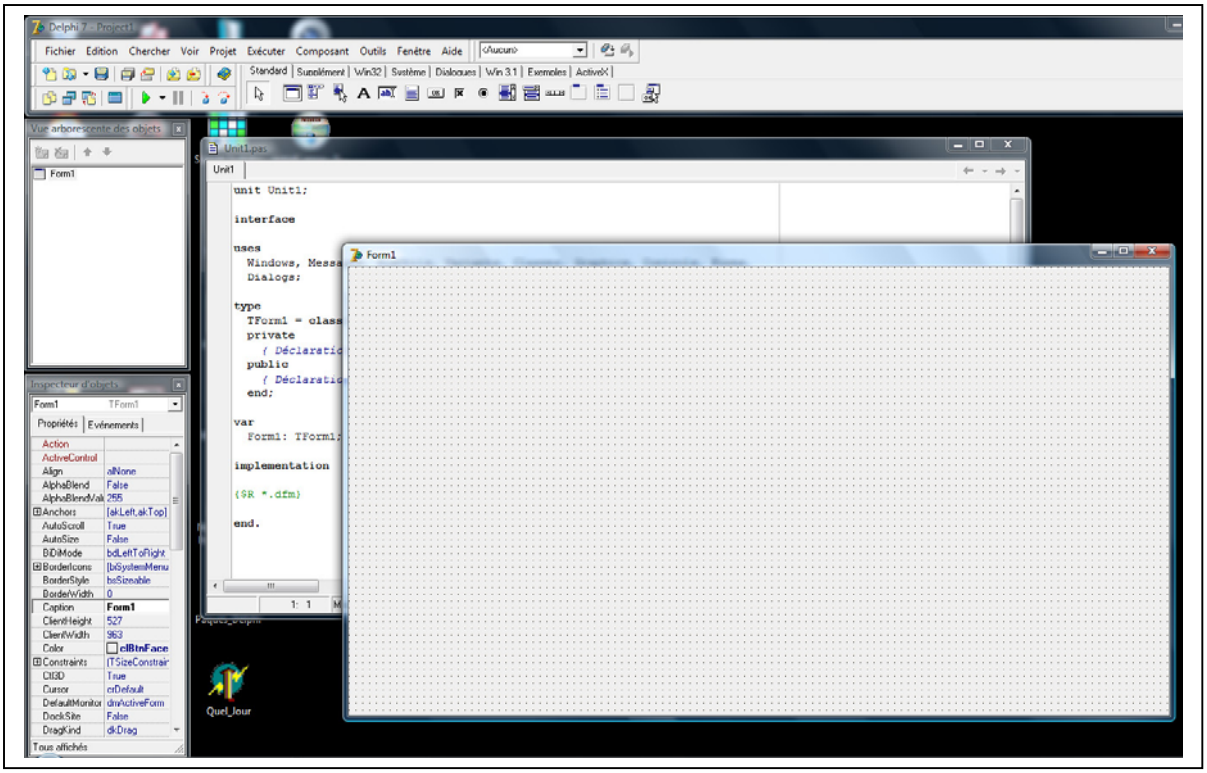
Au lancement du logiciel l'écran a l'aspect ci dessous.



La partie supérieure de l'écran donne accès au système de menus et à la barre d'outils.
Sur la première ligne :
Fichiers permet d'ouvrir un nouveau fichier, un nouveau projet, d'enregistrer votre travail et d'imprimer.
Edition donne accès aux fonctions copier coller classiques ainsi qu'a des outils de présentation.
Exécuter permet de lancer l'exécution d'un programme

La feuille de travail présente à l'écran est le rectangle situé à droite, sous Delphi on parle de **FICHE** , un projet peut comporter plusieurs FICHES de ce type , Au départ la fiche présentée à l'écran porte le nom initial **Form1** .

Cette fiche masque en fait une fenêtre dans laquelle s'écrira le programme source , c'est la fenêtre **UNIT** que l'on peut visualiser en déplaçant la FICHE avec la souris puis en cliquant dessus , mais nous y reviendrons.



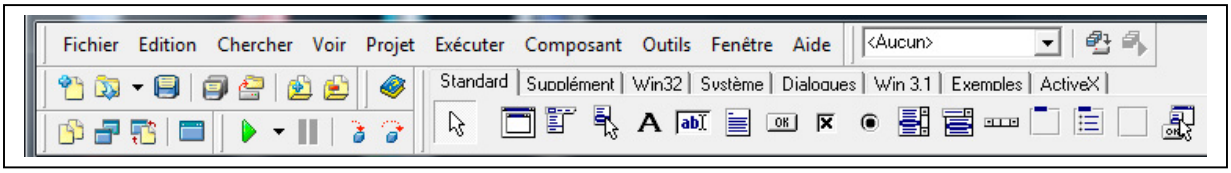
Sur la fiche le programmeur va placer différentes fenêtres secondaires permettant la saisie de données , l'affichage de résultats, des boutons sur lesquels on viendra cliquer avec la souris et peut être aussi des textes et dessins. Toutes ces choses sont des **OBJETS.**

Il y a de nombreux objets définis dans DELPHI , l'utilisateur confirmé peut même en écrire d'autres , mais c'est une autre histoire ! Ces objets sont graphiques ou non .

Chaque objet possède un certain nombre de caractéristiques (taille couleur format du texte etc…) et peut effectuer diverses tâches qui caractérisent le **type** auquel il appartient.

En PASCAL classique on trouve la notion de type pour définir les éléments qui constituent un programme, type booléen, caractère, chaîne etc…Cette notion de type se retrouve dans DELPHI mais en beaucoup plus complexe. Pour manipuler un objet qui apparaît dans un écran sous Windows il est nécessaire d'écrire de nombreuses lignes de code, elles définissent le type de base TObjet à partir duquel sont construits tous les autres en ajoutant à chaque fois une couche logicielle supplémentaire. Cette structure hiérarchisée est une caractéristique essentielle des langages objet L'Objet button par exemple est de type Tbutton qui est un descendant de Tobjet , il en est de même de Tform .

Pour accéder à ces objets regardons la seconde ligne de l'écran .En cliquant sur le premier bouton **Standard** nous voyons apparaître une première liste



Parmi tous ces objets nous retiendrons surtout

A Définit à l'écran une zone dans lequel on peut écrire une ligne de texte .Reconnu dans DELPHI sous le nom de **Label** de type Tlabel.

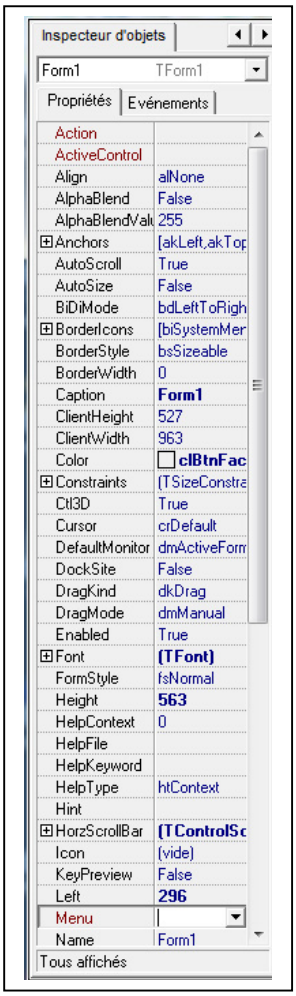
ab est une zone de saisie , c'est dans cette fenêtre que nous introduirons les données demandées par le programme nombres ou caractères. Reconnu dans DELPHI sous le nom de **Edit** de type TEdit

ok est un bouton de commande sur lequel il faudra cliquer avec la souris , sous DELPHI **Button** de type Tbutton .

Nous utiliserons surtout ces 3 outils . Les boutons supplémentaires de la seconde ligne:

Supplément Win32 Système etc donnent accès à d'autres objets , nous en rencontrerons quelques uns par la suite .

Chaque objet possède des **propriétés** qui lui sont propres , taille , fonte utilisée pour le texte, nature du texte , nom etc.. Ces propriétés apparaissent dans la partie gauche de l'écran . **l'inspecteur d'objets** lorsque l'objet est sélectionné.



Nous citerons

Caption (en français légende) c'est le texte qui apparaît dans l'objet considéré

Name c'est le nom de l'objet utilisé par le logiciel , ne pas le confondre avec le précédent.

Color la couleur

Font : qui permet de choisir la fonte utilisée pour les caractères , taille et couleur.

On peut noter encore **AutoSize** (True ou False) qui gère l'ajustage automatique de la fenêtre à celle du texte Caption. , **Visible** (True ou False) qui définit la visibilité à l'écran de l'objet considéré, on peut ainsi masquer des fenêtres pour ne faire apparaître qu'une seule à un moment donné.

La colonne **evenements** permet pour chaque objet de définir des circonstances le concernant, par exemple pour un bouton le déclenchement s'effectuera à l'enfoncement du bouton (**Onclick**) ou son relâchement .

Ci contre l'inspecteur d'objets correspondant à la fiche Form1 elle même , noter **Caption**=Form1 c'est le nom de la fiche **Color**=ClbTnFace (un gris) , sa hauteur **Height**=563 etc…

Premier PROJET

La construction d'un projet comporte deux étapes essentielles, la mise en place de la présentation à l'écran puis l'écriture du logiciel PASCAL associé à tous les objets.

Notre premier projet est très simple mais illustre bien le mécanisme de DELPHI . Nous nous proposons d'écrire un programme d'extraction de racine carrée de nombres entiers.

.L'écran sous Windows sera constitué de deux zones de texte et d'un bouton. La première est la zone de saisie dans laquelle nous allons entrer un nombre entier Dans la seconde apparaît le résultat obtenu en cliquant sur le bouton .
Ces trois objets sont placés sur un fond vert.

Si l'écran d'entrée n'est pas celui décrit plus haut y revenir par ***Fichier*** *⇒ Nouveau une fenêtre s'ouvre, sélectionner ⇒****application***

La première chose à faire est de définir les noms des fichiers constituant le projet, le non du projet lui même et celui de la fiche et programme pascal.

Pour cela effectuons un premier sauvetage de sécurité par :
Fichier ⇒Tout Enregistrer.

Une fenêtre s'ouvre qui nous demande d'abord le nom de l'unité ,c'est à dire du programme PASCAL, par défaut unit1 , choisissons **premier** , il faut ensuite entrer le nom du projet , par défaut Project1 que nous changeons en **Projet 1**

Dans le répertoire Projet plusieurs fichiers sont crès , ils constituent l'ensemble des fichiers nécessaires pour la compilation du projet .

Extension	Description
DPR	Contient l'unité principale du projet
DFM	Contient les informations relatives à une fiche taile couleur etc… C'est un fichier texte
PAS	C'est l'unité écrite en Pascal
DCU	PASCAL + DFM
EXE	L'exécutable fonctionne sous Windows
RES	Ressources de l'application par exemple son icône
DOF/DSK/CFG	Fichiers d'options
~ ???	Tous les fichiers dont l'extension commence par ~sont provisoires et peuvent être effacés

Voir note 1 en fin de document.

Nous allons maintenant construire l'aspect graphique du projet.

Revenons à la fiche dont le nom est toujours Form1. Sur l'Inspecteur d'objets nous modifierons la couleur en cliquant à droite sur la ligne **Color** , une liste déroutante apparaît, choisissons le vert qui a pour nom clLime . En revenant sur la fiche est devient verte.

Nous allons ensuite écrire le titre **Mon premier programme** dans un objet qui est une zone de texte , l'objet **A** vu plus haut. Dans la ligne des objets cliquons sur A , qui est alors activé (il apparaît dans un carré plus clair) , puis au centre de la fiche ou s'inscrit le mot **label1** (nom par défaut) (entouré de petits carrés noirs, les

poignées de réglage , indiquant que l'objet est activé.) Dans la fenêtre de gauche apparaissent les propriétés de cet objet.

La ligne **Name** = Label1 c'est le nom par défaut que nous ne modifierons pas (mais c'est possible) la ligne **Caption** présente le même nom , c'est lui qui est visible à l'écran nous y introduirons le titre choisi **Mon premier programme**.

Les lettres sont toutes petites , en cliquant sur la ligne **Font** une fenêtre s'ouvre permettant de choisir le type de caractères, par exemple ARIAL , la taille , par exemple 24 , la couleur (conservons le noir). Nous ne modifierons pas la propriété Color pour que la fenêtre aie la même couleur que le fond .. .

Il est possible de déplacer le titre à la souris pour l'amener à l'emplacement le meilleur.

.L'écran est alors représenté ci contre.

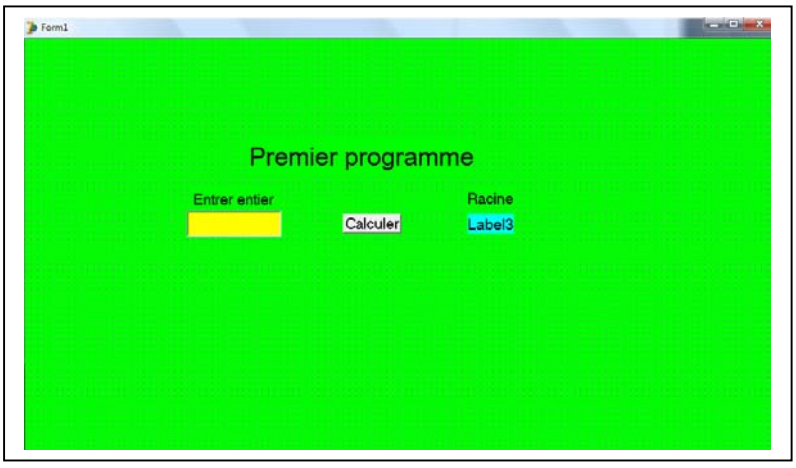
Maintenant plaçons la fenêtre dans laquelle nous écrirons le nombre entier dont nous voulons calculer la racine carrée. C'est un objet de saisie **ab** désigné par Delphi comme Edit . Cliquons donc sur ab dans la ligne des objets pour l'activer et le placer ensuite sur la fiche à gauche en dessous du titre . Un rectangle blanc apparaît avec le titre Edit1 . Dans la fenêtre de gauche définissons ses propriétés . Sa couleur **Color= (clYellow)** Jaune, il ne faut pas modifier son nom (Name) mais effacer le texte en changeant caption= **' ' (** en pressant sur la barre d'espace**)** Choisir ensuite la taille et fonte des caractères dans **Font** (Arial 14, noir)

Si la propriété **AutoSize** est vraie (true) la taille de la fenêtre de saisie s'ajustera automatiquement à celle du nombre que l'on écrit. Pour la conserver de taille fixe (ajustée à la souris) il vaut mieux choisir **AutoSize=False** (fausse)

Au départ du programme cette fenêtre sera vierge de tout nombre, pour guider l'utilisateur nous allons lui associer une étiquette **Entrer entier**. Dans une petite fenêtre (objet A) placée juste au dessus. Pour cela la procédure est la même que celle qui nous a permis de placer le titre , l'objet A introduit prend automatiquement le nom label2

Le résultat du calcul sera présenté dans un troisième objet A (label 3) pour lequel nous fixerons la couleur bleue **color = clAqua** puis la fonte des caractères (arial 14 noirs) .Le mot label3 qui est placé par défaut sera comme précédemment effacer dans **caption** , attention ne pas toucher à **Name**. .Cette fois il vaut mieux que le largeur de la fenêtre soit ajustée automatiquement à la taille du résultat nous ferons donc **AutoSize=True** ;

. Comme plus haut pour guider l'utilisateur plaçons une étiquette **racine** au dessus de l'objet précédent (label4)



Pour finir il nous faut un bouton sur lequel appuyer pour lancer le calcul. C'est un objet **Button** que nous irons chercher dans la ligne des objets et placerons sur la fiche entre les fenêtres précédentes. Ce bouton n'a pas de couleur mais changeons **caption = calcul** Comme d'habitude position et taille peuvent être ajustés avec la souris.

La fiche écran Form1 a maintenant l'allure ci dessus .

Seconde partie du travail écrire le programme PASCAL accompagnant les objets et leurs fonctions. Mais avant examinons la fenêtre du second plan **UNIT 1** en cliquant dessus (si elle est en partie visible ou **Projets⇒Voir⇒Unites ⇒premier)**

Unit1 premier

```
unit premier;

interface

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

type
  TForm1 = class(TForm)
    Label1: TLabel;
    Edit1: TEdit;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Button1: TButton;
  private
    { Déclarations privées }
  public
    { Déclarations publiques }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

end.
```

Il s'agit d'une routine de type unit avec son nom **premier** choisi au début .

Dans le listing présenté deux mots ont un rôle essentiel : **interface** et **implementation**. Ils définissent deux zones , la première entre ces deux mots , la seconde entre la directive de compilation {$R *.dmp} et le mot **fin**.

La première zone est descriptive, on trouve d'abord le terme **uses** suivi de la liste des unités utilisées par le programme PASCAL , cette ligne est la même que dans un programme PASCAL sous DOS. Puis **Type** ou sont annoncés les types des différents objets introduits dans la construction de l'écran. 4 Labels 1 Edit 1

Button.,puis les variables var ici Form1:Tform indiquant que l'objet Form1 (la fiche principale) est de type Tform.

Derrière les mots **Private** et **Public** il est possible de définir des variables . Celles définies dans **private** ne seront accessibles que par les procédures ou fonctions appartenant à la fiche Form1 , celles dans **public** sont accessibles pour toutes les fiches d'un programme s'il en comporte plusieurs , par contre elles ne sont pas reconnues dans des procédures indépendantes (dont le nom ne commence pas par TFOrm1. (voir ci dessous) .

Exemple :
Soit l'unité suivante, la fiche s'appelle Tform1
(ne figurent ici que les parties concernées par le problème).

```
Private
        A :integer ;

Implementation
{$R*.dfm}
Var
        C :integer ;

Procedure  Agmente (b:integer);
Begin;
        A:=4;
        B:=B+1;
        C:=3;
End;

Procedure  Tform1.FormCreate(Sender:Tobject);
Begin
        C :=C+1 ;
        A :=56 ;
        Augmente (A);
End;
```

La variable A est définie dans **Private** donc dans **Tform1**, elle est reconnue par la procédure **Tform1.FormCreate** mais non dans la procédure **Augmente** et le compilateur annonce une erreur (ligne en rouge qu'il faudra supprimer). Par contre C est une variable totalement globale elle est connue aussi bien d'une procédure que de l'autre. Tant qu'a B elle est interne à la procédure **Augmente** et ne sert que d'intermédiaire, la dernière ligne **Augmente (A)** envoie 56+1 dans A.

Dernière remarque :

La procédure Sauve utilisée par Tform1 doit être définie avant cette dernière .

Revenons à notre programme :

Dans la seconde zone avant le end final il n'y a rien pour le moment il faut y introduire les informations sur le rôle du bouton. Dans la fiche cliquons donc sur le bouton .Il apparaît alors à l'écran la partie de l'unité concernant ce composant.

L

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

end;

end.
```

Il s'agit d'une **procedure** ayant pour nom **Tform1.Button1Click(Sender :Tobjet)** ;

Tform1 pour indiquer que l'on est dans la fiche portant ce nom (la seule du programme dans le cas présent**)**

Button1Click car l'événement est un click sur le bouton de nom button1 . Entre parenthèse est indiqué que cet objet est un descendant du type de base Tobject.

Lorsque l'on presse le bouton (c'est l'événement déclenchant) le nombre entier qui a été placé dans Edit1 doit être lu , sa racine carrée calculée et le résultat basculé dans la fenêtre de sortie Label3.

Soit :

Par exemple :

Procedure Tform1.Button1Click(Sender :Tobject) ;

Var:a :integer ; // définition d'un entier a

Var b :real ;/ // définition d'un réel car la racine est un réel

Begin

a := StrToInt(Edit1.text) ; *{ le nombre a est le terme Text de l'objet Edit1, mais il s'agit d'une chaîne de caractères car un Edit ne peut recevoir que des caractères, la fonction StrToInt (lire Str To Int) transforme la chaine de caractères en un nombre entier. }*

b :=sqrt(a) ; { *extraction de la racine , si l'opération demandée était plus complexe il faudrait utiliser ici une fonction ou procédure écrite spécialement* }

Label3.Caption :=FloatToStr(b) ; *{ b est un réel (flottant) , que la fonction Float To Str transforme en chaîne de caractères compatible avec l'objet Label }*

End ;

Si on examine maintenant l'unité complète on pourra constater qu'une ligne supplémentaire à été insérée dans la première zone

Procedure Button1Click(Sender :Tobject) ;

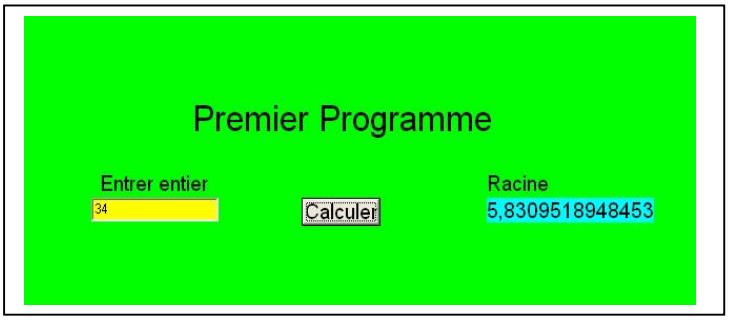
Il s'agit de l'annonce dans cette première zone de la procédure qui est écrite dans la seconde. C'est nécessaire si le programme ne le fait pas automatiquement le programmeur devra le faire lui même.

```
unit premier;

interface

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

type
  TForm1 = class(TForm)
    Label1: TLabel;
    Edit1: TEdit;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  private
    { Déclarations privées }
  public
    { Déclarations publiques }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
Var a:integer;
Var b:real;
begin
 a:=StrToInt(Edit1.Text);
 b:=sqrt(a);
 Labe3.Caption:=FloatToStr(b);
end;

end.
```

Maintenant sauver votre travail comme plus haut et lancer l'exécution , soit par la commande exécuter de la première ligne soit F9. Si une erreur intervient lors de la compilation elle est indiquée et vous la corrigez.

Ci contre l'écran final après avoir introduit le nombre 34 et pressé le bouton **Calculer** .

Remarquer le dimensionnement automatique de la fenêtre de sortie.



Le projet fonctionne , pour créer le fichier exécutable .exe il suffit de

lancer la commande de la première ligne :
Projet ⇒Compiler Projet ou au clavier **ctrl F9**

Remarques sur ce premier programme

Ce programme nous a permis d'utiliser les objets les plus importants , Zone de Texte **((Label**) de saisie (**Edit**) et boutons **(button**) .Nous avons vu comment en cliquant dessus on fait apparaître le logiciel PASCAL qui leur est associé. Le programme le plus important est celui qui est associé au bouton , on y trouve les fonctions et procédures nécessaires pour exécuter les tâches souhaitées , il peut lire ou modifier les propriétés des autres objets, **caption, text**, mais aussi les tailles couleurs ou visibilité. etc..

Nous avons noté que le contenu des fenêtres à l'écran est de type texte ce qui oblige à des conversions .Dans la liste des instructions du PASCAL OBJET cette conversion est assurée par les instructions suivantes.

StrToInt de chaîne de caractère en entier
IntToStr d'entier en chaîne de caractère
FloatToStr Un nombre en virgule flottante est transcodé en chaîne de caractère
FloatTOsrtF effectue la même action mais autorise un formatage précis du résultat
StrToFloat l'inverse

DELPHI autorise aussi une conversion décimale vers Hexa
DecToHex et **HexToDec**

Note sur la structure de l'unité.

Le programme précédent effectue une simple extraction de racine qui est réalisée par la seule instruction sqrt . Si l'opération demandée est plus complexe il est nécessaire de l'effectuer soit par une fonction soit une procédure. Par exemple pour calculer le prix TTC à partir du prix hors taxe, il faut multiplier par 1,196 .

L'écran est le même que précédemment mais la fenêtre de gauche affiche le résultat du calcul c'est à dire le prix TTC , le hors taxe étant entré à gauche. Attention ces deux chiffres sont des réels (real) .

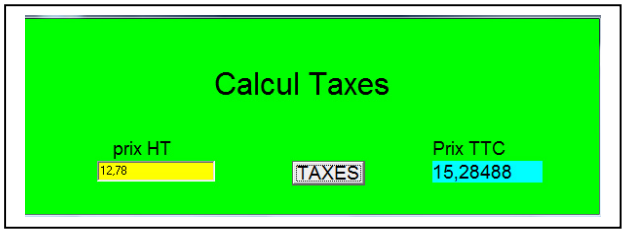
Derrière la directive de compilation :

```
Function  AvecTaxes(p :real ) :real ;
Begin
AvecTaxes(a):=p*1.196;
End;
```

Et la routine associée au bouton est:

```
Procedure Tform1.Button1Click(Sender :Tobjet) ;
Var  a,b:real;
Begin
        a:=StrToReal(Edit1.Text);
        b:=AvecTaxes(a);
```

```
        Label3.Caption:=FloatToStr(b);
End;
```



Remarque:*le séparateur de la partie décimale est un point dans la définition de la fonction et une virgule à la saisie car la version de Delphi est française .*

Le même résultat aurait été obtenu en utilisant une procédure

```
Procedure   LesTaxes(a :real;var b:real);
Begin
        b:=a*1.169;
End;
```

Mais on peut être tenté d'entrer dans cette procédure la saisie du prix dans la fenêtre d'introduction , c'est à dire :

```
Procedure  LesTaxes(var b :real) ;
Var a:real;
Begin
a:=strToFloat(Edit1.text);
b:=a*1,179;
End;
```

Et cette fois le compilateur détecte une erreur à la première ligne , en effet la procédure LesTaxes est complètement indépendante de Form1 et de ses composants, elle ignore l'existence de Form1. I Il faut faire une ***déclaration anticipée*** (prototype de la procédure) au début du programme , puis à l'appel ultérieur faire précéder le titre du terme Form1.
Soit :
Au dessus de la ligne Private :

```
Procedure LesTaxes(var b :real) ;
```

Et ensuite après la directive de compilation: :

```
Procedure  Form1.LesTaxes(var b :real) ;
Var a:real;
Begin
a:=strToFloat(Edit1.text);
b:=a*1,196;
End;
```

Et la routine bouton qui se réduit à :

```
Procedure Tform1.Button1Click(Sender :Tobjet) ;
        Var  a:real;
Begin
        LesTaxes(a);
```

```
        Label3.Caption:=FloatToStr(a);
End;
```

Exemple d'objets importants

Il existe un nombre considérable d'objets et il est hors question de les passer tous en revue ici, nous en citerons seulement quelques uns

Bouton BitMap BitBtn

C'est un objet de **supplément** Il joue le même rôle que l'objet bouton mais peut contenir une image . Les propriétés essentielles sont:
Caption //Le texte apparaissant sur le bouton
Height et **Width** //Hauteur et largeur du bouton en pixels
Left et **Top** //position du bouton sur la fiche l'origine étant en haut à gauche .
Mais surtout
Glyph //Une fenêtre s'ouvre demandant la position de l'image à utiliser. Attention cette image doit avoir une taille compatible avec les paramètres précédents, si elle est plus grande elle sera tronquée. Seuls les fichiers .**bmp** sont admis.
Visible // vrai ou faux visible ou non /

. Boutons Radio 

Ce sont des boutons qui peuvent se trouver dans deux états enfoncés ou non .La propriété correspondante est **Checked** (True ou **False**). , comme les précédents ils peuvent avoir une dénomination **Caption** , une couleur etc… mais surtout ils fonctionnent par groupe , un seul bouton peut être enfoncé à la fois (comme les boutons poussoirs de choix de gamme d'onde d'un poste de radio d'ou leur nom.) A la création de la fiche l'un des boutons peut être activé en faisant Checked=True ; (mais un seul) .

Ces boutons peuvent être groupés dans un bloc **GroupBox** de la palette standard. (il faut ouvrir sur la fiche une fenêtre groupbox et placer les boutons dedans). Des boutons dans deux groupes différents sont indépendants. Pour grouper plusieurs RadioBoutons il est également possible , sans groupbox , de donner à tous les éléments que l'on veut grouper la même valeur de la propriété **GroupIndex**

Dans un groupe si la propriété **AllowAllUp** est vraie (True) il est possible de mettre tous les boutons dans l'état off en cliquant sur le bouton enfoncé.

CheckBox ⊗

C'est une simple case à cocher , l'état est déterminé par la propriété **Checked**

La ListBox

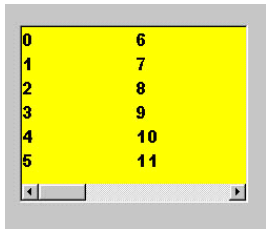
C'est une extension de Edit à plusieurs lignes. Elle est disponible parmi les objets standards ***Ses propriétés:***
Color la couleur de fond

Font la fonte des caractères comme pour Edit ou Label
Clear pour l'effacer
Column le nombre de colonnes
Mais surtout la procedure **Items.Add** qui ajoute une ligne après la dernière ligne présente Ce doit être une chaîne de caractères :

Liste1.Items.Add (' ceci est une nouvelle ligne ');

Par exemple avec une ListBox et un bouton En choisissant column=2 l'affichage des nombres de 0 à 20 s'obtient avec :

```
Procedure TForm1.Button1Click(Sender: TObject);
Var i:integer;
Begin
        ListBox1(Clear);              // effacement préalable
        For I;=0 to 20 do
                ListBox1.Items.Add(intTOstr(i));
End;
```



Compte tenu de la taille de la fenêtre tracée au départ et de la fonte choisie, les nombres de 0 à 20 ne tiennent pas sur 2 colonnes , apparaît alors un curseur permettant un balayage horizontal, il n'y a toujours que 2 colonnes dans la fenêtre.

Si column est mis à 0 il y a une seule colonne avec curseur vertical, pour column=1 le curseur est horizontal .

CheckListBox (TlistBox)

C'est une liste comme la précédente mais on peut cocher chaque ligne.
Exemple :

Plaçons sur une fiche un bouton (Button1) une case de sortie (Label1) et une **CheckListBox** (CheckListBox1). Une fenêtre apparaît sur la fiche , dans l'inspecteur d'objets effectuons un double click sur la droite de la propriété **Items** (ou un seul click sur les points de suspension) , l'éditeur de liste de chaînes s'ouvre. Taper la première ligne puis après un retour chariot la seconde etc… Lorsque toutes les lignes sont entrées faire OK. Redimentionner la fenêtre et modifier au besoin (Font) la taille et forme des caractères. Noter le petit carré situé à gauche de chaque ligne, ce sont les zones de sélection.

Cliquer alors deux fois sur le bouton et dans la routine Button1Click ajouter la ligne :

Label1.Caption :=IntToSrt(CheckListBox1.ItemIndex) ;

Sauver et lancer l'exécution.

Dans la liste cliquer alors la zone de sélection de l'une des lignes puis sur button1. Le numéro de la ligne choisie s'affiche dans Label1. On notera que la première ligne est la ligne 0 et non 1.

Si plusieurs lignes ont été sélectionnées c'est le rang de la dernière à l'être qui est affiché. .

Le Timer

Fait partie des objets **Système**..

Ce n'est pas un objet graphique, il apparaît sur la fiche comme une petite horloge mais ne figurera pas dans la présentation finale à l'écran . Cet objet est un compteur interne qui délivre à des intervalles réguliers un signal déclenchant une action choisie par le programmeur.

Dans l'Inspecteur d'Objets il n'y a que 4 propriétés:

Enabled // Si True (vrai) le compteur interne est lancé sinon il est inhibé.

Intervalle **//**C'est en millisecondes le temps qui s'écoule entre deux événements déclenchants

Name //comme d'habitude le Nom par défaut Timern (n=1 2 3…)

Tag // est simplement un entier associé au Timer à la disposition du programmeur.

Le seul événement proposé est **OnTimer** c'est l'impulsion déclenchante citée plus haut.

A titre d'exemple le programme suivant met en œuvre 2 boutons permettant de lancer un compteur à une vitesse définie dans la fenêtre Edit1. L'intervalle est introduit dans Edit1 et en pressant le bouton les nombres successifs défilent dans Label1 .

Les objets sont sélectionnés et placés sur la fiche comme plus haut;.On définira :

Timer1 **Enabled** =False (il n'est pas lancé au départ)
Intervalle =100 (Dix chiffres par seconde par défaut)
Label1 (ou seront affichés les nombres) **color** clYellow (jaune)
Label1.Caption ' ' // effacement
AutoSize= False // pour que la fenêtre reste de largeur constante
Label2 (le commentaire intervalle) caption =intervalle
Font Arial Black Noir 12

Button1
Caption =Lancer
Button2
Caption= stop
Edit1 (ou sera introduite la valeur de la période)
Text : =100; // valeur par défaut 100mS . Si on avait placé au début un texte, par exemple Intervalle il y aurait un risque d'erreur si l'utilisateur appuyait sur le bouton avant d'y entrer un entier.

Un entier a est déclaré dans la zone Private .

Le programme associé à Button1 Entre Begin et end

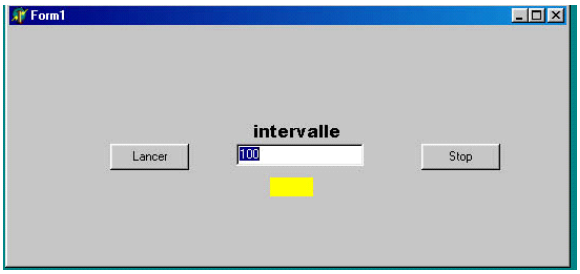
```
a:=0            // initialisation de a
 Timer1.Enabled:=True; // le timer est lancé)
Timer1.Interval:= strTOint(Edit1.Text) ; // lecture de l'intervalle
```

Le programme associé à Button2 , entre Begin et End;

```
Timer1.Enabled:=False          // stop timer
Label1.Caption:=' '              // effacement du contenu de Label1.
```

Le programme associé au timer

```
a:=a+1;                  // à chaque événement a est  incrémenté
Label1.Caption:=intTOstr(a);       // affichage du nombre
```



Ce projet est sauvé sous les noms de Horloge et pr_Horloge .

Le programme:

```
unit Horloge;

interface

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

type
  TForm1 = class(TForm)
    Timer1: TTimer;
    Edit1: TEdit;
    Button1: TButton;
    Button2: TButton;
    Label1: TLabel;
    Label2: TLabel;
    procedure Timer1Timer(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Déclarations privées }
    a:integer;
  public
    { Déclarations publiques }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.Timer1Timer(Sender: TObject);
begin
```

```
a:=a+1;
Label1.Caption:= intTOstr(a)
end;

procedure TForm1.Button1Click(Sender: TObject);
var i:integer;
begin
a:=0;
i:=strTOint(Edit1.Text);
Timer1.interval:=i;
Timer1.enabled:=True;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
Timer1.enabled:=False;
Label1.caption:=' ';
end;

end.
```

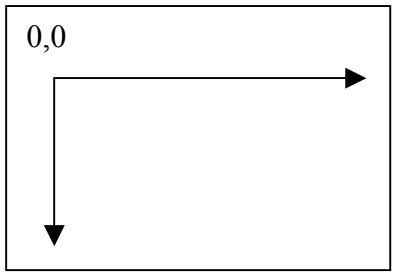
Le Timer peut être utilisé pour réaliser un retard unique par exemple :

```
procedure TForm1.Button1Click(Sender: TObject);
var i:integer;
begin
edit1.color:=clTeal;// lorsque l'on presse le bouton Edit1 devient verte et
le                  //Timer :est activé
Timer1.Enabled:=true;

end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
edit1.color:=clred;        // à l'evênement Timer suivant EDIT1 passe au
                           //rouge et le Timer est stoppé .
Timer1.Enabled:=False;
end;
```

L'objet TCANVAS (canevas)

Le mot **Canvas** (en français canevas) désigne une propriété d'un objet mais c'est aussi un objet. Les canvas sont uniquement disponibles en phase d'exécution c'est pourquoi ils ne figurent pas dans l'inspecteur d'objets .Certains objets seulement possèdent un canvas, les fiches **Form ,Image, Paintbox Grilles**, Le canevas utilise un crayon pour tracer des lignes et les contours, un pinceau pour peindre les surfaces, une fonte pour écrire .On utilise crayon et pinceau pour définir les caractéristiques du dessin que l'on trace : couleur, grosseur des traits .

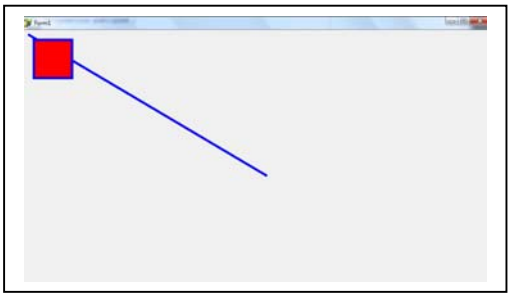
L'outil canvas est la clé de tous les tracés graphiques :Dans la zone concernée le point 0,0 est le coin haut gauche, l'abscisse x augmente vers la droite et l'ordonnée y vers le bas .



La commande **MoveTo (x,y)** positionne le crayon au point de coordonnées x,y
La commande **LineTo(x,y)** trace une ligne de la position précédente du crayon jusqu'au point x,y et repositionne le crayon en ce point.
Rectangle(x1,y1,x2,y2), dessine un rectangle ayant un coin supérieur gauche en x1,y1 et coin inférieur droit en x2,y2. **RoundRect(x1,y1,x2,y2)** donnerait des coins arrondis
De même **Ellipse(x1,y1,x2,y2)** trace une ellipse circonscrite à l'intérieur du rectangle défini par les coordonnées
Enfin **TextOut(x,y,text)** affiche le texte aux coordonnées précisées.
Tous ces tracès sont effectués avec les propriétés du crayon (**pen**) , qui sont **Color, Style** et **Width.** Le pinceau (**Brush**) possède les propriétés **Color** et **Style** , cette dernière définit le motif d'arrière plan.
Exemple 1
Ouvrons un nouveau projet avec seulement la fiche Form1
En cliquant deux fois dessus nous avons accès à la procédure associée Tform1.FormCreate que nous compléterons comme ci dessous. :

```
procedure TForm1.FormCreate(Sender: TObject);
begin
Form1.Canvas.Pen.Color:=clBlue;
Form1.Canvas.Brush.Color:=clRed;
Form1.Canvas.Pen.Width:=5;
Form1.Canvas.MoveTo(10,10);
Form1.Canvas.LineTo(500,300);
Form1.Canvas.Rectangle(20,20,100,100);
end;
```

La première ligne définit la couleur du crayon, la seconde celle du pinceau, la troisième la taille du trait de crayon (5 pixels, par défaut elle serait de 1).La quatrième ligne positionne le crayon en x=10,y=10 la suivante trace un trait (bleu de largeur 5) de ce point au point 500,300.La dernière ligne dessine un rectangle de fond rouge et bords bleus(de largeur 5).



Si on demande l'exécution aucune erreur n'est signalée mais la fiche qui apparaît est vierge. Il faut déclencher son tracé au moment de sa création en choisissant dans l'inspecteur d'objets de cette fiche **onPaint=onCreate** (La procédure à bien pour titre FormCreate)

Il est possible d'alléger l'écriture de la procédure en utilisant la commande : **With Canvas do** derrière le mot with Form1 n'est pas nécessaire car il n'y a aucune ambiguïté, **canvas. do** aurait suffit.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
with Form1.canvas do
  begin
    Pen.Color:=clBlue;
    Brush.Color:=clRed;
    Pen.Width:=5;
    MoveTo(10,10);
    LineTo(500,300);
    Rectangle(20,20,100,100);
  end;
end;
```

Exemple 2 :

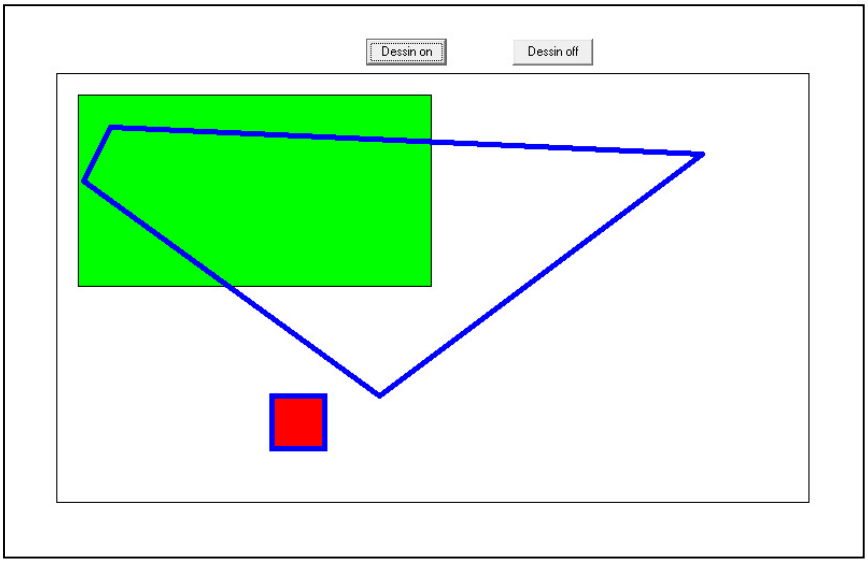
Nous utilisons cette fois un objet **PaintBox** dont nous fixons dans l'inspecteur d'objets la taille **Height**=400, **Width**=700. et la couleur **color=clLime** (verte).

Cette fois nous introduisons un bouton **button1** sur lequel nous cliquerons pour effectuer le dessin. Et un autre **button2** pour l'effacer.

Cliquer deux fois sur le premier bouton donne accès à la procédure associée., soit :

```
procedure TForm1.Button1Click(Sender: TObject);
var h,v:integer;
begin
With PaintBox1.Canvas do
  Begin
    MoveTo(0,0);
    h:=PaintBox1.Width-1;
    v:=PaintBox1.Height-1;
    LineTo(h ,0);
    LineTo(h ,v);
    LineTo(0,v);
    LineTo(0,0);
    Rectangle(20,20,h div 2,v div 2);
    Brush.Color:=clRed;
    Pen.Width:=5;
    Pen.Color:=clBlue;
    Rectangle( 200,300,250,350);
    PolyLine([Point(50,50),Point(600,75),Point(300,300),Point(25,100),
              Point(50,50)]);
  end;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  PaintBox1.Visible:=false;
end;
```

En lançant l'exécution on obtient l'écran suivant :



Au départ la taille du crayon n'a pas été définie elle est donc par défaut 1 pixel , c'est ainsi que sont tracés l'encadrement par les instructions Line, et l'encadrement du premier rectangle. On notera pour ce dernier qu'il prend par défaut la couleur de l'objet PaintBox1 .Le rectangle suivant ainsi que le tracé polynomial utilise un crayon bleu de largeur 5 et un remplissage rouge.

Attention lorsque l'on presse le bouton **off** le dessin disparaît mais si on active de nouveau le bouton **on** le tracé commence avec un trait bleu large et un fond rouge car les instructions précédentes ont été conservées. Pour l'éviter il faudrait réinitialiser pinceau et crayon dans la procédure du bouton off.

Tel qu'il est le programme présente un défaut , au premier appui sur le bouton on le dessin apparaît, il disparaît en pressant off, puis apparaît de nouveau avec on mais ensuite le bouton off n'agit plus. Il faut pour que tout rentre dans l'ordre ajouter une commande **Refresh** à la fin de la routine off pour réinitialiser le composant de la fiche.

```
Procedure  Tform1.Button2Click(sender :Tobjet) ;
Begin
  PaintBox1.Visible:=false;
  Refresh;
End;
```

Exemple 3

Tracé d'une courbe y=f(x).

La courbe est tracée dans un cadre de 512 points de large et 500 de haut. Ce cadre est tracé dans un objet PaintBox .
Trois boutons sont utilisés :

Button1 : ***Cadre*** Trace les côtés du cadre et les axes

La Procedure **Button1Click** contient les commandes canvas de l'objet Paintbox1

Button2 ***Diagonale*** Trace une diagonale sur le repère précédent

La procédure **Button2Click** appelle la procédure ***Diagonale*** qui charge dans le tableau **MonEcran** (qui a été crée comme une variable globale dans les premières lignes derrière la directive de compilation) les 512 ordonnées du tracé de

cette droite, puis appelle la procédure **TraceEcran** qui utilise ce tableau pour effectuer le tracé dans le Canvas de PaintBox1

Button3 ***Courbe*** Trace la courbe. ,(dans cet exemple une sinusoïde.)

La procédure **Button3Click** fonctionne de la même façon que la précédente Elle appelle la procédure **Courbe** qui charge dans **MonEcran** les points de la courbe puis appelle la routine **TraceEcran**.

Il faut noter :

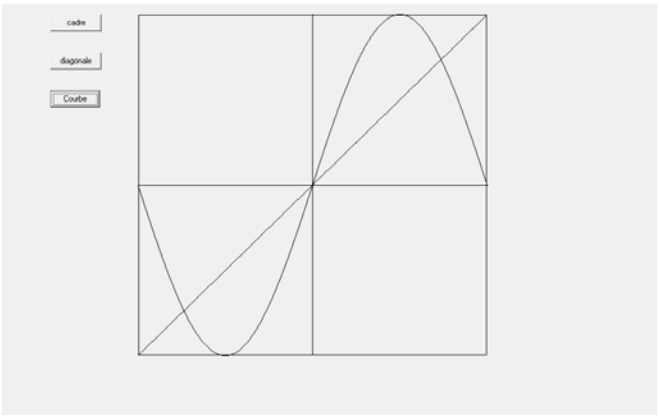
MonEcran est une variable globale utilisable par toute procédure du programme qu'elle soit de la **classe Form1** ou non.

La procédure **TraceEcran** utilise le **Canvas de Paintbox1** , elle doit donc être déclarée dans **Form1** et pour cela déclarée dans la partie interface de l'unité. et appelée avec son nom **Tform1.TraceEcran**.

Les procédures ***Diagonale*** et ***Courbes*** ne sont pas de la classe **Form1** mais peuvent être appelées à partir de cette classe.

Les procédures ***Diagonale*** et Courbe doivent dans le listing apparaître avant les procédures **Button2Click** et **Button3Click** qui les appellent sinon elles ne seraient pas reconnues.

```
unit Graph;

interface

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

type
  TForm1 = class(TForm)
    PaintBox1: TPaintBox;
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    procedure Button1Click(Sender: TObject);
    procedure TraceEcran;
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
  private
    { Déclarations privées }

  public
    { Déclarations publiques }

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}
Type
Ecran = ARRAY[1..512] of integer;
var
MonEcran:Ecran;  { définie comme variable globale }
```



```
procedure TForm1.Button1Click(Sender: TObject);{ tracé du cadre et axes}
begin
  With PaintBox1.Canvas do
  Begin
    Rectangle(1,1,512,500);
    MoveTo(0,250);
    LineTo(514,250);
    MoveTo(256,0);
    LineTo(256,500);
  End;
end;

Procedure Diagonale; {Rempli le tableau  MonEcran avec les points
de la diagonale et appelle la routine de tracé }
Var
i:integer;
Begin
For i:=1 to 512 do
    Begin
        MonEcran[i]:=500-Round(i*500/512);
     End;
Form1.TraceEcran;
End;

Procedure Courbe;{Rempli le tableau  MonEcran avec les valeurs de la
 courbe et appelle la routine de tracé}
Var
i:integer;
Begin
For i:=1 to 512 do
    Begin
        MonEcran[i]:=Round(250+250*sin(2*Pi*i/512));
     End;
Form1.TraceEcran;
End;

Procedure Tform1.traceEcran; { trace l'écran avec les valeurs du tableau en
utilisant une propriété Canvas , elle doit donc être  définie dans la
classe TForm1}
Var
i:integer;
Begin
    With PaintBox1.Canvas do
      Begin
        MoveTo(0,MonEcran[1]);{ accès à la variable globale  MonEcran}
        For i:=2 to 512 do
              Begin
                  LineTo(i,MonEcran[i]);
              End;
      end;
End;

procedure TForm1.Button2Click(Sender: TObject);
begin
Diagonale;
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
Courbe;
```

```
end;

End.
```

Gestion des erreurs .

Un programme peut s'interrompre et même bloquer l'ordinateur si une instruction ne peut pas s'effectuer et génère une erreur. C'est le cas classique de la division par zéro, de dépassement de capacité (par exemple écrire à la position n dans une chaîne de caractères de longueur n-1) etc.. mais aussi erreur de l'utilisateur qui entre une lettre alors que le logiciel attend un chiffre ou un point au lieu d'une virgule. La protection vis à vis de ces erreurs est parfois difficile car on ne peut pas toujours imaginer l'erreur que commettra l'utilisateur.

Exemple 1 :

Protection vis à vis d'une entrée dans un objet Edit qui attend un entier.
C'est le cas le plus simple.
L'instruction habituelle est dans ce cas a:=strToint(Edit1.text) ; mais si l'utilisateur a tapé 45RT la conversion ne peut se faire et le programme se plante.
Le plus simple est d'utiliser une autre instruction de conversion :, la procédure val

Val(s:chaîne,Var a,err:integer);
S est la chaîne de caractère lue sur Edit1 , a l'entier correspondant et err un entier qui vaut 0 s'il y a erreur.

Soit par exemple une fiche avec un bouton, une fenêtre d'entrée Edit1 et de sortie Label1. La routine associée au bouton est :

```
 Procédure  Tform1.Button1Click(Sender :Tobjet) ;
Var
Err,a:integer;
Begin
  Val(Edit1.text,a,err);
If err<>0 then  Label1.Caption:='erreur d''exécution' else Label1.caption:=Edit1.text;
End
```

Si l'on tape 15 dans la fenêtre d'entrée et presse le bouton, 15 apparaît dans la fenêtre de sortie , mais si l'on tape 15,8 ou ZERT56 Label1 affiche *erreur de saisie* sans que le programme ne se plante , il est possible de continuer sans problème avec un entier valide.

Cette méthode ne fonctionne hélas pas avec un réel mais reste utilisable si le nombre attendu est un entier et que l'opérateur entre par erreur un nombre avec virgule et décimales ou une lettre . En effet si :

La fiche comporte une fenêtre de saisie Edit1 , de sortie Label1 et un bouton
Le nombre N est déclaré comme un réel ou un entier
Et l'on exécute l'instruction Val(Edit1.text,N,err) ;
Puis Label1.caption :=floatTostr(N) ;

Tapé dans Label1	N lu et affiché dans Label1	Err
4,67	4	2
5	5	0
Xyz45	0	1

L'instruction val a évité une erreur fatale qui aurait arrêté le programme mais la valeur N saisie est dans tous les cas un entier , le mot err permet de distinguer une entrée de type réel (err 2) ou de type caractère (err = 1) ce qui peut être bien utile si pour indiquer à l'opérateur quelle est l'erreur qu'il a commise.

Exemple 2

Contrôle de tous les caractères entrés dans Edit.

L'idée est de vérifier que tous les caractères entrés sont des nombres ou une virgule. On utilise pour cela plusieurs fonctions de gestion des chaînes de caractères.

Chn :=Copy(Chaine1,position,largeur) ; qui fournit une chaîne chn qui est un morceau de la chaine1 de largeur défini. Par exemple

Chn :=copy((Edit1.text),i,1) ; délivre une chaîne de largeur 1 à partir de la position i

Si Edit1.text= 9527 Copy ((Edit1.text),3,1) fourni la chaîne (qui se réduit dans ce cas à un seul caractère) '2'.

Cette fonction permet donc d'explorer le nombre entré caractère par caractère.

La seconde fonction est :

Pos(Chaine1,ChaineReference) ; délivre un entier qui est le rang à partir duquel chaine1 est présente dans ChaîneReference. Cet entier est nul si rien n'est trouvé.

Par exemple **position :=Copy(chn,'0123456789,')** lorsque chn est un caractère fourni par la fonction précédente , délivre 0 si le caractère chn n'est ni un chiffre ni une virgule c'est à dire qu'il y a erreur de donnée entrée.

Soit par exemple une fiche comportant une fenêtre d'entrée Edit1, deux fenêtres de sortie Label1 et Label2, et un bouton button1. On veut que lorsque le bouton est pressé le nombre entré sur Edit1 apparaisse dans Label1 , Label2 restant vide. Si par contre il y a une erreur de saisie la fenêtre Label2 affiche **erreur** et la fenêtre d'entrée est vidée.

La routine associée au click sur le bouton se contente d'appeler la procédure de contrôle de l'entrée.

```
Procedure Tform1Click(Sender :Tobject) ;
Begin
   ControlEdit;
End;
```

Cette dernière fait appel aux objets de Form1 elle doit donc être déclarée dans le paragraphe Type au début de l'unité.

```
Procedure Tform1.ControlEdit ;
Var
Chn :string
Position,lg,err,i :integer ;
```

```
Begin
   Lg :=length(Edit1.Text) ;        La longueur de la chaine tapée dans Edit1
   Err :=0 ;                        Initialisation  du  mot d'erreur
   For i := 1 to lg do              Pour balayer tous les caractères entrés
      Begin
            Chn :=copy((Edit1.text),i,1) ;     isolement d'un caractère
            Position:=Pos(chn,'0123456789,');      ; position du caractère dans la
liste des  caractères valides.
            If position=0 then  err :=1 ;        err mis à 1 si erreur trouvée
      End
            If err=1 then
            Begin         Si  il y a erreur  Edit1 effacé  erreur affichée
                  Label2.Caption :=' erreur de saisie' ;
                  Edit1.clear;          Effacer le texte d'une fenêtre de saisie
                  Label1.caption:='';         Il n'existe pas de fonction clear ici
            End
            Else          Si non erreur  EDIT1 transféré dans Label1  Label2 vidé.
                  Begin
                        Label2.caption :='' ;
                        Label1.caption :=Edit1.text ;
                  End ;
End ;
```

On peut cependant remarquer qu'aucune erreur n'est détectée si l'opérateur a tapé deux fois la virgule. Il n'est pas difficile d'y remédier par exemple en mettant à 1 une variable **Virgule** lors de la première détection de ce caractère puis de l'incrémenter à chaque nouvel entrée de , et en fin de boucle afficher erreur si **virgule** est supérieur à 1 .

Notes

.**Note 1 . : Sauvetage d'un projet.**

Soit un projet enregistré initialement sous les noms UnA pour les unités et PrA pour le projet on trouve sur le disque dans le répertoire Projets les fichiers :
UnA.dcu UnA.dfm UnA.pas et
PrA.cfg PrA.dof PrA.dpr PrA.exe PrA.res

Si l'on modifie alors le projet en lui ajoutant de nouvelles fonctions et que après l'avoir vérifié et lancé on souhaite le sauver sous un autre nom par exemple PrB grâce à la commande **sauver le projet sous** , on trouve dans le répertoire projet les 4 fichiers suivants avec le nouveau nom :
. PrB.cfg PrB.dof PrB.dpr PrB.exe PrB .res
mais aussi :
UnA.~dfm UnA.~pas **UnA.dfm UnA.pas UnA.dcu** les deux premiers dont le suffixe commence par ~ sont les deux fichiers dfm et pas précédents , deux nouveaux ont été crées mais le nom de l'unité n'a pas été changé.
Si l'on veut enregistrer le travail à ce niveau il faut copier dans un autre répertoire ou une clé USB les 8 fichiers en caractères gras.

A chaque fois qu'au cours du travail on sauve le projet les fichiers dfm et pas sont recrées et les anciens conservés avec le suffixe en ~, *mais attention seuls les fichiers de l'état précédent sont sauvés , les plus anciens sont perdus*.

. Si de multiples versions successives sont crées PrA PrB PrC etc ;;;et si l'on n'a pas pris la précaution de sauver la totalité des fichiers à chaque fois il se peut qu'en essayant de lancer PrB par exemple une erreur se manifeste puisque le fichier de configuration de cette étape B a été écrasé.

Pour aller plus loin.

Jouer avec la souris :

Delphi exploite entièrement les routines de Windows pour gérer les déplacements et actions de la souris.

1° La procédure **GetCursorPos(pt) ;**

C'est une procédure issue directement de Windows elle permet de saisir les coordonnées X,Y du point au dessus duquel se trouve la souris. Attention il s'agit de la position sur l'écran de l'ordinateur et non relative à une fenêtre dessinée par Delphi. .Par exemple :

Soit un projet comprenant seulement une zone d'affichage Label1 sur une fiche Form1. Dans l'inspecteur d'objets associé à Form1 ouvrir la colonne **évènements** puis repérer la ligne **ONClik**, cliquer dessus pour ouvrir la case de droite puis cliquer sur cette dernière pour faire apparaître **FormClik.** S'ouvre alors la structure de la routine associée :

```
Procedure Tform1.Formclick(Sender:Tobject);
Begin
End;
```

La compléter comme suit:

```
Procedure Tform1.Formclick(Sender:Tobject);
 Var
        Pt:Tpoint;
Begin
        GetCursorPos(pt);
        Label1.caption:=' position  '+FloarTostr(pt.X)+'  '+FloarTOstr(pt.Y);
End;
```

On notera:

Le type **Tpoint** constitué par l'association de deux entiers X et Y est prédéfini dans Delphi et n'a pas besoin d'être redéfini .

Dans le titre de la procédure le mot **Sender** désigne l'objet au dessus duquel a lieu l'événement. Nous reviendrons plus loin sur ce mot.

A l'exécution seule apparaît la fiche Form1 vierge et la zone Label1.Si l'on clique sur le bouton gauche de la souris la position est affichée dans Label1

Par exemple **Position 344 567**

. Bien remarquer que la position affichée est bien une coordonnée dans la totalité de l'écran. Si avec la souris on déplace la fenêtre Form1 les coordonnées sont modifiées. Bien sûr le click n'est pris en compte que s'il est effectué à l'intérieur de Form1.

2° Les Evènements **MOUSE**

Dans l'inspecteur d'objets colonne évènements on repère en particulier :

OnMouseDown L'événement est bouton de souris enfoncé
OnMouseMove L'événement est Déplacement de la souris
OnMouseUp L'événement est relâchement du bouton.

Nous allons les utiliser .

Créons d'abord un nouveau projet comprenant un bouton Button1 3 zones d'affichage Label 1 2 et 3 une zone de dessin PaintBox1 .

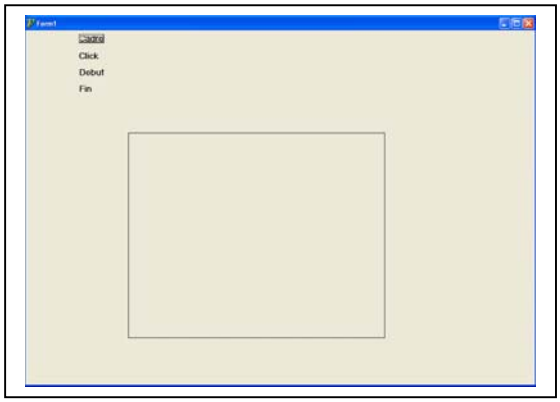
Positionnons soigneusement la zone de dessin en précisant dans l'inspecteur d'objets ses coordonnées Taille Hauteur Largeur :**Height400 Width500** et position **Top200 Left200**

A l'exécution une zone à peindre vierge est invisible , pour la faire apparaître nous allons y dessiner un rectangle en pressant le bouton. Pour cela cliquer sur le bouton pour faire apparaître la routine click et la compléter comme suit :

```
Procedure Tform1.Bitton1Clik(Sender :Tobject);
  Begin
        With PaintBox1.canvas do
        Rectangle(0,0,500,400);
  End;
```

Puis précisons les textes affichés au départ par les trois zones d'affichage

Label1.caption :=' Position' ;
Label2.caption :=' Debut ' ;
Label3.caption :=' Fin ' ;
Et sur le bouton : Button1.caption :=' Cadre' ;



Dans cet état une exécution après un unique appui sur le bouton donne l'écran ci contre.

Activons maintenant la PaintBox de façon à faire apparaître son inspecteur d'objets et la colonne evenements. Dans cette dernière sélectionnons l'événement **OnMouseDown** et cliquons sur la case à droite pour y afficher **PaintBox1Mouse**
La routine de gestion de l'événement s'ouvre :

```
Procedure
Tform1.PaintBox1MouseDown(Sender :Tobject ;Button :TmouseButton ;Shift :
TshiftState ;X,Y :Integer) ;
Begin
End;
```

Elle est assez différente de celle que nous avons souvent rencontrée pour le click sur un bouton.

Pour la gestion d'un événement souris il existe 4 paramètres :

Sender qui est l'objet ayant détecté l'action de la souris .Ce n'est pas toujours un bouton car il est parfaitement possible de gérer un click sur n'importe quel objet

Button Indique quel est le bouton de la souris qui est utilisé ***mbLeft*** (bouton gauche) ***mbMiddle*** (du milieu) ou ***mbRight*** (bouton droit) Par défaut c'est le bouton gauche qui est actif mais il est possible de modifier ce paramètre .
Shift Indique l'état des touches Alt Ctrl et Maj au moment de l'action de la souris permettant ainsi de les tester.
X,Y Les coordonnées de l'endroit où l'événement a eu lieu.

De la même façon nous allons activer sur cette même zone l'événement **OnMouseUp**

Nous allons maintenant compléter les procédures de façon à tracer dans la fenêtre de dessin un trait du point où l'on enfonce le bouton de la souris jusqu'à celui où on le relâche.

```
Procedure
Tform1.PaintBox1MouseDown(Sender :Tobject ;Button :TmouseButton ;Shift :
TshiftState ;X,Y :Integer) ;
Begin
  PaintBox1.Canvas.MoveTO(X,Y);     // Positionne le crayon au point XY
  Label2.Caption :=' Debut'+intTostr(X)+'  '+ intToStr(Y) ;
End;
```

Puis

```
Procedure
Tform1.PaintBox1MouseUp(Sender :Tobject ;Button :TmouseButton ;Shift :Tsh
iftState ;X,Y :Integer) ;
Begin
  PaintBox1.Canvas.LineTO(X,Y);      //Trace du point précédent au nouveau Xy ?
  Label3.Caption :=' Fin '+intTostr(X)+'  '+ intToStr(Y) ;
 End;
```

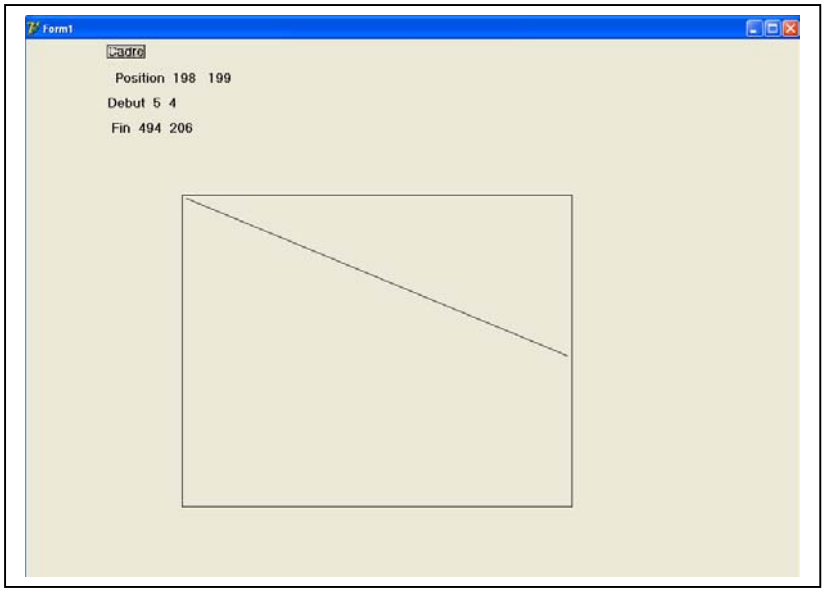
La première zone d'affichage Label1 sera utilisée pour afficher les coordonnées d'un click sur la fiche La première idée qui vient à l'esprit est d' activer l'événement **OnClick=FormClick** dans l'inspecteur d'objets de **Form1** et compléter la procédure associée : Cependant lorsque le OnClick est activé la routine qui apparaît ne fait pas état des propriétés Button Shift et XY il est bien sur possible de reprendre la routine du paragraphe précédent utilisant GetCursorPos mais on a vu que les coordonnées sont prises dans l'écran total et indépendantes de la fenêtre Delphi. Dans Form1 il faut donc utiliser plutôt **OnMouseDown .**

```
Procedure
Tform1MouseDown(Sender:Tobject;Button :TmouseButton ;Shift :TshiftState ;
X,Y :Integer) ;
      Begin
            Label1.caption:=' position  '+FloarTostr(X)+'  '+FloarTOstr(Y);
      End;
```

Lançons maintenant l'exécution. En cliquant tout près du coin supérieur gauche du cadre (en dehors) Label1 affiche une position voisine de 200 200 par exemple position 198 198 .(nous avons placé le coin de la PaintBox1 en 200 200)

Amenons alors le pointeur de la souris tout près du même coin et enfonçons lson bouton gauche puis faisons glisser jusqu'au milieu du côté droit et relâchons. Un

trait est tracé entre ces deux points et leurs coordonnées affichées dans Label 2 et 3 . Figure ci dessous/



Remarque :

Si dans les procédures précédentes on avait omis de préciser ***PaintBox****.canvas Mais écrit seulement Canvas.Move(X,Y) et LineTo(X,Y) Le logiciel aurait fait appel au Canvas de Form1 et le trait précédent aurait été tracé en dehors de la zone de dessin au voisinage du coin haut gauche de Form1.*