Tutorial 2, Delphi: Parabool

Versie	Datum	Auteurs	Opmerkingen
1	23-09-2001	Kees Hemerik (code) Roel Vliegen (tekst)	
2	18-03-2005	Kees Hemerik	Diverse tekstuele wijzigingen. Code van DrawGraph gewijzigd.
3	19-11-2008	Tom Verhoeff	Aangepast voor Lazarus

Inleiding

Doelstelling

In de eerste tutorial ('Geldspraak') heb je kennis gemaakt met *Lazarus* en een eenvoudige GUI applicatie gemaakt. Deze tutorial is hier het vervolg op. Er zullen in deze tutorial dan ook in principe geen dingen uitgelegd worden die al in de eerste verteld zijn, wel zullen enkele van de vele andere opties die *Lazarus* biedt uitgelegd worden. Meer specifiek de *UpDown*, *Image* en *Panel* controls en enkele properties zoals *Align* en *Anchors*. Bovendien wordt er een manier uitgelegd waarmee eenvoudige tekeningen gemaakt kunnen worden.

Benodigdheden:

- *Lazarus*, of een *Delphi* versie, het maakt niet zoveel uit welke. Merk op dat deze oefeningen oorspronkelijk gemaakt zijn met *Delphi 7 Enterprise Edition,* en daarna met *Lazarus* op een Mac. Als je een andere versie hebt kan het een er op jouw machine ander een beetje anders uitzien, mocht je hier toch onverhoopt problemen mee hebben, vraag het dan gewoon.
- de unit *Calculate*; deze wordt met de tutorial meegeleverd.

Parabool

In deze oefening wordt een applicatie uitgewerkt die betrekking heeft op de functie $F(x) = Ax^2 + Bx + C$. Van deze functie wordt de grafiek getekend (een parabool) en tevens worden de discriminant en de eventuele nulpunten bepaald met de bekende abc-formule. Deze parameters kun je dan real-time aanpassen zodat je de effecten van deze parameters op de parabool goed kunt zien.

Oefening Parabool

1. Maak een nieuw project aan met de naam parabool; let er op dat je ook dit project weer netjes in een eigen map opslaat. Denk ook aan de Compiler Options.

2. In de Geldspraak-tutorial heb je verschillende controls direct op het form gezet. Als je verschillende groepen controls op een form hebt staan die je kunt rangschikken in afzonderlijke groepen is het niet netjes als je ze allemaal gewoon samen op het form zet. Het wordt dan 1 grote brei van controls. Dit is te voorkomen door alle controls die bij elkaar horen in een panel te plaatsen. Dit heeft tot gevolg dat de positie van de controls wordt bepaald ten opzichte van de linker boven hoek van het panel. Plaats nu een *TPanel* op je form.

3. Je kunt nu controls in het panel zetten. Vul het panel zoals in *Figure 1*. Geef als namen aan de edit-boxen EditA, EditB, EditC, EditD, EditW1, EditW2. Stel de default waarden in voor A, B en C, zorg dat de sneltoetsen bij de labels correct zijn en dat de onderste drie edit-boxen read-only zijn. Let ook op de tab-order.

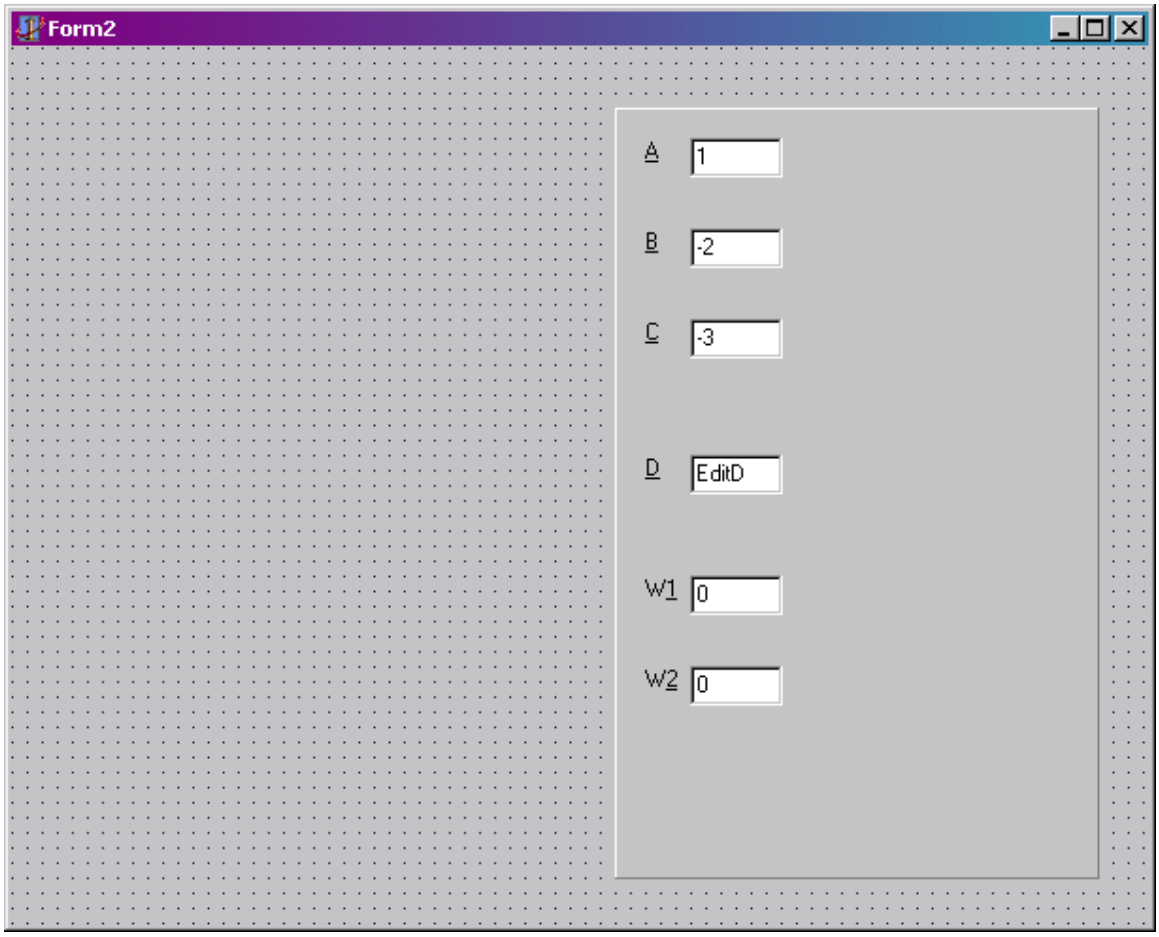


Figure 1

4. Het is de bedoeling dat het panel mooi passend aan de rechterkant in het form komt te staan, ook als het form van groter of kleiner gemaakt wordt. Dit kan ingesteld worden met de *Align*-property. Stel de *Align*-property van het panel in op *alRight*. Start vervolgens het programma eens op en probeer het form te resizen.

5. Voeg nu aan de drie bovenste edit-boxen *UpDown*-controls toe, deze staan bij de *Common Controls*. In *Figure 2* kun je hier een voorbeeld van zien. Geef dan met de *Associate*-property aan bij welke Edit-box ze horen. Stel ook de *Min* en *Max* properties in (–100 tot 100).

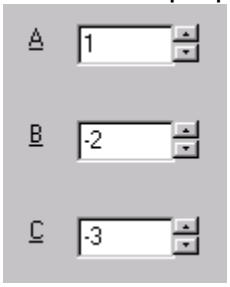


Figure 2

6. Voeg nu de unit Calculate toe aan het project. In deze unit staan de volgende procedures:

```
function F(A, B, C, X: Real): Real;
{pre: true}
{ret: A*X^2 + B*X + C}
```

Met deze procedure wordt de functiewaarde van de functie $F(X)=AX^2+BX+C$ bij een bepaalde X berekend.

```
function Discr(A, B, C: Real): Real;
// Discriminant
{pre: true}
{ret: B*B - 4*A*C}
```

Met deze functie wordt de discriminant van de functie $F(X)=AX^2+BX+C$ berekend.

```
procedure VKV(A, B, C: Real; out W1, W2: Real);
// Roots of A*X^2 + B*X + C = 0
{pre : A <> 0, 0 <= Discr(A, B, C)}
{post: (forall X: A*X^2 + B*X + C = A*(X-W1)*(X-W2) )}
```

Deze functie berekent de nulpunten van de functie $F(X)=AX^2+BX+C$, indien deze bestaan. **N.B.**: Let er bij het aanroepen van VKV op, dat aan de preconditie is voldaan.

Gebruik deze functies nu om de edit-boxen D, W1 en W2 in te stellen. Het is bedoeling dat D de discriminant van de functie vastgelegd door A, B en C wordt en W1 en W2 de twee nulpunten. Indien A, B of C veranderd worden moeten deze waarden opnieuw berekend worden. Dit kun je net als in de eerste tutorial doen door de *OnChange*-events van de edit-boxen te gebruiken. Je kunt hierbij kiezen of je een enkele event-handler maakt voor alle drie de edit-boxen of dat je voor elke een aparte maakt.
N.B.: Houd rekening met speciale gevallen, zoals A = 0 .

7. Het is de bedoeling dat de linkerkant van het form gebruikt gaat worden om de parabool op te tekenen. Nu zou er direct op het form getekend kunnen worden, maar dan zou er bijvoorbeeld per ongeluk door de knoppen heen getekend kunnen worden. Zet daarom een *TPaintBox*-control neer die gebruikt kan worden om op te tekenen. Stel ook van deze control de *Align*-property in, en wel op *alClient*. Dit heeft tot gevolg dat alle overgebleven oppervlakte gebruikt wordt door de TPaintBox-control.

8. Als je het programma nu opstart zie je een wit veld. Op dit veld kan getekend worden. Zoals in het vorige punt al opgemerkt is, zou er ook op het form getekend kunnen worden, zo kan er op heel veel controls getekend worden, dit is in Lazarus dan ook netjes gestandaardiseerd door aan deze controls een *Canvas* toe te voegen. Een Canvas is eigenlijk gewoon een tekenvel, waarop je op verschillende manieren kunt tekenen, in deze tutorial zal hiervoor een *Pen* gebruikt worden. Deze “pen” kun je bijvoorbeeld verplaatsen (al dan niet met de pen op het “papier”) en je kunt bijvoorbeeld ook het soort pen aanpassen. Een positie op het canvas is bepaald door een x- en een y-coördinaat, waarbij de linkerbovenhoek (x=0, y=0) is en (x=200, y=100) is bijvoorbeeld 200 pixels naar rechts en 100 pixels naar onder vanaf de linkerbovenhoek geteld.

   Er zullen voor het tekenen twee methodes en een property van het Canvas gebruikt worden:

   Canvas.MoveTo(x, y) verplaats de pen naar coördinaat (x, y) terwijl de pen “niet op het papier is”, bij gebruik van deze methode zal er op beeld niets veranderen, alleen de positie van de pen zal veranderen.

   Canvas.LineTo(x, y) verplaats de pen naar coördinaat (x, y) terwijl de pen “op het papier is”, dit heeft tot gevolg dat er een lijn wordt getrokken van de oude naar de nieuwe plaats van de pen, de pen staat hierna op de nieuwe positie.

9. Er moet een parabool getekend worden en wel in een assenstelsel. Dit tekenen doen we in de event-handler voor het *OnPaint* event van de PaintBox. Dubbelklik daartoe in het OnPaint event van de PaintBox.

   De x-as van dit assenstelsel kan getekend worden door eerst de pen naar bijvoorbeeld het meest linkse punt van de x-as te verplaatsen met *MoveTo* en daarna met *LineTo* een lijn naar rechts te trekken. Op dezelfde manier kan ook de y-as getekend worden. Plaats de x-as en de y-as halverwege de PaintBox. De afmetingen van de PaintBox zijn te verkrijgen via *Width* en *Height*. Sla ze voor het gemak op in lokale variabelen w, h.

   Voor de x-as moet dus eerst met MoveTo(0, h div 2) de pen naar de goede beginpositie worden gebracht en hierna met LineTo(w, h div 2) wordt de x-as getrokken.

   Implementeer nu de OnPaint handler die de assen tekent:

```
with PaintBox1.Canvas do
```

```
begin
  w := PaintBox1.Width;
  h := PaintBox1.Height;
   // draw x-axis
  MoveTo(0, h div 2);
  LineTo(w, h div 2);
   // draw y-axis
   ..
   ..
end;
```

10. Nu moet alleen de parabool nog op het scherm getekend worden. Ook dit kan weer gedaan worden in de OnPaint handler en wel door eerst de pen naar het meest linkse punt van de parabool te verplaatsen en daarna steeds een lijn te trekken naar de volgende functiewaarde. Dan blijft de vraag nog hoe de parabool afgebeeld kan worden op het scherm.

    Het is natuurlijk de bedoeling dat het punt (0,0) samenvalt met het snijpunt van de twee assen van het assenstelsel. Bovendien is het niet handig als een pixel op het beeld met een roostereenheid overeenkomt. Dit zou betekenen dat de parabool getekend wordt op een gebied van ongeveer 600 bij 600 roosterpunten. Neem daarom 30 pixels als roostereenheid (deze definitie komt direct onder **implementation**):

```
const
  GridUnit = 30; { number of pixels per grid unit }
```

    Neem de volgende procedure op om een roostercoordinaat om te rekenen naar een schermcoordinaat:

```
function TForm1.Scale(S: Real; C: Integer): Integer;
{pre: true}
{ret: screen coordinate for grid coordinate S centered at C}
begin
  Scale := Round(S * GridUnit + C);
end;
```

    Er zal straks ook een procedure nodig zijn om een schermcoordinaat weer terug te rekenen naar een roosterpunt, dit kan met de volgende procedure:

```
function TForm1.UnScale(S: Integer; C: Integer): Real;
{pre: true}
{ret: grid coordinate for screen coordinate S centered at C}
begin
  UnScale := (S-C) / GridUnit;
end;
```

    Met volgende procedure kan nu de parabool op beeld gezet worden:

```
procedure TForm1.GDrawGraph(A, B, C: Real);
{pre: true}
{post: graph of A*x^2 + B*x + C is drawn on PaintBox}
var
  X, Y: Integer; { to traverse the graph }
  w, h: Integer; { width and height of paintbox }
begin
  with PaintBox1.Canvas do begin
    w := PaintBox1.Width;
    h := PaintBox1.Height;

    {Find starting point for drawing}
    X := 0;
    Y := Scale(-F(A, B, C, UnScale(X, w div 2)), h div 2);
    MoveTo(X, Y);

    {Draw line segments}
    while (X <> PaintBox1.Width) do begin
      X := X+1;
      Y := Scale(-F(A, B, C, UnScale(X, w div 2)), h div 2);
        { note the minus: screen Y grows down, grid up }
      LineTo(X, Y);
    end;
  end;
end;
```

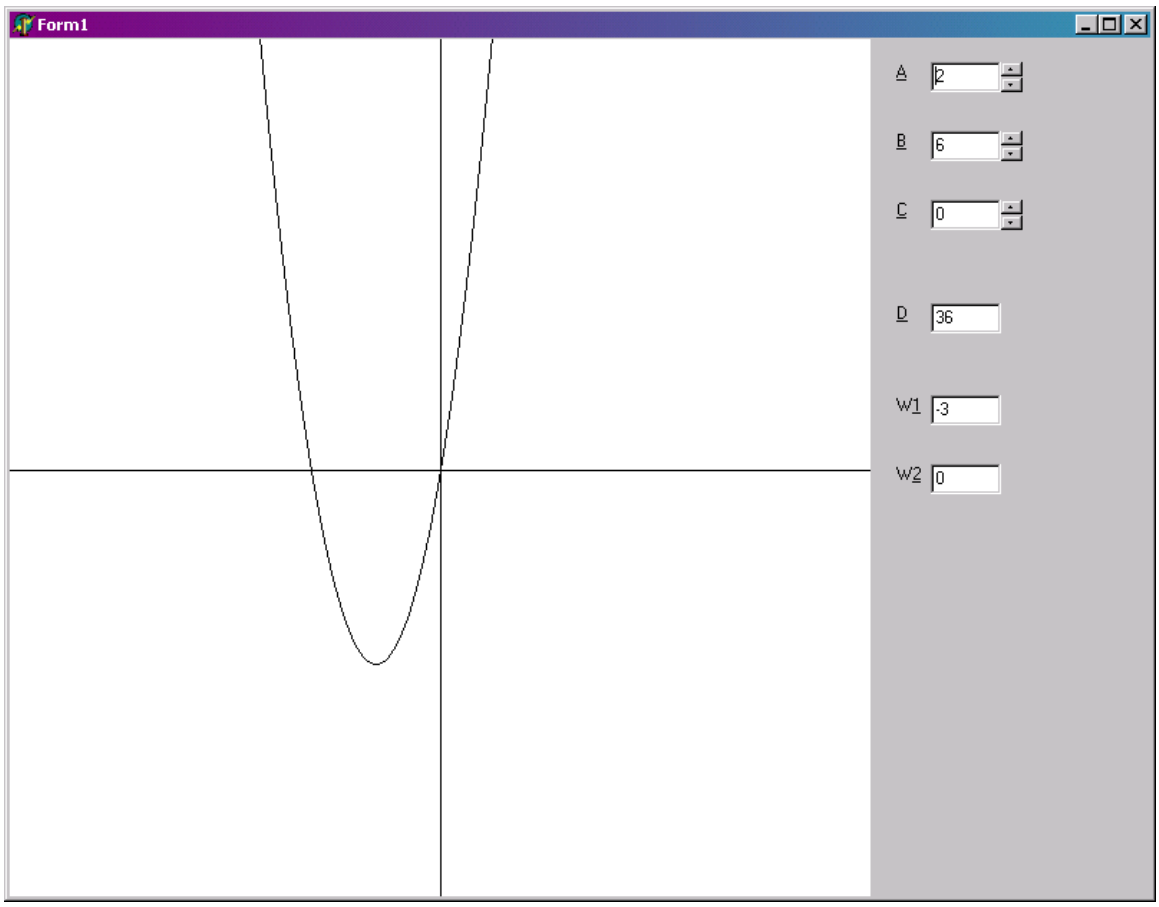
Deze procedure rekent voor alle x-waarden van de PaintBox een functiewaarde uit en trekt tussen deze functiewaarden een lijn. Hierbij wordt Scale en UnScale gebruikt om ervoor te zorgen dat X-waarde van het tekenveld wordt geconverteerd naar de juiste waarde voor het berekenen van de functieaanroep en dat de functiewaarde hierna weer wordt omgerekend naar een coördinaat in de bitmap.

Zorg er voor dat deze procedure wordt aangeroepen in de OnPaint handler.

11. Bij het veranderen van de A, B of C waarde verandert de parabool niet. Dat komt omdat de OnPaint handler alleen automatisch wordt aangeroepen als het run-time systeem dat nodig vindt. In dit geval moet het expliciet gebeuren door PaintBox1.Repaint aan te roepen in de OnChange handler van de Edit-boxes.

12. Hieronder volgt nog een voorbeeld van hoe het programma er ongeveer uit zou moeten zien.

13. Je kan het programma nog op allerlei manier verfraaien, bijvoorbeeld door streepjes en getallen bij de assen te zetten. Het is dan beter om een aparte procedure TForm1.GDrawAxes te maken voor het tekenen en deze in de OnPaint handler aan te roepen. Het is ook nuttig om de grafiek in een ander kleurtje te tekenen (zeg clRed).


Form1
A
2
B
6
C
0
D
36
W1
-3
W2
0