Delphi im Informatikunterricht

Robert P. Michelic

Einleitung

In den folgenden Beiträgen möchte ich zeigen, wie Delphi sich einerseits als praktikables Werkzeug für den Programmierunterricht verwenden lässt und wie sich andrerseits mit diesem Unterricht auch weitere Inhalte des Informatikunterrichtes verknüpfen lassen.

Als Informatiklehrer ist man immer hin- und hergerissen zwischen Wünschen, möglichst wenig oder möglichst viel zu programmieren; in manchen Klassen hat man Freaks, die während des Unterrichtes am liebsten im PC sitzen würden, andere Klassen betrachten Programmieren als überflüssige geistige Anstrengung, die es weitestgehend zu vermeiden gilt.

Ich unterrichte seit zwanzig Jahren Informatik an einer AHS, es ist eigentlich überflüssig, anzumerken, dass sich der Unterricht, die Unterrichtsmethoden in dieser Zeit vielfach geändert haben. Nicht nur vom Lehrplan herrührend (früher hieß das Fach EDV, im Lehrplan war viel "Informatik" enthalten, nun heißt das Fach Informatik, im Lehrplan findet sich nahezu ausschließlich "EDV"), sondern natürlich auch von den zur Verfügung stehenden Werkzeugen.

Visuelle Entwicklungswerkzeuge wie Delphi haben (im Hinblick auf den Einsatz im Informatikunterricht) wesentliche Vorteile, bergen aber auch gewisse Gefahren:

- Die Ansprüche der Schüler, das Äußere eines Programms betreffend, sind so hoch, dass das Arbeiten mit einfachen Tools, mit denen die Erstellung einer halbwegs brauchbaren Benutzeroberfläche mühsam und langwierig ist, oft zu Frustration und Desinteresse führt. Mit Werkzeugen wie Delphi, wo im Nu ein optisch ansprechendes Programm gestrickt ist, werden die Schüler angeregt und entwickeln selber Ideen, was "ihr" Programm noch alles können sollte.
- Die Zeit, die man sich bei der Erstellung der Benutzeroberfläche spart, kann man für die Inhalte, die Programmlogik und die Algorithmen verwenden. Zweifelsohne anspruchsvoller, dafür aber befriedigender.
- Die Gefahr, die ich sehe: Man kann mit Tools wie Delphi wesentliche Aspekte der Programmierung, die meines Erachtens auch notwendig sind für ein Verständnis der Arbeit, so elegant umschiffen, dass die Schüler wochenlang mit Delphi gearbeitet haben und nicht wissen, was Botschaften sind oder wo die Trennlinie zwischen Windows und Delphi zu ziehen ist.
  Es ist eine Gefahr, eine Versuchung, es bleibt ja jedem überlassen, zu einem passenden Zeitpunkt diese Aspekte in einem Exkurs zu besprechen und zumindest ein Verständnis dafür weiterzugeben.

In den folgenden Beiträgen gehe ich von der Idee aus, dass man mit Delphi im Informatikunterricht Programme entwickeln kann, die neben dem Erwerb von Programmierkenntnissen auch andere Inhalte des Informatikunterrichtes vermitteln. In welcher Form die hier vorgestellten Projekte eingesetzt werden können, hängt von so vielen Parametern ab, dass ich hier keine Vorschläge machen möchte - es reicht vom Einsatz der fix-fertigen Programme bis zum Nachbau einzelner Teile davon. Ich habe die Projekte bewusst auch mit Ideen vollgestopft, so kann man sich heraussuchen, was man will. In der vorliegenden Form sind sie für den Unterricht zweifelsohne zu überladen.

Ich habe bewusst in der Beschreibung fast keine Hinweise zur Programmierung unter Delphi bzw. mit Object-Pascal gegeben, diese Beiträge richten sich daher an Leser, die entweder mit Delphi bereits gearbeitet haben oder genügend allgemeine Programmiererfahrung besitzen.

Teil I: Bits und Bytes mit Delphi veranschaulicht.

Projektbeschreibung

Ziel des Projekts ist es, ein Gefühl dafür zu vermitteln, dass das, was in einzelnen Bytes im Speicher steht, auf ganz unterschiedliche Art interpretiert werden kann: als Zahl in verschiedenen Darstellungen, als Zeichen in verschiedenen Codes, auch als Farbe bzw. Farbkomponente (siehe dazu auch Teil II).

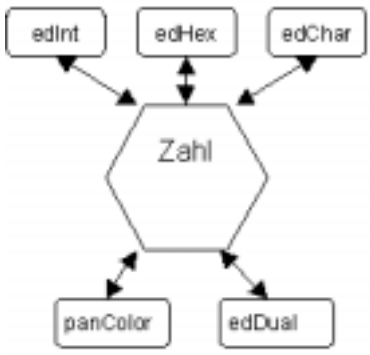
Das hier vorgestellten Programm soll alle Darstellungsarten simultan bieten und auch die Möglichkeit geben, Eingaben in allen Varianten vorzunehmen. Das ist natürlich nicht in jedem Fall nötig, gibt uns aber Gelegenheit, sehr vorteilhaft Eigenschaften (properties) in Delphi auszunutzen.

Neben den Standardumwandlungsalgorithmen, die bereits in Delphi implementiert sind (z.B. IntToHex zum Umwandeln einer Ganzzahl in einen Hexadezimalstring), brauchen wir noch Algorithmen zur Umwandlung von Hexdezimalzahlen in Ganzzahlen und zum Umwandeln von Ganzzahlen in Dualzahlen und umgekehrt.

Unser Hauptformular bekommt eine Eigenschaft (property) Zahl, die mit allen Ein-Ausgabemöglichkeiten der verschiedenen Darstellungen gekoppelt ist. Jede Eingabe ändert den Wert in Zahl, jede Änderung der Zahl wiederum ändert alle angezeigten Werte. Wir müssen nur aufpassen, dass wir uns nicht selber aufrufen.

Schematisch

Erzeugen wir zunächst unser Formular, dem wir die nötigen Komponenten und Beschriftungen geben (Form1 mit den Komponenten edInteger, edHex, edDual, edChar, panColor).



Dann ergänzen wir das Formular um die Eigenschaft Zahl:

```
property Zahl: integer read FZahl write SetZahl;
```

und arbeiten die Set-Routine aus:

```
procedure TForm1.SetZahl(const Value: integer);
begin
  try
    Internal:=true;
    FZahl := Value;
    If not edZahl.Focused then edZahl.Text:=IntToStr(Value);
    If not edHex.Focused then edHex.Text:=IntToHex(Value);
    If not edDual.Focused then edDual.Text:=IntToDual(Value);
    If not edChar.Focused then edChar.Text:=Char(Byte(Value));
    panColor.Color:=Value
  finally
    Internal:=false;
  end
end;
```

Damit ist die eine (leichte) Richtung in unserem Schema implementiert, abgesehen von den Umwandlungsalgorithmen, die weiter unten beschrieben sind. Die Änderungen werden nur in die einzelnen Edit-Komponenten geschreiben, wenn diese nicht gerade fokussiert sind, d.h., wenn wir dort nicht gerade editieren. Die Variable Internal (Typ:Boolean) überwacht, dass wir uns nicht selber aufrufen.

Für die umgekehrte Richtung hängen wir uns in das OnChange-Ereignis der einzelnen Edit-Komponenten, um Zahl immer aktuell zu halten. Zum Beispiel bei edHex.OnChange:

```
procedure TForm1.edHexChange(Sender: TObject);
begin
  if not Internal then try
    Zahl:=HexToInt(edHex.Text)
  except
  end
end;
```

In unserem einfachen Fall soll einfach nichts passieren, wenn die eingegebene Hex-Zahl ungültig ist. (Anm.: Stellen Sie während des Entwickelns die EConvertError-Exceptions ab (Delphi 4) oder verhindern Sie, dass das Programm bei jeder Exception anhält (Delphi 3), da sonst beim Editieren ständig Exceptions generiert werden.)

Analog funktioniert es bei den anderen Feldern.

Zuletzt brauchen wir noch die Umwandlungsalgorithmen, die Delphi nicht als Standard anbietet. Als Beispiel hier IntToDual, die Umwandlung einer Ganzzahl in einen "Dualstring":

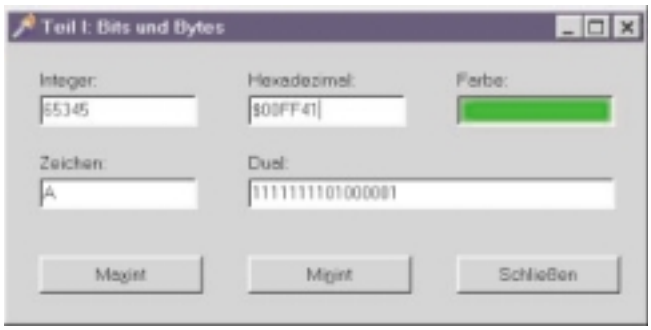
```
function IntToDual(Number:integer):String;
begin
  Result:='';
  while Number<>0 do begin
    if Odd(Number) then Result:='1'+Result
    else Result:='0'+Result;
    Number:=Number div 2;
  end
end;
```

Die Codes der weiteren Umwandlungsalgorithmen finden Sie beim Programmlisting, dort sind auch Varianten als Kommentare enthalten.

In dieser Form enthält das Projekt zwei Schwerpunkte:
1. Inhaltlich: Codes und ihre Interpretation.
2. Programmieren: Umwandlungsalgorithmen.

Fortsetzung folgt.

Screenshot



Programmlisting

```
unit u1;

interface

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

type
```

```
  TForm1 = class(TForm)
    edZahl: TEdit;
    edHex: TEdit;
    edDual: TEdit;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    edChar: TEdit;
    Label5: TLabel;
    Label6: TLabel;
    btnClose: TButton;
    panColor: TPanel;
    btnMaxint: TButton;
    btnMinint: TButton;
    procedure btnCloseClick(Sender: TObject);
    procedure edZahlChange(Sender: TObject);
    procedure edHexChange(Sender: TObject);
    procedure edDualChange(Sender: TObject);
    procedure edCharChange(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure btnMaxintClick(Sender: TObject);
    procedure btnMinintClick(Sender: TObject);
  private
    FZahl: integer;
    Internal: Boolean;
    procedure SetZahl(const Value: integer);
    { Private-Deklarationen }
  public
    { Public-Deklarationen }
    property Zahl:integer read FZahl write SetZahl;
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

{Umwandlungsalgorithmen:}

{Umwandlung der Zahl Number (integer)in einen Dualstring}
{Anm.:
 Statt "if Odd(Number)" wäre auch ein Abfrage der Art
 "if Number and 1 = 1" möglich.
 Statt "div 2" könnte man auch "shr 1" verwenden.}
function IntToDual(Number:integer):String;
begin
  Result:='';
  while Number<>0 do begin
    if Odd(Number) then Result:='1'+Result
    else Result:='0'+Result;
    Number:=Number div 2;
  end
end;

{Umwandlung der Zahl Number (integer)in einen Hex-String}
function IntToHex(Number:integer):String;
var Rest:integer;
begin
  Result:='';
  while Number<>0 do begin
    Rest:=Number mod 16;
    case Rest of
      0..9: Result:=char(Rest+48)+Result;
      10..15: Result:=char(Rest+55)+Result;
    end;
    Number:=Number div 16
  end
end;

{Umwandlung eines Hex-Strings in eine Zahl}
{Der Hex-String kann von einem $-Zeichen eingeleitet werden,
 Groß- oder Kleinbuchstaben ist egal.}
function HexToInt(Hex:String):integer;
var h,i:integer;
begin
  Result:=0;
  if length(Hex)>0 then begin
    if Hex[1]='$' then Hex:=Copy(Hex,2,255);
    for i:=1 to length(Hex) do begin
      case Upcase(Hex[i]) of
        '0'..'9': h:=(Hex[i])-48;
        'A'..'F': h:=(Hex[i])-55;
        else raise EConvertError.Create('Keine gültige Hex-Zahl');
      end;
      Result:=Result*16+h
    end
  end
end;

{Umwandlung eines Dual-Strings in eine Zahl}
function DualToInt(Dual:String):integer;
var i:integer;
begin
  Result:=0;
  for i:=1 to length(Dual) do begin
    case Dual[i] of
      '1': Result:=Result*2+1;
      '0': Result:=Result*2;
      else raise EConvertError.Create('Keine gültige Dual-Zahl');
    end;
  end;
end;

{TForm1}

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

procedure TForm1.SetZahl(const Value: integer);
begin
  if Value<>Zahl then try
    Internal:=true;
    FZahl := Value;
    If not edZahl.Focused then edZahl.Text:=IntToStr(Value);
    If not edHex.Focused then edHex.Text:=IntToHex(Value);
    If not edDual.Focused then edDual.Text:=IntToDual(Value);
    If not edChar.Focused then edChar.Text:=Char(Byte(Value));
    panColor.Color:=Value
  finally
    Internal:=false;
  end
end;

procedure TForm1.edZahlChange(Sender: TObject);
begin
  if not Internal then try
    Zahl:=StrToInt(edZahl.Text);
  except
  end;
end;

procedure TForm1.edHexChange(Sender: TObject);
begin
  if not Internal then try
    Zahl:=HexToInt(edHex.Text);
  except
  end;
end;

procedure TForm1.edDualChange(Sender: TObject);
begin
  if not Internal then try
    Zahl:=DualToInt(edZahl.Text);
  except
  end;
end;

procedure TForm1.edCharChange(Sender: TObject);
begin
  {aufpassen, damit wir keinen leeren String untersuchen}
  if not Internal and (length(edChar.text)>0) then try
    Zahl:=ord(edChar.text[1])
  except
  end
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Randomize;
  Zahl:=random(Maxint); {löscht alle Edit-Controls!}
end;

procedure TForm1.btnMaxintClick(Sender: TObject);
begin
  Zahl:=Maxint;
end;

procedure TForm1.btnMinintClick(Sender: TObject);
begin
  Zahl:=-Maxint-1;
end;

end.
```