Detlef Overbeek, Noud van Kruysbergen

"Hallo Wereld!"

Programmeercursus Pascal, deel 1: de basis

Pascal? Bestaat dat dan nog? Het lijkt inderdaad alsof de programmeertaal Pascal uit de oertijd stamt en als fossiel ergens is achtergebleven. Maar niets is minder waar. Pascal heeft zich al die jaren doorontwikkeld en is platformonafhankelijk, oftewel geschikt voor meerdere besturingssystemen als Windows, Mac en Linux. In deze c't beginnen we met een serie artikelen die gezamenlijk een basiscursus Pascal vormen.

Programmeren wordt door veel mensen als gortdroge kost gezien, maar is in feite een buitengewoon creatief proces. Het is een wonderbaarlijke ervaring als je voor het eerst een eigen programma hebt gemaakt. Al moet je natuurlijk wel van wat puzzelwerk houden en een beetje geduld hebben.

De meeste programmeertalen zijn imperatief, wat wil zeggen dat zij bevelen, commando's of instructies aan een processor geven. Die worden netjes op volgorde uitgevoerd. Sommige bewerkingen die vaker uitgevoerd moeten worden, kunnen worden samengevoegd in een deelprogramma, dat dan als functie, procedure. subroutine of methode wordt gebruikt. In dat geval spreken we van procedureel programmeren. Een onderdeel van het procedurele programmeerparadigma is objectgeoriënteerd programmeren. Daarbij worden abstracte datatypes vervangen door klassen en subklassen. Dat maakt het makkelijker om programmeerwerk uit te wisselen om gezamenlijk te ontwikkelen, maar het programmeren zelf wordt daarmee wel een stuk complexer en abstracter.

Niklaus Wirth introduceerde in 1970 de nieuwe programmeertaal Pascal. Deze taal was voornamelijk bedoeld voor het programmeeronderwijs, maar kreeg door zijn eenvoud en door de snelle, soms gratis compilers ook snel daarbuiten een zeer groot aantal gebruikers. Wirth zette zich sterk af tegen de gangbare teneur van objectgeoriënteerde talen, die naar zijn mening doorgaans de verkeerde oplossingen voor de gestelde problemen boden. Het door Wirth voorgestelde mechanisme leverde dan ook een compactere en eenvoudige taal op.

Later ontwikkelde hij de taal Oberon, die als eerste taal aanknopingspunten voor softwarecomponenten had en dan ook bekender is onder de naam Component Pascal. Pascal was in veel opzichten een concurrent voor het vergelijkbare en ongeveer even oude C, maar er zijn duidelijke verschillen in de uitgangspunten. Heel globaal zou je kunnen zeggen dat Pascal gebaseerd is op het uitgangspunt dat een programmeur zijn werk duidelijk moet structureren, terwijl C-achtige talen meer aan de eigen verantwoordelijkheid van de programmeur overlaten.

Lazarus

De kracht van de Pascal-programmeertaal is dat je door de helderheid en leesbaarheid van de code al snel in staat bent om ermee te programmeren. Je kunt er letterlijk Nederlands in gebruiken (uiteraard alleen voor variabelen en procedure- en functienamen, de taal op zich blijft Engels wat betreft de commando's) zodat je de code en structuur beter kunt begrijpen en volgen. De compiler is veel compacter dan bij C-achtige talen en daardoor ook veel sneller. Dat was zeker vroeger van groot belang, maar heden ten dage nog steeds.

Door de eenvoud van de taal is het mogelijk om zonder grote wiskundige kennis te beginnen met programmeren. Jongeren worden niet afgeschrikt door ingewikkelde en onbegrijpelijke structuurindelingen. Gezien het grote tekort aan programmeurs is laagdrempeligheid een voorwaarde om met name veel jonge mensen enthousiast te krijgen voor dit vak, wat bij andere talen een stuk minder eenvoudig ligt.

Het feit dat Pascal helemaal bij de tijd is, wordt geïllustreerd door de nieuwste ontwikkelingen. Met het opensourceproject Lazarus kan worden ontwikkeld voor Android en zelfs voor iOS. Lazarus is een ontwikkelomgeving die gebaseerd is op de Free Pascal Compiler. De ontwikkelomgeving is te downloaden via de softlink aan het eind van dit artikel. Daar vind je ook een beperkte versie van Delphi, de beroemdste telg van de Pascal-talen, waarvoor echter betaald moet worden.

De programma's die we in deze programmeercursus maken, zijn qua code zowel voor Lazarus als Delphi geschikt. Ook alle bronbestanden zijn via de softlink te downloaden. Als achtergrondinformatie bij de cursus is het boek 'Essential Pascal' van Marco Cantù handig. Via de softlink kun je aan de juiste programma's komen om deze cursus te volgen. Er is een versie van Dephi die je een half jaar lang volledig kunt gebruiken.

Eenvoud

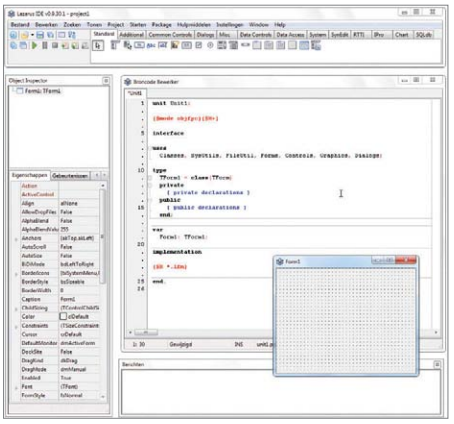
Om de eenvoud van de taal en het programmeren te illustreren, zullen in dit eerste artikel een paar mini-programma's worden uitgelegd en getoond. Ten eerste natuurlijk het programma zoals dat bij elke taal wordt gebruikt: "Hello World!". Dat doet niets anders dan die tekst op het scherm weergeven. We maken daar een paar kleine variaties op met knoppen om op die manier de ontwikkelomgeving (Integrated Development Environment – IDE) voor de programmeertaal te leren kennen met zijn teksteditor en een aantal extra's om snel en makkelijk code te kunnen schrijven.

Het tweede programma doet niet meer dan een mededeling op het scherm zetten met een mogelijkheid om die uit te schakelen. Het derde programma laat zien hoe je een afbeelding kunt laten bewegen en daar muziek aan toevoegen. Dat zou het begin van een spelletje kunnen zijn. Bij geen van deze programmaatjes is enige voorkennis van programmeren vereist. Alles wordt stap voor stap uitgelegd.

Procedureel programmeren

In de inleiding is de term procedureel programmeren al gevallen. Het is een begrip uit de informatica waarbij programma's gemaakt worden in de vorm van opdrachten die meteen uitgevoerd kunnen worden. Een belangrijk begrip binnen Pascal zijn routines. Dat zijn een aantal opdrachten die gezamenlijk een unieke naam hebben en met die naam meerdere keren aangeroepen kunnen worden. Dan hoef je dus niet de hele tijd dezelfde opdrachten in te typen, maar heb je één stuk code dat je op verschillende plekken in het programma kunt gebruiken door de betreffende routine aan te roepen. Vanuit dit gezichtspunt kun je routines beschouwen als een eerste, simpele, vorm van code-inkapseling (code encapsulation).

In Pascal kan een routine twee vormen aannemen: een procedure of een functie. In theorie is een procedure een operatie die je de computer kunt laten uitvoeren en is een functie een berekening die een waarde oplevert. Beide type routines kunnen verschillende parameters bevatten of gebruiken. In de praktijk is het verschil tussen functie en procedure niet zo groot: je kunt een functie aanroepen om uit te voeren en niets met



De ontwikkelomgeving Lazarus met bovenaan het menu, de tabbladen voor de componenten en de pictogrammen, links de object inspector, in het midden het scherm met de broncode en een formulier voor de grafische interface en onderin een mededelingenvenster.

het resultaat doen (wat eventueel tot een foutmelding kan leiden) of een procedure aanroepen die een resultaat doorgeeft via de gebruikte parameters.

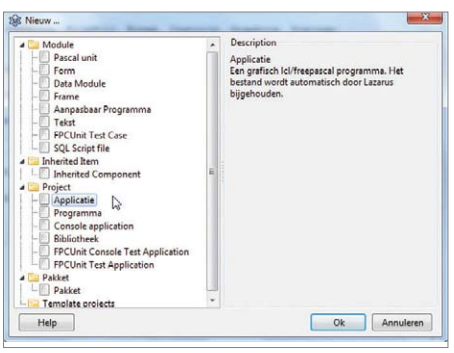
Hier de definitie van een procedure:

```
Procedure Hello;
Begin
 Writeln ('Hello World!')
End;
```

Het enige wat deze procedure doet is de tekst 'Hello World!' uitvoeren. Hier een voorbeeld van een functie:

```
Function Verdubbelen (value: Integer): Integer;
Begin
    Verdubbelen {Een rekeneenheid} := value {Waarde} * 2;
End;
```

Dit is een programmaonderdeel dat iets berekent. Eigenlijk staat hier: een bepaalde waarde (value) kan van alles zijn, maar moet wel van het type integer zijn, oftewel een geheel getal. Dat in tegenstelling tot een rationeel getal als 1/4 of een reëel getal als pi (π). Een integer kan ook negatief zijn, maar er zit geen komma in. Een getal van het type integer kan variëren



In de meeste gevallen zul je beginnen met het maken van een nieuwe applicatie.

tussen de 0 en 2^{32} – 1. Een andere schrijfwijze (syntax) van deze functie is:

```
Function Verdubbelen2 (value: Integer) : Integer;
// Verdubbelen2 is hier de naam van de Functie
Begin
    Result := value * 2; { dit is de eigenlijke berekening }
End
```

Het teken := betekent 'wordt': resultaat wordt waarde vermenigvuldigd met 2. De tekens // of {} geven aan dat hierachter of hiertussen commentaar of een toelichting kan worden getypt. Deze tekst maakt geen deel uit van de programmeercode. Bij // geldt de hele regel als commentaar. De code begint dan pas weer op de volgende regel. Als je een tekstblok wilt maken over meerdere regels, is het handiger om de {}-accolades te gebruiken. Die regels worden dan niet uitgevoerd – de compiler leest deze commentaarregels niet.

Bij de laatste functie is Result dan de uitslag van de berekening. Result is geen type en heeft dus geen beperkende eigenschappen. Een variabele kan waarden aannemen en is eigenlijk niet meer dan een lege container waar iets in kan.

Wat bij deze voorbeelden opvalt is dat sommige woorden vetgedrukt zijn. Dat zijn gereserveerde begrippen die in de taal Pascal een vaste betekenis hebben die door de compiler herkend wordt. Een compiler is een computerprogramma dat geschreven programmacode omzet in een uitvoerbaar programma. Het vertalen of omzetten wordt dan ook compileren genoemd.

Het voorbeeld van de procedure is in principe al voldoende om een programma te maken. De enige voorwaarde is dat het iets moet uitvoeren. We kunnen nu dan ook beginnen met ons eerste programma.

Installeer de IDE van Lazarus en kies bij 'Bestand / Nieuw' onder 'Project' voor 'Applicatie'. Je kunt de taal e=ventueel ook wijzigen: Ga naar 'Instellingen / Options' en dan naar 'Omgeving / Bureaublad', waar je rechts onder 'Taal' een taal kunt instellen.

Hello World!

Na het selecteren van 'Applicatie' gebeuren er twee dingen: er verschijnt een venster met een leeg formulier ('Form1') en in de 'Broncode Bewerker' komt de code te staan voor dit formulier. Bij de grafische variant kun je objecten door middel van slepen of dubbelklikken op je formulier zetten.

We gaan nu de objectgeoriënteerde applicatie maken, waarvoor we de eerder genoemde procedure Hello met het bericht "HelloWorld!" en een knop gaan gebruiken:

```
unit Unit1;
{$mode objfpc}{$H+}
interface
uses
    Classes, SysUtils, FileUtil, LResources, Forms,
        Controls, Graphics, Dialogs;
type
    TForm1 = class(TForm)
    private
        { private declarations }
    public
        { public declarations }
    end;
var
    Form1: TForm1;
implementation
    {$R *.lfm}
end.
```

De code begint met een unit. Pascal-toepassingen maken uitgebreid gebruik van units oftewel programmamodules. Units waren in eerste instantie de basis van de modulaire opbouw in de taal voordat de term klassen als modulaire elementen werd ingevoerd. In een Pascal-applicatie heeft elke Form een corresponderende unit achter zich. Als je een nieuwe Form aan een project toevoegt via de corresponderende werkbalkknop of met 'Bestand / Nieuwe Form' in het menu, dan voegt Pascal in feite een nieuwe unit toe die de klasse voor de nieuwe Form definieert.

Het uses-deel aan het begin van de code geeft aan welke andere units we nodig hebben. Dat is inclusief de units die de datatypen definiëren waar we naar refereren in de definitie van andere datatypen, zoals de componenten die gebruikt worden bij een formulier dat we definiëren.

Een (eventueel) tweede uses-deel aan het begin van de implementatiesectie wijst op meer units die we nodig hebben voor een aantal systeemroutines. Als je units van andere (zelfgemaakte) routines en methoden wilt gebruiken, dan moet je die in dit tweede uses-deel toevoegen in plaats van in het eerste.

Alle units waar je naar verwijst, moeten aanwezig zijn in de directory van het project of in de directory van het zoekpad (je kunt het zoekpad voor een project opgeven bij 'Project / Projectopties' onder 'Compileropties / Paden:').

Zet op het lege formulier nu een knop uit de componentenwerkbalk. Dat kan op twee manieren: klik met de muis op de TButton-knop en vervolgens op het formulier of dubbelklik op de TButton-knop in de componentenwerkbalk. Aan de code wordt nu de regel Button1: TButton; toegevoegd. Als je niet de hele tijd met de muis tussen de vensters van de 'Broncode Bewerker' en 'Form1' wilt wisselen, kan dat ook met de F12-toets. Dubbelklik nu op de button en dan verschijnt de automatisch gegenereerde procedure TForm1.Button1Click(Sender: TObject);.

Zet tussen de regels begin en end een regel met de code Button1.Caption:= 'Hello'. De kleuren die door de editor gebruikt worden, zijn in te stellen via 'Instellingen / Opties / Editor / Beeld / Kleur'. Lazarus heeft de mogelijkheid om delen van de code automatisch aan te vullen. Toets tijdens het intypen van 'Cap' op Ctrl+Spatie en je krijgt de mogelijke opties te zien. Ga met cursortoetsen naar de eigenschap 'Caption' en druk op Enter, waarna de resterende code wordt toegevoegd en je alleen je eigen gegevens nog hoeft in te vullen. De eigenschap (property) Caption slaat op de tekst die op de knop komt te staan. De naam van de knop is Button1, als je er meer

aanmaakt heten die Button2 enzovoort. Als het programma uiteindelijk gecompileerd is en uitgevoerd wordt, zal deze knop het opschrift 'Hello' krijgen als je er op klikt.

Er zijn een aantal mogelijkheden om het programma te compileren en uit te voeren. Je kunt op de groene driehoekige startknop klikken linksboven op de IDE-werkbalk, of op de Functietoets F9 van je toetsenbord. Met Ctrl+F9 wordt de code gecompileerd zonder dat het programma wordt uitgevoerd. Daarmee kun je controleren of alles goed is gegaan. Bij een wat groter project kan het compileren even duren.

Als je het project wilt bewaren, ga je naar 'Bestand / Opslaan als' en vervolgens verschijnt er een venster dat je de gelegenheid biedt te kiezen waar je het wilt opslaan met een naam naar keuze en de extensie .lpi. Standaard wordt dat project1.lpi. Daarna volgt een tweede venster dat vraagt hoe de .pas-file moet heten. Standaard is dat unit1.pas.

De extensie lpi voor de projectnaam komt van Lazarus Project Information. Het projectbestand is in feite een XML-file. Een lpr-bestand is een Lazarus Program Resource en de extensie .pas is gereserveerd voor de Pascal-sources. Die laatste zijn meestal gekoppeld aan een projectfile. Op http://wiki.lazarus.freepascal.org/File_extensions staat een overzicht van alle extensies.

Nieuw project

Sla eerst alles op wat je tot nu toe gedaan hebt voordat we met een nieuw project beginnen. Deze keer zetten we meerdere onderdelen op het formulier: een TButton van de componentenwerkbalk Standard en een TLabel van dezelfde werkbalk. Geef deze ongeveer dezelfde indeling als op de afbeelding te zien is. De grootte van het formulier kun je zelf aanpassen.

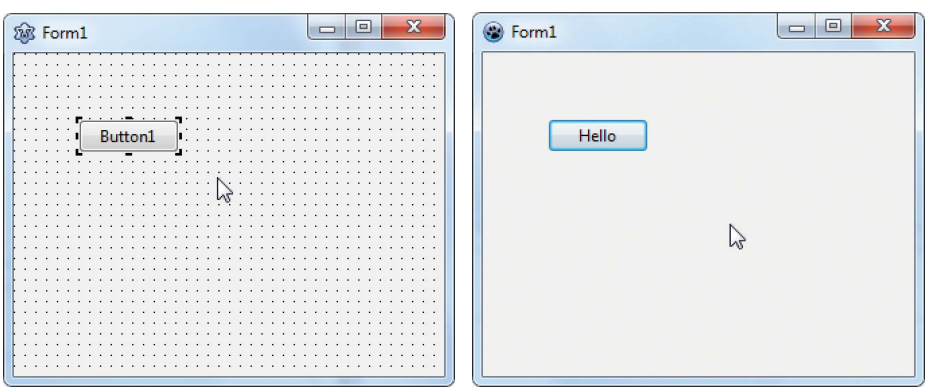
In de Object Inspector kun je een aantal eigenschappen van de verschillende objecten aanpassen. De naam van het formulier kun je bijvoorbeeld veranderen van Form1 in 'voorbeeldformulier'. Die naam mag overigens alleen uit letters en cijfers bestaan, zonder spatie. Bij de Caption kun je een titel aan het formulier meegeven, in dit geval 'Voorbeeldvenster'. Dat kan ook een complete mededeling zijn. Bij de objecteigenschappen zie je ook dat ieder object een 'Width' heeft: ook het formulier is een object, net zo goed als het TLabel en de TButton.

Bij de eigenschap 'Hint' van de elementen kun je intypen wat je te zien moet krijgen als je met de muis over die elementen beweegt. Om die tijdens de uitvoer van het programma ook zichtbaar te maken, moet je de eigenschap 'ShowHint' wel op 'True' zetten, wat standaard niet het geval is. Je kunt een korte tekst rechtstreeks in het invoerveld intypen, maar als je op de drie puntjes erachter klikt, verschijnt er een venster waarin je langere teksten kunt bewerken. Je kunt bij de Object Inspector nog meer eigenschappen van elementen aanpassen, waaronder kleuren en lettertypen.

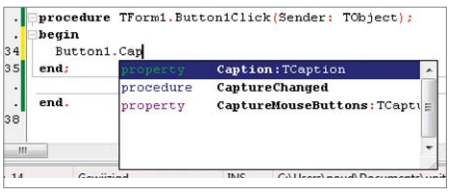
Hieronder volgt de voorbeeldcode van unit:

```
unit Unit1;
{$mode objfpc}{$H+}
interface
uses
    Classes, SysUtils, FileUtil, Forms, Controls, Graphics,
        Dialogs, StdCtrls;
type
    { TForm1 }
    TForm1 = class(TForm)
        Button1: TButton;
        Label1: TLabel;
        procedure Button1MouseEnter(Sender: TObject);
    private
        { private declarations }
    public
        { public declarations }
    end;
var
    Form1: TForm1;
implementation
{$R *.lfm}
{ TForm1 }
procedure TForm1.Button1MouseEnter(Sender: TObject);
begin
    if Button1.Left > Button1.Width
    then
        Button1.Left := Button1.Left - Button1.Width
    else
        Button1.Left := Form1.Width - Button1.Width;
end;
end.
```

In de Object Inspector moet bij de 'Gebeurtenissen' van Button1 dan wel de routine Button1MouseEnter geselecteerd worden bij de gebeurtenis 'OnMouseEnter', anders werkt het niet. In deze routine wordt de waarde Button1.Left gebruikt, die aangeeft wat de linkerpositie van de knop is. Deze waarde wordt telkens met de breedte van de knop verlaagd. Hierdoor springt de knop telkens naar links als je er met de muis overheen beweegt. Als hij aan de linkerkant is aanbeland, springt hij weer helemaal naar rechts. Op deze manier wordt het lastig zoniet onmogelijk op die knop te klikken. Je kunt het programmaatje altijd afsluiten door op het Windows-kruisje rechtsboven te klikken.



De kop is eraf: er staat een knop op het formulier. Het oogt wat sober en weinig spectaculair, maar het is wel je eerste Pascal-programma!



Om het je makkelijk te maken kan Lazarus zelf de code aanvullen met de gewenste opties.

Bewegen op muziek

Voor gevorderden is het leuk om te weten dat er aardig wat ingewikkelde spelletjes beschikbaar zijn, waaronder zelfs een versie van Pacman. Het is zeker interessant om te zien hoe dat technisch in elkaar zit. Deze klassieker op spelletjesgebied is bovendien leuk om te spelen. Er zijn ook nog legio andere voorbeelden beschikbaar.

Hier willen we het echter simpel houden met een programmaatje dat geïnspireerd is op het sprookje van de Chinese Nachtegaal. Dit programma bestaat in principe uit twee delen: je ziet de nachtegaal bewegen en hoort tegelijkertijd zijn lied. Voor de nachtegaal zelf hebben we een aantal tekeningen nodig. Eigenlijk zouden dat er 25 per seconde moeten zijn om een vloeiend beeld op te leveren, maar dat wordt wel erg bewerkelijk. In de praktijk gebruiken we slechts 10 afbeeldingen, die we in verschillende volgorde steeds laten herhalen. Het staat je vrij om ook andere afbeeldingen of foto's te gebruiken. Als extraatje hebben we ook een pictogram toegevoegd dat voor het uiteindelijke programma gebruikt zal worden.

We zorgen ervoor dat de vogel continu op dezelfde plek blijft zitten, anders verspringt hij voortdurend en maken we het onszelf onnodig moeilijk. Alleen de opening van de bek varieert om het zingen te simuleren. Daarnaast variëren we de kleur van de keel om het zingen wat inspannender te doen lijken.

Aan het formulier van dit project is al te zien dat hier meer onderdelen nodig zijn. Er is een timercomponent voor het afwisselen van de afbeeldingen. Deze staat op de werkbalk 'System'. De component voor afbeeldingen heet 'TImage' en staat op de werkbalk 'Additional'.

De basiscode voor dit project ziet er inmiddels herkenbaar uit:

```
unit uNightingale;
{$mode objfpc}{$H+}
interface
uses
    Forms, StdCtrls, ExtCtrls, Buttons, MMsystem,
        Controls, Classes;
type
    { TfrmNightingale }
    TfrmNightingale = class(TForm)
        btnClose: TBitBtn;
        btnStart: TBitBtn;
        imgNightingale: TImage;
        Panel1: TPanel;
        Timer1: TTimer;
        procedure btnStartClick(Sender: TObject);
        procedure FormCreate(Sender: TObject);
        procedure Timer1Timer(Sender: TObject);
    end;
var
    frmNightingale: TfrmNightingale;
implementation
uses sysutils;
{$R *.lfm}
{ TfrmNightingale }
procedure TfrmNightingale.btnStartClick(Sender: TObject);
begin
    // must be wav files...
    sndPlaySound('..\wav\Nightingale_song.wav', SND_
        FILENAME or SND_ASYNC);
    Timer1.Enabled := True;
end;
```

Met daaronder de resterende procedures. Met de knop 'Start' kun je het programma laten beginnen met bewegen en het geluid aanzetten. De opdracht sndPlaySound is verbonden aan de unit sysutils zoals genoemd bij de implementation-sectie. Het is niet meer dan een simpele opdracht, Windows heeft alles wat hier ingewikkeld aan is alvast voor je opgelost via deze opdracht. Het gevolg is dat je tegelijk met de beweging geluid kunt laten horen. Normaal gesproken hoor je een beetje geluid en dan beweegt het beeld, enzovoort. Dat komt nogal hakkelig en lelijk over, maar is een logisch gevolg van het feit dat een processor maar een ding tegelijk kan.

De timer is een hele nuttige component. Hij loopt synchroon met de interne tijd van Windows. De timer moet via de Object Inspector worden ingesteld. Hij moet standaard op uit staan (Enabled False) met tien als interval. Maar daar mag je natuurlijk ook mee spelen. Als geluid hebben we een originele opname van een nachtegaal gebruikt.

Bij het starten van het programma gaat de procedure FormCreate eerst een aantal dingen instellen voor de afbeelding, waaronder de positie en de grootte. De procedure Timer1Timer is het hart van de code:

```
procedure TfrmNightingale.Timer1Timer(Sender:
    TObject);
    // deze procedure is genest om ervoor te zorgen dat
    er eerst gegevens geladen worden
        procedure DoPic(const fPic: TFilename; aTag:
            integer);
        begin
            imgNightingale.Picture.LoadFromFile(fPic); //
            het plaatje wordt geladen
            imgNightingale.Tag:= aTag; // om de
            volgorde van de plaatjes aan te passen
        end;
    begin
        case imgNightingale.Tag of
            0: DoPic('..\jpg\Nightingale1.jpg',  1 );
            1: DoPic('..\jpg\Nightingale2.jpg',  2 );
            2: DoPic('..\jpg\Nightingale2.jpg',  3 );
            3: DoPic('..\jpg\Nightingale2.jpg',  4 );
            4: DoPic('..\jpg\Nightingale3.jpg',  5 );
            5: DoPic('..\jpg\Nightingale3.jpg',  6 );
            6: DoPic('..\jpg\Nightingale3.jpg',  7 );
            7: DoPic('..\jpg\Nightingale4.jpg',  8 );
            8: DoPic('..\jpg\Nightingale4.jpg',  9 );
            9: DoPic('..\jpg\Nightingale4.jpg', 10 );
            10: begin
                    DoPic('..\jpg\Nightingale2.jpg', 0);
                                        // terug naar af
            end;
        end;
    end;
end.
```

Het zingen van de nachtegaal is eigenlijk niet meer dan het afwisselen van wat afbeeldingen waarin de stand van de snavel en de kleur van de keel varieert.

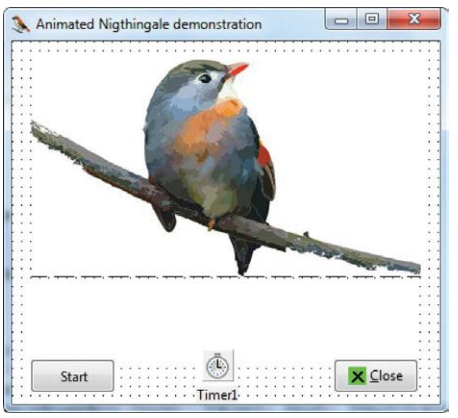
Cursus Pascal

Het doel van deze minicursus is dat je de basisbeginselen leert van het programmeren in Pascal met de ontwikkelomgeving Lazarus en dat je een aantal eenvoudige programma's kunt maken met behulp van componenten.

Deze cursus is geschreven in samenwerking met de Stichting Ondersteuning Programmeertaal Pascal.

Deel 1. De basis
– Wat is Pascal, wat is Lazarus, overzicht van de IDE, voorbeeldprogramma's
Deel 2. Debuggen
Deel 3. Classes
Deel 4. Multimedia
Deel 5. Netwerken, internet
Deel 6. Databases

Het voorbeeldvenster voor het tweede programma. Met de Object Inspector kun je alle namen en teksten aanpassen.



Op het formulier is al te zien dat we hier extra componenten gebruiken. In dit geval een afbeelding- en een timercomponent.
In deze procedure wordt met een vast interval een nieuwe afbeelding van de nachtegaal getoond. Het interval is in te stellen in de Object Inspector: klik op 'Timer1: Timer' en stel bij 'Eigenschappen' een ander 'Interval' in. In bovenstaande code gebruiken we 10 afwisselende afbeeldingen die telkens herhaald worden, in de uitgebreide versie zijn dat er 50 voor een natuurlijkere beweging.

De voorbeelden in dit eerste deel van de Pascal-cursus met Lazarus geven waarschijnlijk genoeg aanknopingspunten om daar zelf mee aan de slag te gaan. In ieder geval hopen we dat het duidelijk is dat je op eenvoudige wijze zelf een programma kunt maken.

In het volgende deel gaan we kijken hoe je fouten in een programma kunt opsporen. Lazarus biedt daar een aantal mooie mogelijkheden voor die je helpen om snel een typfout te achterhalen of de afhandeling van de code te volgen. (nkr)

Softlink 1204102