Arbeitsmaterial Informatik Klassen 11 und 12

Unterrichtsmaterial für die Verwendung im Informatikunterricht Gymnasium Haldensleben
Erstellt: Andrea Wilke (2004)
Überarbeitete Version 2.1 (2006)

Abb. 1: Ein Tempel innerhalb der Orakelstätte von Delphi"

Inhaltsverzeichnis

0. Hinweise zur Arbeit mit diesen Seiten

Die einzelnen Seiten sind farblich nach folgendem Schema aufgeteilt:

Text ohne eine besondere Hinterlegung bedeutet:

Hier handelt es sich um theoretisches Grundwissen, welches zum Verständnis der Beispiele unbedingt bearbeitet werden sollte.

Diese Tabellen enthalten eine Reihe von wichtigen Angaben, die in den folgenden Beispielen verwendet werden.

Diese Texte bzw. Tabellen enthalten Beispiele für die zuvor behandelten Themen. Diese sollten unbedingt in einem eigenen Delphi-Programm nachgearbeitet werden.

Alle Beispielprogramme stehen Ihnen im Ordner "PROGRAMME" als lauffähige Delphi-Programme zur Verfügung. Zu diesen Programmen gibt es vom jeweiligen Thema einen direkten Link. Um die Programme starten zu können ist es notwendig:

- Eine Delphi-Version (möglichst Delphi 7) auf dem eigenen Rechner zu installieren.
- Das gesamte Projekt (also die Web-Seiten und die Programme) auf die Festplatte zu kopieren.

Falls Sie noch keine Delphi-Version auf Ihrem Rechner installiert haben, finden Sie alle ausführbaren Dateien im Ordner "AUSFÜHRBAR".

Lösungen zu einigen Aufgabenstellungen finden Sie im Ordner "LÖSUNGEN".

Weitere Programme mit themenübergreifenden Beispielen finden Sie im Ordner "PROGRAMME2".

PROGRAMM	Dieses Zeichen am Ende eines Kapitels führt zu lauffähigen Delphi-Projekten.
LSG	Dieses Zeichen im Aufgabenteil zeigt an, dass für die Programmieraufgabe eine Lösung zur Verfügung steht.

1. Algorithmen und Algorithmenstrukturen

1.1. Der Begriff des Algorithmus

1.1.1. Beispiele für Algorithmen

Algorithmen begegnen Ihnen jeden Tag. Sie werden für die verschiedensten Aufgabenstellungen benutzt.

Beispiel Kochrezept:

2 kg Kartoffeln und 1 mittelgroße Zwiebel schälen. Beide Zutaten gründlich waschen und dann reiben. Anschließend mit 4 Eiern und 60g Mehl verrühren und mit etwas Salz würzen. Etwas Öl in einer Pfanne erhitzen, dann den Teig esslöffelweise hineingeben, flach drücken und von beiden Seiten braun und knusprig backen.

Rechenablaufplan für den Taschenrechner:

Zum Berechnen des Terms (4 + 9)(18 - 7) wie folgt vorgehen:

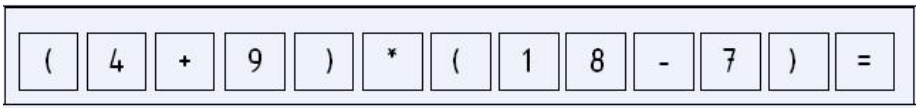


Abb. 2: Rechenablaufplan

Bedienungsanleitung für einen Videorecorder:

- Videotextinfo zur Sendung anzeigen
- Markieren Sie die Sendung im EPG mit Hilfe der Pfeiltasten auf / ab.
- Durch Drücken der Taste Videotext wird nun die angezeigte Videotexttafel auf dem Bildschirm dargestellt.
- Durch erneutes Drücken der Taste Videotext kehren Sie wieder in die EPG-Tafel zurück.

Weitere Beispiele:

- Montageanleitungen
- Lösen der quadratischen Gleichung
- Sortieren von 100 Adressen

1.1.2. Algorithmusbegriff

Damit stellt sich die Frage nach den Gemeinsamkeiten dieser doch sehr verschiedenartigen Algorithmen:

- Alle Algorithmen verarbeiten Eingabegrößen zu Ausgabegrößen:
  - Beim Kochrezept werden die Zutaten zu Kartoffelpuffern verarbeitet.
  - Beim Rechenablaufplan werden die eingegeben Zahlen zu einem Ergebnis verarbeitet.
  - ...
- Alle Algorithmen beschreiben eine eindeutige Vorgehensweise.
- Alle Algorithmen sind nach einer bestimmten Anzahl von Schritten abgearbeitet.
- Der Algorithmus kann beliebig oft wiederholt werden.

Ein Algorithmus ist eine endliche Folge eindeutig ausführbarer Anweisungen zur Lösung einer Klasse von Problemen. Er gibt an, in welcher Art Eingabegrößen schrittweise zu Ausgabegrößen umgewandelt werden.

1.2. Eigenschaften von Algorithmen

Aus der in 1.1. kennen gelernten Definition eines Algorithmus lassen sich dessen allgemeinen Eigenschaften ableiten:

Eindeutigkeit:

Mit jeder Anweisung ist auch die nächstfolgende Anweisung festgelegt.
Das heißt, dass bei gleichen Eingabegrößen bei wiederholter Abarbeitung des Algorithmus das gleiche Ergebnis erreicht wird.

Endlichkeit:

Ein Algorithmus besteht aus endlich vielen Anweisungen endlicher Länge.
Er ist nach endlich vielen Schritten beendet.

Ausführbarkeit:

Jede Anweisung des Algorithmus ist für den Ausführenden (Prozessor) verständlich und ausführbar.

Allgemeingültigkeit:

Ein Algorithmus muss auf alle Aufgaben des gleichen Typs (Klasse von Aufgaben) anwendbar sein und bei richtiger Anwendung stets zum gesuchten Resultat führen.

Beispiele für Problemstellungen, die sich nicht algorithmisch lösen lassen:

- Bestimmen aller Primzahlen. (widerspricht der Endlichkeit)
- Schreiben von 15 NP im nächsten Mathetest (widerspricht Eindeutigkeit und Allgemeingültigkeit)
- Berechnen der Lottozahlen für nächsten Samstag (wäre auch zu schön)

1.3. Darstellungsformen für Algorithmen

Algorithmen lassen sich auf verschiedene Weise darstellen. Einige sind Ihnen dabei sicherlich geläufig.

Verbale Formulierung:

z.B. Kochrezepte
Bei der verbalen Formulierung werden die Algorithmen in einer natürlichen Sprache (z.B. Deutsch oder Englisch) abgefasst. Diese sind meist ohne weitere Kenntnisse ausführbar.

Rechenablaufplan:

z.B. Rechenablaufpläne für einen Taschenrechner
Diese Ablaufpläne sind meist nur für ein spezielles Gerät anwendbar. Die speziellen Funktionsweisen sind zu beachten.

Nummerierte Skizzen:

z.B. Montageanleitungen
Mit Hilfe von grafischen Darstellungen werden die einzelnen Schritte des Algorithmus für den Anwender aufgezeigt.

Programmablaufpläne:

Hier werden die einzelnen Programmschritte mit Hilfe von Programmlinien dargestellt. Die aufeinander folgenden Schritte werden meist durch Pfeile gekennzeichnet. Die Schritte können verbal oder auch abstrakt aufgeschrieben werden. Programmablaufpläne werden heute meist nur noch zur Erläuterung sehr komplexer Programme eingesetzt. Dabei wird nicht der einzelne Algorithmus dargestellt, sondern es werden größere Programmabschnitte algorithmisiert.

BEISPIEL:

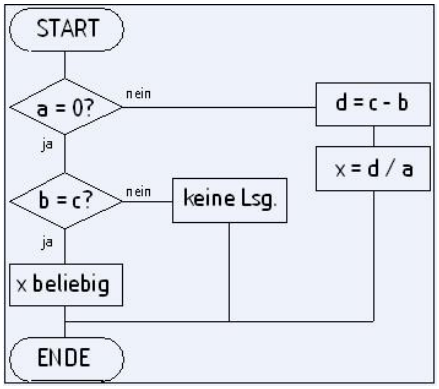


Abb. 3: Lösung der Gleichung ax + b = c

Struktogramme:

Struktogramme werden heute für die Vorüberlegungen beim Programmieren am häufigsten benutzt. Sie sind einheitlich normiert. Mit der Darstellung von Struktogrammen werden wir uns im Unterricht noch genauer befassen.

BEISPIEL:

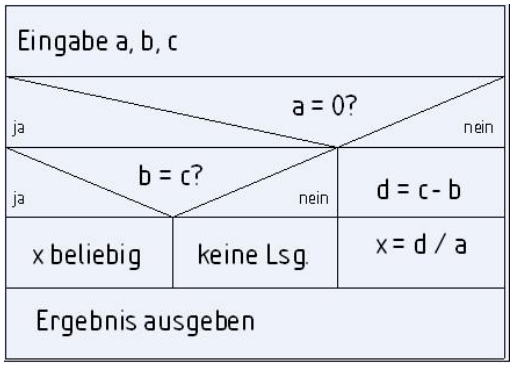


Abb. 4: Struktogramm: ax + b = c

Computerprogramm:

Computerprogramme sind Sprachen die vom Betriebssystem eines Computers verstanden werden. Je nach Abstraktionsgrad der benutzten Programmiersprache können diese mehr oder weniger gut auch vom Menschen verstanden werden.

BEISPIEL 1: Delphi 7 - Programm zur Lösung der Gleichung ax + b = c:

```
procedure tform1.button1Click (Sender : TObject);
var a, b, c, d, x : real;
begin
  a := strtofloat (edit1.text);
  b := strtofloat (edit2.text);
  c := strtofloat (edit3.text);
  if a = 0 then
    begin
      if b = c then
        edit4.text := 'x beliebig'
      else
        edit4.text := 'keine Lösung';
    end
  else
    begin
      d := c - b;
      x := d / a;
      edit4.text := floattostr (x);
    end;
end;
```

BEISPIEL 2: C++ - Programm zur Lösung der Gleichung ax + b = c:

```
int main ()
double a, b, c, d, x;
{cout << "Eingabe a:"
 cin  >> a;
 cout << "Eingabe b:"
 cin  >> b;
 cout << "Eingabe c:"
 cin  >> c;
 if (a == 0)
   {
     if (b == c)
       {
         cout << "x beliebig" << endl;
       }
     else
       {
         cout << "keine Lösung" << endl;
       }
   }
 else
   {
     d = c - b;
     x = d/a;
     cout << x << endl;
   }
 return 0;
}
```

1.4. Algorithmenstrukturen

1.4.1. Anweisungen und Sequenzen

Algorithmen bestehen aus einer Folge eindeutig ausführbarer Anweisungen...

Diese einzelnen Anweisungen werden sehr oft vom Programmbeginn bis zum Programmende abgearbeitet (z.B. bei einem Kochrezept). Treten mehrere aufeinander folgende Anweisungen auf, dann spricht man von einer Programmsequenz. Programmsequenzen werden in Struktogrammen durch aufeinander folgende Rechtecke dargestellt. Das Programm selbst wird von oben nach unten ausgeführt.

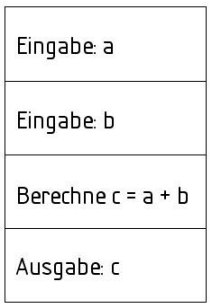


Abb. 5: Beispiel für eine Sequenz

1.4.2. Verzweigungen

Nicht in jedem Fall ist die Ausführung des Programms so linear abarbeitbar. Bereits bei der Division zweier Zahlen kann eine einzelne Sequenz zu Programmfehlern führen. Da die Division durch Null nicht definiert ist, kann diese auch von einem Computer nicht ausgeführt werden. Es muss also sichergestellt werden, dass bei einem Divisor Null eine für den Anwender aussagekräftige Fehlermeldung angezeigt wird. Dies erreichen wir durch die Nutzung von Programmverzweigungen. Bei den Programmverzweigungen werden drei Arten unterschieden:

- Einfache Alternative
- Vollständige Alternative
- Fallauswahl

Die einfache Alternative

Bei einer einfachen Alternative werden die darauf folgenden Anweisungen nur dann abgearbeitet, wenn eine zuvor aufgestellte Bedingung mit "JA" (Wahr) beantwortet wird. Ansonsten werden die Anweisungen übersprungen.

BEISPIEL:

Aus einer Zahl soll die Quadratwurzel berechnet werden. Für den Fall, dass eine negative Zahl eingegeben wurde, wird zunächst der Betrag dieser Zahl ermittelt.

Struktogramm für das Beispiel:

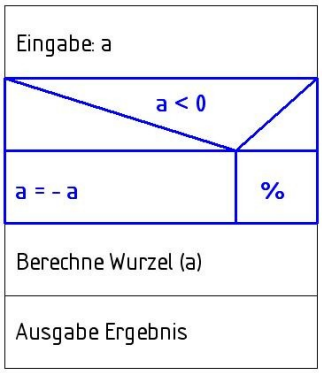


Abb. 6: Einfache Alternative

Die vollständige Alternative

Bei der vollständigen Alternative wird in Abhängigkeit von der Bedingung der eine oder der andere nachfolgende Weg eingeschlagen. Es werden also keine Anweisungen übersprungen sondern von der Bedingung abhängige nachfolgende Anweisungen ausgeführt.

BEISPIEL:

Im Einführungsbeispiel sollte die Division ausgeführt werden. Für den Fall, dass der Divisor Null ist, soll eine Fehlermeldung ausgegeben werden, ansonsten soll der Quotient berechnet werden.

Struktogramm für das Beispiel:

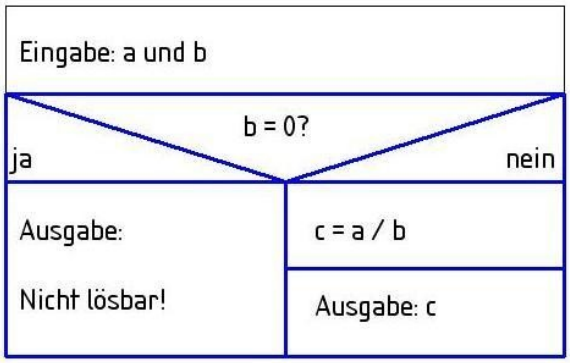


Abb. 7: Vollständige Alternative

Die Fallauswahl

Mit einer Fallauswahl kann man, in Abhängigkeit von einer Bedingung, zwischen mehreren (> 2) nachfolgenden Anweisungen auswählen. Der auszuwertende Ausdruck muss dabei (in der Regel) einen ganzzahligen Ausdruck zurückgeben.

BEISPIEL:

Bei einer Klassenarbeit konnten insgesamt 50 Punkte erzielt werden. Für jede Zensur soll der Punktebereich ausgegeben werden (zur Vereinfachung nutzen wir hier die Zensuren 1 bis 6).

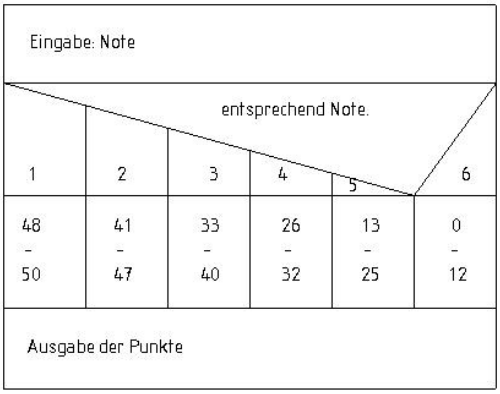


Abb. 8: Fallauswahl

1.4.3. Wiederholungen

Wiederholungen (Zyklen, Iterationen, Schleifen) werden immer dann eingesetzt, wenn ein bestimmter Teil eines Algorithmus mehrmals wiederholt werden muss. Zyklen sind dadurch gekennzeichnet, dass diese bis zu einer Abbruchbedingung immer wieder ausgeführt werden. Auch für solche Zyklen stehen wieder mehrere Arten zur Verfügung:

- Zählschleife
- Wiederholung mit vorhergehender Prüfung der Abbruchbedingung (abweisende Schleife)
- Wiederholung mit nachfolgender Prüfung der Abbruchbedingung (nichtabweisende Schleife)

Zählschleifen

Zählschleifen können immer dann eingesetzt werden, wenn die Anzahl der Schleifendurchläufe bereits vor Beginn der ersten Schleifenabarbeitung bekannt ist. Die Zählschleife ist ein Spezialfall der abweisenden Schleife.

BEISPIEL:

Es ist ein Algorithmus zu entwickeln, mit dem die ersten 100 natürlichen Zahlen summiert werden. Hier ist bereits am Anfang bekannt, dass der Schleifenkörper genau 100-mal durchlaufen werden muss. Der Einsatz einer Zählschleife bietet sich damit an.

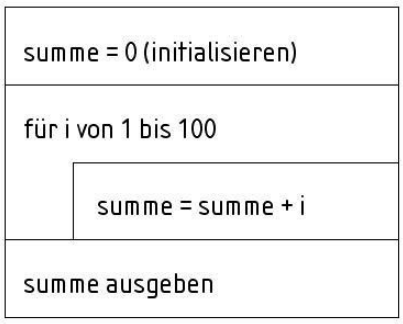


Abb. 9: Zählschleife

Nicht immer ist vor der Schleifenausführung die Anzahl der Schleifendurchläufe bekannt. Wenn z.B. Zahlen so lange addiert werden sollen, bis eine bestimmte vorgegebene Summe erreicht ist, kann mit einer Zählschleife nur sehr umständlich gearbeitet werden. In solchen Fällen können Schleifen mit Abbruchbedingungen eingesetzt werden. Bei diesen Schleifenstrukturen ist es aber besonders wichtig, dafür zu sorgen, dass die Abbruchbedingung auch nach endlich vielen Schritten erfüllt wird. Das Programm gerät ansonsten in eine Endlosschleife.

Abweisende Schleifen

Bei abweisenden Schleifen wird die Abbruchbedingung vor dem ersten Schleifendurchlauf geprüft. Es kann daher durchaus sein, dass der Schleifenkörper überhaupt nicht durchlaufen wird.

Nichtabweisende Schleifen

Bei nichtabweisenden Schleifen wird die Abbruchbedingung erst nach dem Schleifendurchlauf geprüft. Diese Schleifen werden also in jedem Fall mindestens einmal durchlaufen.

BEISPIEL:

Es sollen die ersten n natürlichen Zahlen so lange addiert werden, bis die Summe 100 erreicht oder übertroffen wird. Die letzte summierte Zahl soll ausgegeben werden.

Die Problemstellung kann sowohl mit Hilfe einer abweisenden als auch einer nichtabweisenden Schleife gelöst werden.

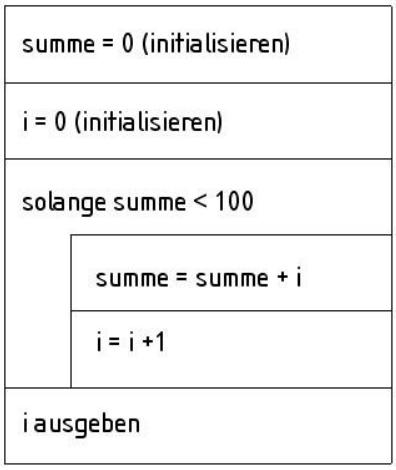


Abb. 10: abweisende Schleife

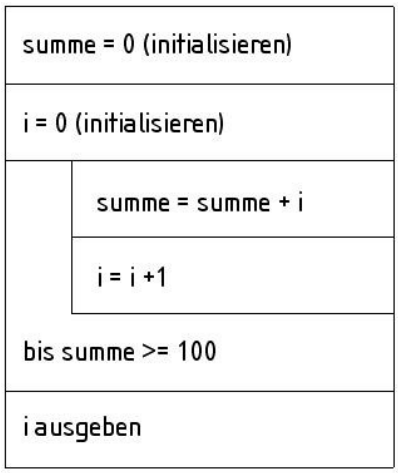
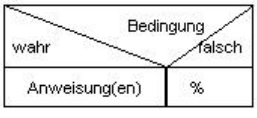
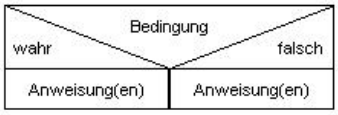
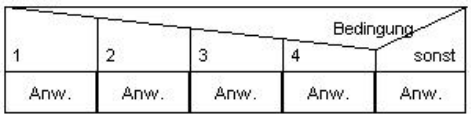
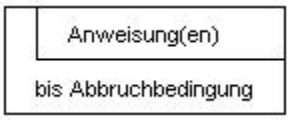


Abb. 11: nichtabweisende Schleife

1.4.4. Zusammenfassung der Kontrollstrukturen

Verzweigungen (Alternativen)		
einfache Alternative	vollständige Alternative	Fallauswahl
		
Wiederholungen (Zyklen, Schleifen)		
Zählschleife	abweisende Schleife	nichtabweisende Schleife
		

1.4.5. Unterprogrammaufrufe

In komplexen Algorithmen (speziell Computerprogramme) werden häufig Unterprogramme zum eigentlichen Programm verwendet. Diese Unterprogramme setzt man meist dann ein, um das Programm insgesamt übersichtlich zu strukturieren und damit ein einzelner Programmteil nicht zu lang wird.

Abb. 18: Struktogrammsymbol für einen Unterprogrammaufruf

1.5. Arbeiten mit Struktogrammen

1.5.1. Arbeiten mit Struktogrammen

Vor der Erstellung eines jeden Programms sollten zunächst entsprechende Struktogramme, bei größeren Programmen zusätzlich ein Strukturbaum (Programmablaufplan) angefertigt werden. Für das Erstellen von Struktogrammen sind einige Regeln zu beachten:

- Die Größe eines Struktogramms ist immer auf eine Seite (A4) beschränkt.
- In Struktogrammen gibt es immer nur einen Eingang und einen Ausgang. Dies gilt auch für die einzelnen Grundsymbole.
- Der Programmablauf erfolgt grundsätzlich von oben nach unten.
- Die Grundsymbole können ineinander geschachtelt und aneinander gereiht werden.
- Alle Programmverzweigungen laufen an einer Stelle wieder zusammen.
- Bei der Entwicklung von Struktogrammen ist das Prinzip der schrittweisen Verfeinerung anzuwenden (TOP-DOWN).

1.5.2. Der Struktogrammeditor

Struktogramme müssen nicht aufwendig mit Grafikprogrammen erstellt werden. Dem Arbeitsmaterial liegt ein einfacher Struktogrammeditor bei. Dieser kann zum Erstellen der Struktogramme genutzt werden.

Abb. 19: Symbol des Struktogrammeditors

Aufbau des Struktogrammeditors:

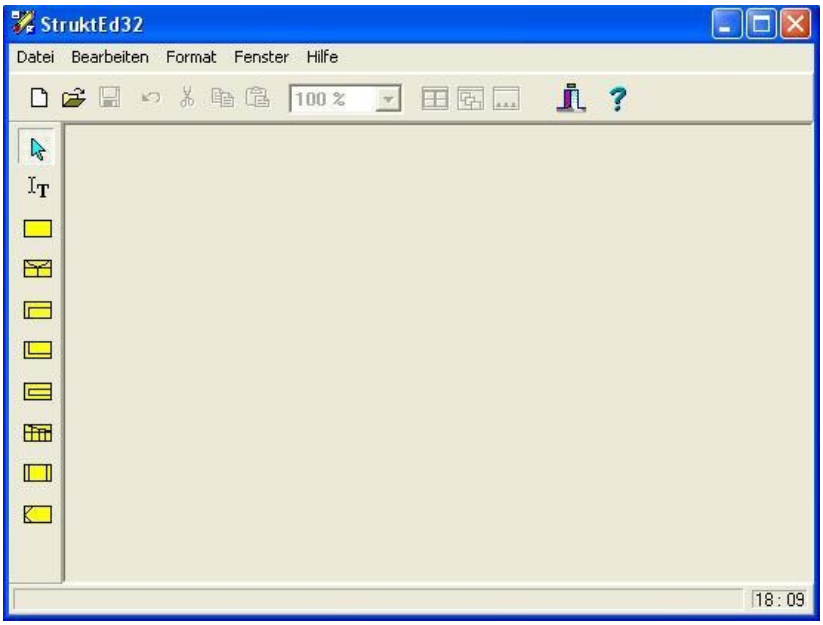


Abb. 20: Ansicht nach dem Start

Erstellen Sie zunächst mit Datei / Neu ein neues Struktogramm. Speichern Sie dieses unter einem aussagekräftigen Namen.

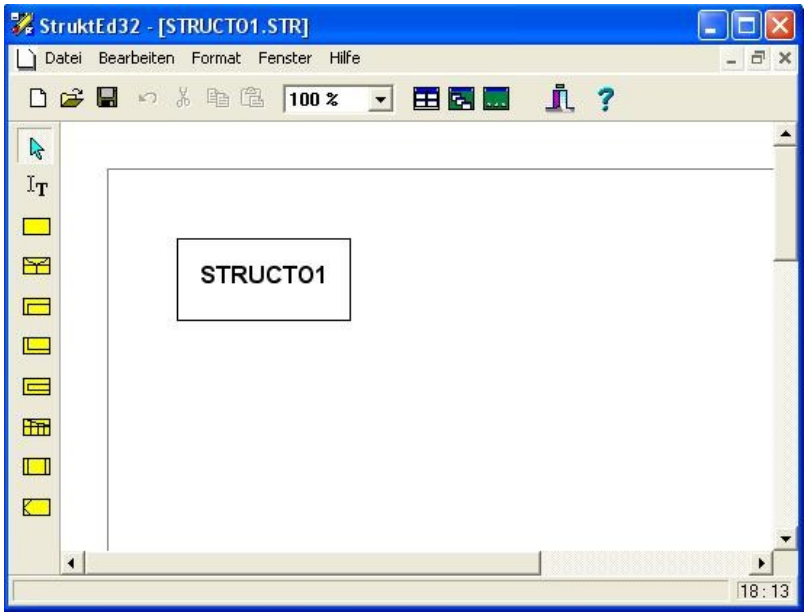


Abb. 21: Ansicht nach Neu

Klicken Sie anschließend auf das TEXT-Symbol.

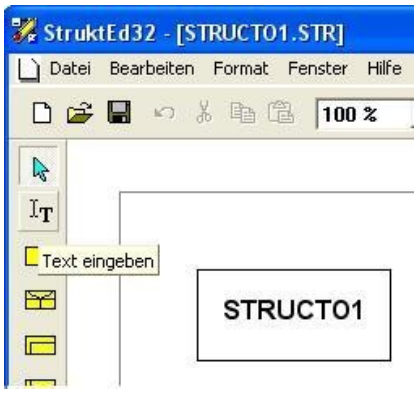


Abb. 22: Textsymbol

Und dann auf das bereits vorhandene Kästchen.
Sie können jetzt Standardeinstellungen für Ihr Struktogramm eingeben und den Struktogrammtitel festlegen.

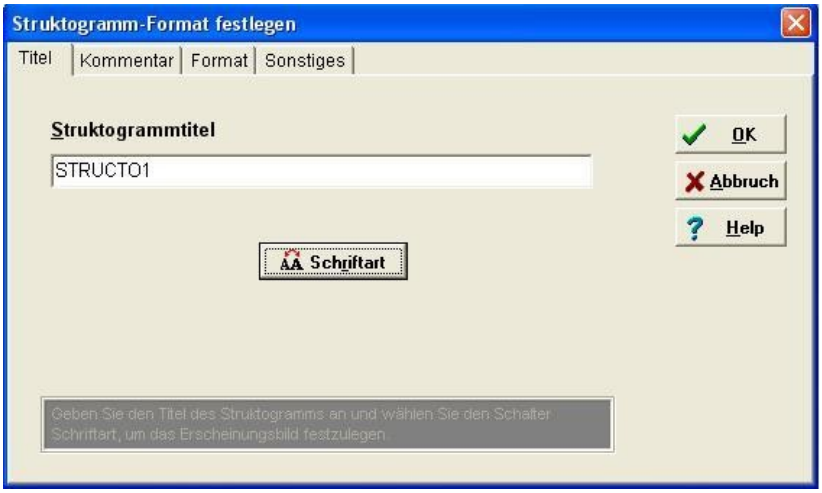


Abb. 23: Standardeinstellungen und Titel

Um mit dem Struktogrammeditor zu arbeiten, klicken Sie ein gewünschtes Symbol an und legen dieses an die benötigte Position. Klicken Sie dann auf das Textsymbol und dann auf das Struktogrammsymbol. Sie können jetzt den Struktogrammtext eingeben. Zum Löschen eines Symbols müssen Sie zunächst den Pfeil auswählen und dann das zu löschende Symbol damit markieren.

Abb. 24: Markiertes Symbol

Um ein Struktogramm in einem Textverarbeitungsprogramm einfügen zu können, klicken Sie zunächst auf Bearbeiten / Grafik kopieren. Sie können die Kopie dann problemlos z.B. in PAINT einfügen und als BMP oder JPG Datei abspeichern.

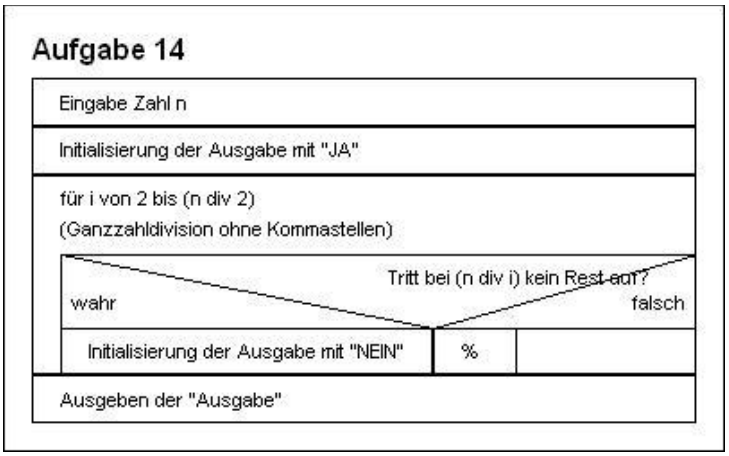
Fragen und Aufgaben
1. Algorithmen und Algorithmenstrukturen

Theoretische Grundlagen

1. Geben Sie eine mögliche Definition eines Algorithmus an.
2. Nennen und erläutern Sie die Eigenschaften von Algorithmen
3. Nennen Sie 5 Problemstellungen, die sich nicht algorithmisch lösen lassen und begründen Sie.
4. Nennen Sie die Darstellungsformen für Algorithmen.
5. Welche Kontrollstrukturen können bei der Arbeit mit Algorithmen unterschieden werden?
6. Worauf ist bei der Arbeit mit abweisenden und nichtabweisenden Schleifen zu achten?
7. Geben Sie die Struktogrammsymbole für die einzelnen Kontrollstrukturen an.
8. Welche der Aussagen zu den Schleifenstrukturen ist richtig. Begründen Sie.
    a. Eine Zählschleife kann jederzeit durch eine abweisende Schleife ersetzt werden.
    b. Eine abweisende Schleife kann jederzeit durch eine Zählschleife ersetzt werden.
    c. Eine Zählschleife kann jederzeit durch eine nichtabweisende Schleife ersetzt werden.

Praktische Aufgaben

9. Entwickeln Sie ein Struktogramm zur Lösung der Gleichung $ax + b = cx + d$.
10. Welche Spezialfälle können bei der Gleichung unter 9. auftreten. Ergänzen Sie das Struktogramm entsprechend.
11. Geben Sie ein mögliches Struktogramm an, mit dem die ersten 10 Quadratzahlen summiert werden.
12. Ändern Sie das Struktogramm zur Berechnung der Summe der ersten 100 natürlichen Zahlen mit einer nichtabweisenden Schleife um.
13. Entwickeln Sie ein Struktogramm zur Lösung der quadratischen Gleichung $x^2 + px + q = 0$. Gehen Sie auf mögliche Spezialfälle in Abhängigkeit von der Diskriminante ein.
14. Interpretieren Sie das nachfolgende Struktogramm. Testen Sie dieses dazu mit verschiedenen Zahlen (z.B. von 1 bis 5).



15. Schreiben Sie das Struktogramm aus Aufgabe 14 mit einer nichtabweisenden Schleife um.
16. Zur Umwandlung von ganzen Dezimalzahlen in Binärzahlen haben Sie das Divisionsverfahren kennen gelernt. Entwickeln Sie ein Struktogramm, zur Umwandlung einer solchen Dezimalzahl in eine Binärzahl.
17. Entwickeln Sie ein Struktogramm zur Lösung der allgemeinen quadratischen Gleichung. Gehen Sie auf alle möglichen Spezialfälle der Variablen a, b, c ein.

2. Programmiersprachen

2.1. Grundlegende Begriffe

SPRACHE (allgemein):

Eine SPRACHE über einem Alphabet (hier eine Menge von Zeichen und Symbolen) ist eine beliebige Menge von Wörtern über diesem Alphabet. Die Wörter sind dabei Zeichenketten, die durch Aneinanderreihen von Zeichen des Alphabets entstehen.

SYNTAX:

Die SYNTAX einer Sprache ist die Menge aller Regeln, nach denen zulässige Sätze in dieser Sprache gebildet werden können. Sie beschreibt die Struktur der Sprache und entspricht damit der Grammatik einer natürlichen Sprache.

SEMANTIK:

Die SEMANTIK ist die Lehre von der Bedeutung einer Sprache. Bei einer Programmiersprache beschreibt die Semantik, was während der Ausführung eines Programms oder eines Programmteils geschieht. Dabei werden auch die Wechselwirkungen berücksichtigt.

PROGRAMMIERSPRACHE:

Eine Programmiersprache ist eine Sprache zur Formulierung von Algorithmen und Datenstrukturen für die Abarbeitung auf einem Computer.
Je nach dem Grad, mit der die Hardware bei der Programmierung beachtet werden muss, kann man Programmiersprachen in:

- Maschinensprachen,
- Assemblersprachen und
- höhere Programmiersprachen untergliedern.

Voraussetzung für die Abarbeitung von Algorithmen durch Computer ist, dass die Algorithmen und notwendigen Daten in einer für den Computer interpretierbaren Form zur Verfügung gestellt werden.
Menschen benutzen unterschiedliche Formen von Sprachen zur Kommunikation. Sprachen können geschrieben, gelesen, gesprochen, gehört und auch gezeichnet werden. Zur Kommunikation mit dem Computer müssen auch heute noch spezielle Sprachen, die Programmiersprachen, verwendet werden, da Computer natürliche Sprachen nicht verstehen.
Moderne Programmiersprachen verwenden ebenso wie natürliche Sprachen Wörter, haben grammatische Regeln, eine Syntax und eine Interpunktion. In der Regel sind Programmiersprachen aber stark spezialisiert und beschränkt. Da Computer die Sprachen nicht verstehen, sondern nur interpretieren, müssen diese Sprachen eindeutig definiert sein.
Computer verstehen nur Zahlen. Daher muss dem Computer Software zur Verfügung gestellt werden, die die Anweisungen der Programmiersprache in für ihn verständliche Zahlenfolgen übersetzt. Diese Programme werden entsprechend ihrer Arbeitsweise als Compiler oder Interpreter bezeichnet.
Je mehr eine Programmiersprache von der maschinellen Realisierung der durch sie beschreibbaren Algorithmen abstrahiert, je höher ist ihr Sprachniveau.

2.2. Einteilung der Sprachen

Sprachen der 1. Generation:

Als Sprachen der ersten Generation werden die MASCHINENSPRACHEN bezeichnet. Ihr Merkmal war die Verwendung absoluter Adressen im Hauptspeicher. Sie werden heute nur noch für Spezialanwendungen, wie z.B. ROM-BIOS oder Steuerungsprogrammierung eingesetzt. Sie benötigen dabei nur minimalen Speicherplatz.

Sprachen der 2. Generation:

ASSEMBLERSPRACHEN werden als Sprachen der 2. Generation bezeichnet. Sie arbeiten im Wesentlichen mit den Befehlen des Computersystems, für das sie definiert wurden, und erlauben den direkten Zugriff auf alle Komponenten des Computers. Mit diesen Sprachen wurde die Verwendung von Operatoren (+, -, *, /) und relativer Adressen eingeführt. Zur Vereinfachung der Programmierung wurden auf der Basis der Assemblersprachen Makrosprachen entwickelt.

Sprachen der 3. Generation:

Als Sprachen der 3. Generation werden die PROZEDURALEN PROGRAMMIERSPRACHEN (problemorientierte Sprachen) bezeichnet. Sie dienen der rechnerunabhängigen Codierung prozeduraler Algorithmen und abstrahieren von der zugrunde liegenden Rechnerstruktur. Einige Vertreter sind FORTRAN, COBOL, ALGOL, BASIC, C und Pascal. Diese Sprachen sind besonders für Computer der von-Neumann-Architektur geeignet.

Sprachen der 4. Generation:

Als Sprachen der 4. Generation werden DATENBANKSPRACHEN bezeichnet. Sie erlauben die Definition von Datenbanken und die Ausführung einfacher Operationen über diese Datenbanken. Weiterhin werden der 4. Generation noch die Werkzeuge zur Erzeugung und Gestaltung von graphischen Nutzeroberflächen zugeordnet.

Sprachen der 5. Generation:

DEKLARATIVE PROGRAMMIERSPRACHEN werden als Sprachen der 5. Generation bezeichnet. Sie legen den Schwerpunkt auf die Beschreibung des Problems und überlassen dem System die Generierung einer effektiven Lösung.

2.3. Sprachübersetzung

Damit die von uns geschriebenen Programme durch den Computer ausgeführt werden können, müssen diese zunächst in eine vom Betriebssystem und damit vom Prozessor des Computers ausführbare Form umgewandelt werden. Das bedeutet nichts anderes, als das die Programme in einen Maschinencode mit den Befehlen des Prozessors umgewandelt werden müssen. Diese Übersetzung kann (abhängig von der Entwicklungsumgebung / Programmiersprache) auf verschiedene Art erfolgen.
Für die Übersetzung werden entweder Interpreter oder Compiler eingesetzt. Leider werden diese beiden Begriffe in der Literatur nicht immer ganz sauber benutzt.

2.3.1. Interpreter

Ein Interpreter arbeitet nach dem folgenden grundsätzlichen Schema:
- Lies die erste Anweisung.
- Übersetze (falls möglich) die Anweisung in den Maschinencode.
- Arbeite die Anweisung ab.
- Lies die nächste Anweisung.
- ...

Die Abarbeitung durch den Interpreter erfolgt bis zum Programmende oder bis ein Fehler entdeckt wird.

Vorteil:

- Nicht benötigte Befehle von Alternativen müssen nicht übersetzt werden.

Nachteil:

- Befehle von Schleifenstrukturen werden mehrfach übersetzt.
- Auf dem ausführenden Computer muss der Interpreter stets ebenfalls vorliegen.

Typisches Beispiel: VisualBASIC

2.3.2. Compiler

Ein Compiler arbeitet nach dem folgenden Prinzip:
- Lies das Programm zeilenweise ein.
- Übersetze jede Zeile (falls möglich) in den Maschinencode.
- Führe das Programm aus (falls die Übersetzung bis zum Programmende ausgeführt werden konnte).

Der Compiler übersetzt also zunächst das vollständige Programm und arbeitet es anschließend ab. Treten Fehler im Programmquelltext auf, dann werden diese bereits vor der Programmabarbeitung erkannt und ggf. angezeigt. Dies trifft insbesondere auf syntaktische Fehler zu.
Logische Fehler (wie z.B. Endlosschleifen) können von einem Compiler nicht erkannt werden (es wird auch in Zukunft einen solchen Compiler nicht geben, da das Finden von Endlosschleifen nicht algorithmisch gelöst werden kann).
Häufiges Ergebnis einer Compilierung ist eine EXE-Datei, die auch auf Computern ohne den Compiler ausgeführt werden können.

Vorteil:

- Schleifenstrukturen werden nur einmal übersetzt.
- Der Compiler muss auf dem ausführenden Computer nicht unbedingt vorliegen.

Nachteil:

- Nicht benötigte Programmteile werden ebenfalls übersetzt.

Typisches Beispiel: Delphi7, C++

2.4. Wichtige Programmiersprachen

FORTRAN (FORmular TRANslator):

Die erste höhere Programmiersprache war FORTRAN. Sie wurde in den Jahren 1954 bis 1956 bei IBM für wissenschaftliche Berechnungen geschaffen. 1958 entstand die erste Weiterentwicklung - FORTRAN II. Im Jahre 1962 wurde FORTRAN IV als Quasi-Standard entwickelt und blieb über ein Jahrzehnt die am weitesten verbreitete FORTRAN - Version. Nach der Standardisierung 1966 wurde diese Version zu FORTRAN-66 weiterentwickelt. FORTRAN ist einfach und leicht erlernbar. Es werden sehr effiziente Zielcodes erzeugt. Sie ist bis heute die dominierende Sprache der Naturwissenschaftler.
Zu den wesentliche Eigenschaften, die mit FORTRAN in die Programmierung eingeführt wurden, gehörten

- Felder,
- Schleifen, die durch Indizes gesteuert werden konnten und
- die verzweigende IF-Anweisung.

COBOL (COmmon Busines Orienented Language)

COBOL wurde 1959-1960 vom U.S. Verteidigungsministerium entwickelt. Diese Sprache wurde schnell von Industrieunternehmen und Banken für umfangreiche Belegverarbeitung und andere kaufmännische Anwendungen übernommen. COBOL zählt immer noch zu den am weitesten verbreiteten Programmiersprachen. Mit dieser Sprache werden große Datenmengen leicht handhabbar (Massendatenverarbeitung). Die Programme sind auch für einen Nichtfachmann gut lesbar. Beim Lösen wissenschaftlich-technischer Probleme ist ein hoher Programmieraufwand erforderlich. Die Sprache besitzt einen großen Umfang an Sprachelementen.
Die wesentlichen Eigenschaften, die COBOL zum Sprachentwurf beigesteuert hat, sind:

- die RECORD-Struktur zur Organisation von Daten,
- die Trennung der Datenstrukturen vom Ausführungsteil eines Programms und
- vielseitige Formatierungsmöglichkeiten für die Ausgabe.

ALGOL (ALGOrithmic Language)

ALGOL wurde in den Jahren 1958 bis 1960 von einer internationalen Arbeitsgruppe als Sprache für die Beschreibung und Programmierung vorrangig von Algorithmen der numerischen Mathematik konstruiert. Damit wurde erstmals eine Sprache auf der Grundlage einer formalen Grammatik geschaffen. ALGOL ermöglicht eine Blockstruktur der Programme und enthält damit Elemente der strukturierten Programmierung. Sie zeichnet sich durch sprachliche Geschlossenheit und Transparenz aus. Nachteilig sind die mangelhafte Gestaltung der Dateneingabe und -ausgabe sowie das Fehlen der Zeichenkettenverwaltung.
Für die Sprachentwicklung wurden mit ALGOL folgende Konzepte eingeführt:

- Formatfreiheit,
- strukturierte Anweisungen,
- begin-end-Blöcke,
- Typdeklarationen für Variable,
- Rekursionen und
- call-by-value-Parameter (Wertparameter).

Simula67

Simula67 wurde im Zeitraum von 1965-1967 von Nygaard und Dahl am Norwegischen Computercenter in Oslo entwickelt. Sie basiert auf Simula I und enthält ALGOL60 als Teilmenge. Diese Sprache wurde als Spezialsprache für Simulationen entworfen. Mit der Einführung des Klassenkonzepts wurde diese Sprache zur ersten objektorientierten Programmiersprache.

BASIC (Beginners All-purpose Symbolic Instruction Code)

BASIC ist im Jahre 1963-1965 am Darthmouth College als Lehrsprache für Anfänger entstanden. Sie ist sehr einfach und bestand am Anfang nur aus 10 Schlüsselwörtern. BASIC erwies sich als besonders geeignet für leistungsschwache Home- und Personalcomputer (auch ohne Diskettenlaufwerk). Ein weiteres Einsatzgebiet ist der Einsatz in rechnergesteuerten Messgeräten.

Mit BASIC ist eine günstige Gestaltung des Mensch-Maschine-Dialoges möglich. Sie ist die am weitesten verbreitete Interpretersprache. Es gibt eine Vielzahl von Sprachdialekten, die den Austausch von Programmen zwischen verschiedenen Computern erschweren. Erweiterte BASIC - Versionen umfassen etwa 200 verschiedene Grundbefehle. Moderne BASIC - Versionen sind BASI, BASICA und GWBASIC. Die Bedeutung in der Wirtschaft ist rückläufig. Die Nutzung von BASIC verleitet zur unstrukturierten Programmierung und zur Schaffung von unübersichtlichen Programmen.
Mit der Integration von BASIC als Erweiterungssprache in die Microsoft-Office-Produkte nimmt die Bedeutung stark zu. Die verwendete Version ist Visual Basic for Applications und im Lieferumfang von MS-EXCEL enthalten. Der Nutzer hat z.B. die Möglichkeit eigene Funktionen und Dialoge zu programmieren und unter der EXCEL-Oberfläche zu verwalten.

PASCAL

Die Programmiersprache PASCAL ist nach dem Mathematiker Blaise Pascal benannt und wurde 1971 an der ETH Zürich von N. Wirth als Ausbildungssprache geschaffen. Das Ziel war eine Sprache zu schaffen, mit der fundamentale Konzepte und Strukturen systematisch, präzise und adäquat ausgedrückt werden können und neue Ergebnisse der systematischen Programmentwicklung berücksichtigt werden. PASCAL ist eine universelle Programmiersprache für mathematische und kommerzielle Aufgaben sowie für die Systemprogrammierung.
PASCAL - Compiler stehen für alle gebräuchlichen Computer zur Verfügung.
Die Weiterentwicklung hat zu maschinenspezifischen Dialekten geführt, wodurch die Portabilität von Programmen erschwert wird.
Drei Gruppen von Datentypen sind zugelassen. Dies sind einfache Typen, Zeiger (Pointer) und strukturierte Typen. Zur Programmierung stehen umfangreiche Standardprozeduren und -funktionen zur Verfügung. Zur Datenein- und Datenausgabe sind leistungsfähige Standardprozeduren vorhanden. Die Möglichkeiten der strukturierten Programmierung werden durch eine leistungsfähige Unterprogrammtechnik und die Möglichkeit des rekursiven Funktionsaufrufes unterstützt.
PASCAL ist eine weit verbreitete Programmiersprache. Die Nachteile bestehen vor allem darin, dass keine dynamischen Feldvereinbarungen und keine Echtzeitbefehle möglich sind. Daraus resultierte die Weiterentwicklung zu TURBO-PASCAL. TURBO-PASCAL gilt derzeit als effektivste PASCAL-Implementierung. Borland-TURBO-PASCAL hat sich zum Marktführer entwickelt.
Die letzte entwickelte Borland-Pascal-Version 7 wurde zusätzlich mit dem Modul TURBOVision ausgestattet, der eine anwenderfreundliche Anwendung der objektorientierten Programmierung ermöglicht. Es wurden drei Komponenten eingeführt, Turbo-Pascal, Borland-Pascal und Borland-Pascal für Windows. Damit wurde die Entwicklung von WINDOWS-Programmen möglich.
Eine Weiterentwicklung von Borland-Pascal ist die Programmiersprache Delphi, in der aktuellen Version 2005 (Delphi2006 soll demnächst erscheinen). Delphi gehört, wie auch die von Microsoft entwickelte Programmiersprache Visual Basic, zu einer neuen Generation von Programmiersprachen. Kennzeichnend für diese Programmiersprachen sind die Eigenschaften:

- objektorientiert,
- ereignisgesteuert und
- visuell.

Sie wurden mit dem Ziel entwickelt, die Erstellung von WINDOWS-Anwendungen zu beschleunigen und zu vereinfachen. Die enthaltene Pascal-Version wird als Object-Pascal bezeichnet.

C

Die Programmiersprache C wurde Mitte der 70er Jahre von Ritchie (Bell Laboratories) entwickelt. C ist gegenwärtig die populärste Systemprogrammiersprache (z.B. ist das weit verbreitete Betriebssystem UNIX in C geschrieben). Die Sprache ist wie PASCAL blockorientiert und erlaubt eine sehr kompakte Schreibweise. Ein C-Programm ist eine Menge von Funktionen, die einander aufrufen können. C eignet sich besonders für die Echtzeitprogrammierung (Steuerungen) und die Systemprogrammierung. Diese Sprache wird häufig eingesetzt, wenn systemnahe Aufgaben effektiv ausgeführt werden müssen.
Es wurde zur Version C++ weiterentwickelt. Damit steht ein Werkzeug für die Entwicklung großer Softwaresysteme zur Verfügung, das leistungsfähige Komponenten zur objektorientierten Programmierung enthält.

FORTH

Die Programmiersprache FORTH wurde von Moore Ende der 60er Jahre entwickelt und 1968 auf einem IBM-Computer realisiert. Der ursprüngliche Verwendungszweck war die Programmierung von Industrierobotern. Daraus ergibt sich auch ein wesentlicher Nachteil, denn es können nur ganze Zahlen verarbeitet werden.
FORTH zählt zu den dialogorientierten Programmiersprachen. Sie ist sowohl als Interpretersprache, als auch als Compilersprache nutzbar.
Die Attraktivität von FORTH ist bis heute erhalten geblieben, da eine strukturierte Programmierung unterstützt wird und eine Übertragung in sehr effizienten Maschinencode möglich ist. FORTH benötigt wenig Grundspeicherplatz und erlaubt eine maschinennahe Programmierung.
FORTH unterscheidet sich grundlegend von anderen höheren Programmiersprachen. Die Befehle werden in UPN (umgekehrt polnischer Notation) eingegeben.
Beispiel: zur Berechnung von 12 + 15 wird eingegeben: 12 15 + .

LOGO (vom griechischen logos - Vernunft)

Die Programmiersprache LOGO wurde am MIT-Institut in Boston von Papert entwickelt, damit schon Kinder im Vorschulalter spielend geometrische Zusammenhänge lernen, wobei das Programmieren von Figuren der "Turtlegeometrie" (Schildkrötengrafik) eine wesentliche Rolle spielt.
LOGO ist im Gegensatz zu den gebräuchlichen Programmiersprachen funktionsorientierter. Das bedeutet, dass auch komplizierte zusammengesetzte Datenstrukturen als Funktionswerte von Funktionen dargestellt werden können. Grundlegende Unterschiede zu anderen Programmiersprachen ergeben sich aus dem Speicherkonzept. Es werden Behälter statt Variablen verwendet und es gibt keine Typen. Das LOGO-System nutzt neben Text immer eine hochauflösende Grafik. Daraus ergibt sich in der Regel eine Speicherplatzforderung von mindestens 1 MByte RAM. Logo wird interpretativ abgearbeitet.

PROLOG (PROgramming in LOGic)

Erste Versionen dieser Sprache wurden bereits 1972 von Colmerauer entwickelt. Die heutigen Versionen entstanden etwa 1981. PROLOG ist zu einer der Hauptsprachen der künstlichen Intelligenz und der 5. Computergeneration geworden. Sie zählt zu den deklarativen Programmiersprachen.
Der Programmierer beschreibt sein Problem durch Fakten und Regeln (Wissensbasis). Das Programm hat die Aufgabe zu prüfen, ob eine gestellte Frage als wahr oder falsch im Rahmen der Wissensbasis zu beantworten ist. PROLOG gewinnt im Zusammenhang mit der künstlichen Intelligenz immer mehr an Bedeutung und wird heute beispielsweise schon in komplexen Steuerungssystemen eingesetzt. PROLOG ist das wichtigste Beispiel für die logische Programmierung.

Smalltalk

Die Sprache Smalltalk wurde im Zeitraum von 1972 bis 1980 von Kay, Ingalls u. a. am Xerox Palo Alto Forschungszentrum entwickelt. Smalltalk wurde so entworfen, dass es den objektorientierten Ansatz auf eine vollständig konsistente Weise anwendet. Besonders für Mainframe-Applikationen nimmt die Bedeutung von Smalltalk immer mehr zu, da vielfach veraltete COBOL-Anwendungen durch Smalltalk-Applikationen ersetzt werden.

Java

Die Programmiersprache Java wurde im Frühjahr 1995 von Sun Microsystems vorgestellt. Ursprünglich wurde sie für das interaktive Fernsehen sowie die Steuerung von Geräten im Haushalt entwickelt. Java ist eine einfache, objektorientierte, multithreaded, robuste, architekturneutrale, übertragbare und dynamische Programmiersprache. Sie basiert auf der Programmiersprache C++, wurde aber wesentlich vereinfacht.
Java-Programme werden in Bytecode compiliert und können damit ohne weitere Compilierung auf verschiedenen Computern ausgeführt werden, sofern es eine Java Virtual Machine dafür gibt. Der maschinenunabhängige Bytecode wird bei der Ausführung von einem Interpreter, der Java Virtual Machine, abgearbeitet.
Mit der Verbreitung des Internet nahm auch die Bedeutung dieser Programmiersprache zu. Java Programme, die auch als Java-Applets bezeichnet werden, lassen sich in HTML-Seiten einbinden.

Wird auf eine solche Seite von einem Benutzer zugegriffen, so werden die darauf enthaltenen Applets auf den eigenen Computer heruntergeladen und ausgeführt.
Aus dem Java wurde Java-Skript abgeleitet. Java-Skript ist eine in die HTML-Syntax integrierte Programmiersprache, deren Befehle direkt in die HTML-Syntax von WWW-Seiten eingefügt werden.

Delphi

Delphi ist eine Programmiersprache, die zur Erstellung von Windows-Applikationen entwickelt wurde. Kennzeichnend für diese Programmiersprache sind die Eigenschaften

- objektorientiert,
- ereignisgesteuert und
- visuell.

Windows-Anwendungen sind, im Gegensatz zu DOS-Anwendungen, in der Regel graphikorientiert. Da Delphi für Windows-Anwendungen entwickelt wurde, unterstützt es besonders den Entwurf von Nutzeroberflächen. Der Zugriff zu Windows erfolgt über die Windows-API. Delphi verfügt über einen echten Compiler, der den in Pascal geschriebenen Quelltext in C transferiert und mit einem Backend-Compiler übersetzt. Damit sind kurze Abarbeitungszeiten für die Programme erreichbar. In Delphi wurden Datenbankschnittstellen integriert. Die derzeit aktuelle Version Delphi2005 erzeugt einen effizienteren Laufzeitcode durch die Einführung von Packages. Standard-Code wird nicht mit in die .exe-Datei übernommen, sondern in einer oder mehrerer DLL's abgelegt. Delphi wurde auch um Werkzeuge erweitert, um Programme für die wichtigsten Internet-Dienste zu erstellen. Es besteht auch die Möglichkeit, Active-X-Controls zu erzeugen.

Visual Basic/ Visual C++/ Visual J++

Diese Sprachen sind verbreitete Sprache aus der Klasse der Microsoft Visual-Sprachen. Ihr Hauptanwendungsgebiet ist die Programmierung von Windows- und Internetanwendungen. Sie verfügen über eine graphische Entwicklungsoberfläche. Die Programmentwicklung erfolgt objektorientiert.

2.5. Registermaschinen

2.5.1. Begriff

Der Prozessor eines Computers hat nur ein eingeschränktes Befehlsverzeichnis. Die im Kapitel 1 genannten Kontrollstrukturen können in der Art von ihm nicht ausgeführt werden. Es ist also notwendig alle Befehle einer Programmiersprache so umzusetzen, dass diese vom Prozessor verstanden und ausgeführt werden können. Dazu dienen die Interpreter und Compiler.

Eine Registermaschine ist keine Maschine im herkömmlichen Sinne. Sie ist vielmehr ein mathematisches Modell für die Abläufe im Prozessor eines Computers.

Eine Registermaschine besteht aus drei einzelnen Registern:

- dem Befehlsregister,
- dem Arbeitsregister und
- dem Speicherregister.

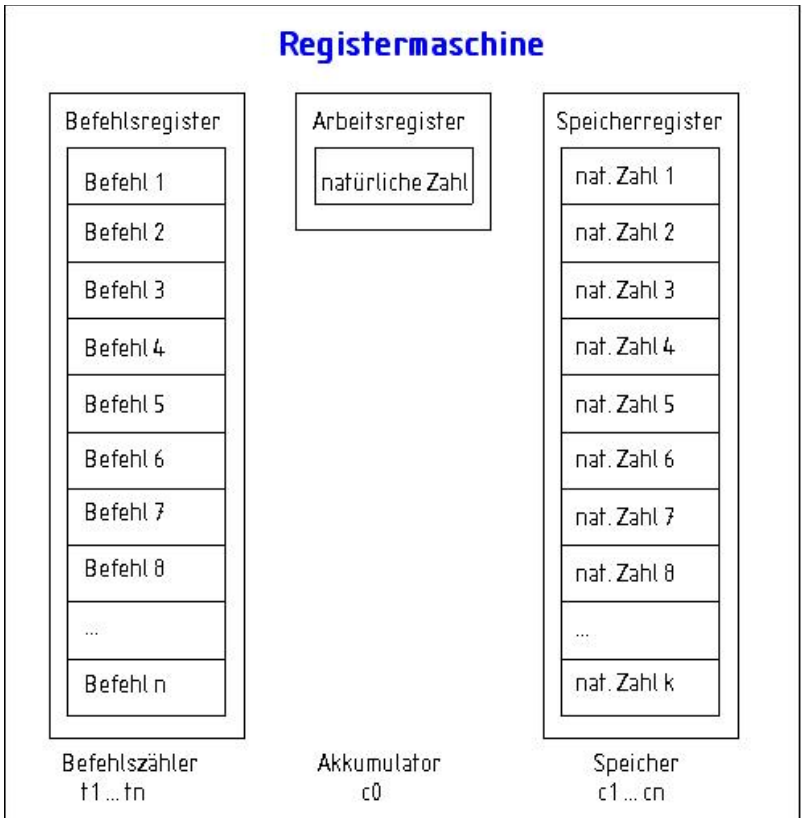


Abb. 26: Schematische Darstellung einer Registermaschine

Befehlsregister

Das Befehlsregister enthält eine endliche Menge von Befehlen. Es stehen die folgenden Befehle zur Verfügung:

Ein- und Ausgabebefehle		
LOAD i	Lädt den Wert aus dem i-ten Speicher in den Akkumulator.	$c_0 = c_i$
CLOAD i	Lädt den Wert i in den Akkumulator.	$c_0 = i$
ILOAD i	Liest den Wert k aus dem i-ten Speicher und lädt den Wert aus dem k-ten Speicher in den Akkumulator.	$c_0 = c_k$, $k = c_i$
STORE i	Speichert den aktuellen Wert des Akkumulators in den i-ten Speicher.	$c_i = c_0$
ISTORE i	Liest den Wert k aus dem i-ten Speicher und speichert dann den aktuellen Wert des Akkumulators in den k-ten Speicher.	$c_k = c_0$, $k = c_i$
Sprungbefehle		
GOTO i	Springe zum i-ten Befehl des Befehlsregisters.	
IF c0 = 0 GOTO i	Falls der Inhalt des Akkumulators 0 ist, springe zum i-ten Befehl des Befehlsregisters, ansonsten mache mit dem nächsten Befehl weiter.	
Arithmetische Befehle		
ADD i	Addiert den Wert des i-ten Speichers mit dem Akkumulator. Ergebnis wird im Akkumulator gespeichert.	$c_0 = c_0 + c_i$
CADD i	Addiert i mit dem Akkumulator. Ergebnis wird im Akkumulator gespeichert.	$c_0 = c_0 + i$

SUB i	Subtrahiert den Wert des i-ten Speichers vom Akkumulatorwert. Ergebnis wird im Akkumulator gespeichert. Ist das Ergebnis kleiner als 0, dann wird 0 gespeichert.	$c_0 = c_0 - c_i$
CSUB i	Subtrahiert i vom Akkumulatorwert. Ergebnis wird im Akkumulator gespeichert. Ist das Ergebnis kleiner als 0, dann wird 0 gespeichert.	$c_0 = c_0 - i$
MULT i	Multipliziert den Wert des i-ten Speichers mit dem Akkumulator. Ergebnis wird im Akkumulator gespeichert.	$c_0 = c_0 * c_i$
CMULT i	Multipliziert i mit dem Akkumulator. Ergebnis wird im Akkumulator gespeichert.	$c_0 = c_0 * i$
DIV i	Dividiert den Akkumulatorwert durch den Wert des i-ten Speichers. Ergebnis wird im Akkumulator gespeichert. Es wird nur der ganzzahlige Anteil gespeichert.	$c_0 = c_0 / c_i$
CDIV i	Dividiert den Akkumulatorwert durch i. Ergebnis wird im Akkumulator gespeichert. Es wird nur der ganzzahlige Anteil gespeichert.	$c_0 = c_0 / i$
Programmende		
END	Anzeige des Programmendes	

Arbeitsregister (Akkumulator)

Der Akkumulator ist der eigentliche Arbeitsspeicher. Hier werden sämtliche arithmetischen Operationen ausgeführt und zunächst abgespeichert. Das Arbeitsregister kann immer nur eine einzelne natürliche Zahl (einschließlich Null) aufnehmen.
Die Arbeitsweise mit natürlichen Zahlen ergibt sich aus der allgemeinen Arbeitsweise des Computers, der nur mit den Dualzahlen arbeitet. Somit können auch nur entsprechende Dualzahlen (ohne Vorzeichen, ohne Komma) abgespeichert werden.

Speicherregister

Das Speicherregister dient der Zwischenspeicherung einzelner Werte für die nachfolgenden Rechenoperationen. Es hat eine endliche Länge. Innerhalb des Speicherregisters selbst kann nicht gerechnet werden. Es sind nur Lese- und Schreibzugriffe gestattet.

2.5.2. Beispiel

Das nachfolgende Beispiel stellt eine Registermaschine zur Berechnung der Potenz 2^3 dar.

1. CLOAD 2
2. STORE 1
3. CLOAD 3
4. STORE 2
5. CLOAD 1
6. STORE 3
7. LOAD 2
8. IF c0 = 0 GOTO 16
9. LOAD 3
10. MULT 1
11. STORE 3
12. LOAD 2
13. CSUB 1
14. STORE 2
15. GOTO 8
16. LOAD 3
17. STORE 1
18. END

Im Folgenden sollen die einzelnen Schritte näher untersucht werden. Öffnen Sie dazu das folgende Delphi-Programm.

PROGRAMME / 001 / REGISTERMASCHINE.DPR

Fragen und Aufgaben
2. Programmiersprachen

Theoretische Grundlagen

1. Erläutern Sie die Begriffe Sprache, Syntax und Semantik aus der Sicht einer Programmiersprache.
2. Erläutern Sie die Einteilung der Programmiersprachen.
3. Unterscheiden Sie die Arbeitsweise von Compiler und Interpreter. Nennen Sie jeweils typische Vertreter.
4. Erläutern Sie die Arbeitsweise einer Registermaschine. Gehen Sie auf die einzelnen Befehle ein.
5. Welche Bedeutung haben Registermaschinen aus der Sicht der Informatik?

Praktische Aufgaben

6. Entwickeln Sie eine Registermaschine, die die erste Kommastelle bei der Division durch 3 (bzw. 5; 6) berechnet.
7. Entwickeln Sie eine Registermaschine, die die ersten 5 (10; n) natürlichen Zahlen summiert.
8. Entwickeln Sie eine Registermaschine, die die Relation (<, >, =) zweier natürlicher Zahlen überprüft. Als Ausgabe soll 1 für a < b, 2 für a > b bzw. 3 für a = b angegeben werden.

3. Einführung in die Programmierung mit Delphi 7

3.1. Grundsätzliches zum Aufbau eines Delphi-Programms

Umgangssprachlich verstehen wir unter dem Begriff Programm z.B. den Ablauf einer Veranstaltung (Theaterprogramm). Ein Programm in der Informatik gibt an, in welcher Art bestimmte Eingabegrößen in die von uns erwarteten Ausgabegrößen umgewandelt werden (EVA-Prinzip). Der Programmtext wird dabei auch als Quelltext bezeichnet. Damit ein Programmtext vom Computer verstanden werden kann benötigt man einen Compiler oder Interpreter.

Arbeitsweise Compiler

Ein Compiler "übersetzt" zunächst den gesamten geschriebenen Quelltext in die Computersprache (Maschinensprache) und beginnt dann erst die Programmausführung. Alle Syntaxfehler im Quelltext werden im Vorfeld erkannt.

Arbeitsweise Interpreter

Ein Interpreter "übersetzt" immer nur eine Anweisung des Quelltextes und führt diese Anweisung sofort aus. Danach wird die darauf folgende Anweisung übersetzt und abgearbeitet, Fehler im Quelltext werden erst bei Erreichen der entsprechenden Position im Programmtext erkannt. Delphi arbeitet mit einem Compiler.

Die Delphi-Entwicklungsumgebung besteht im Wesentlichen aus drei Teilen:

- Der Benutzeroberfläche (Formular - TForm)
- Den einzelnen Komponenten auf der Benutzeroberfläche
- Den Quelltexten zu diesen Komponenten, die dafür sorgen, dass bestimmte Ereignisse ausgelöst werden können.

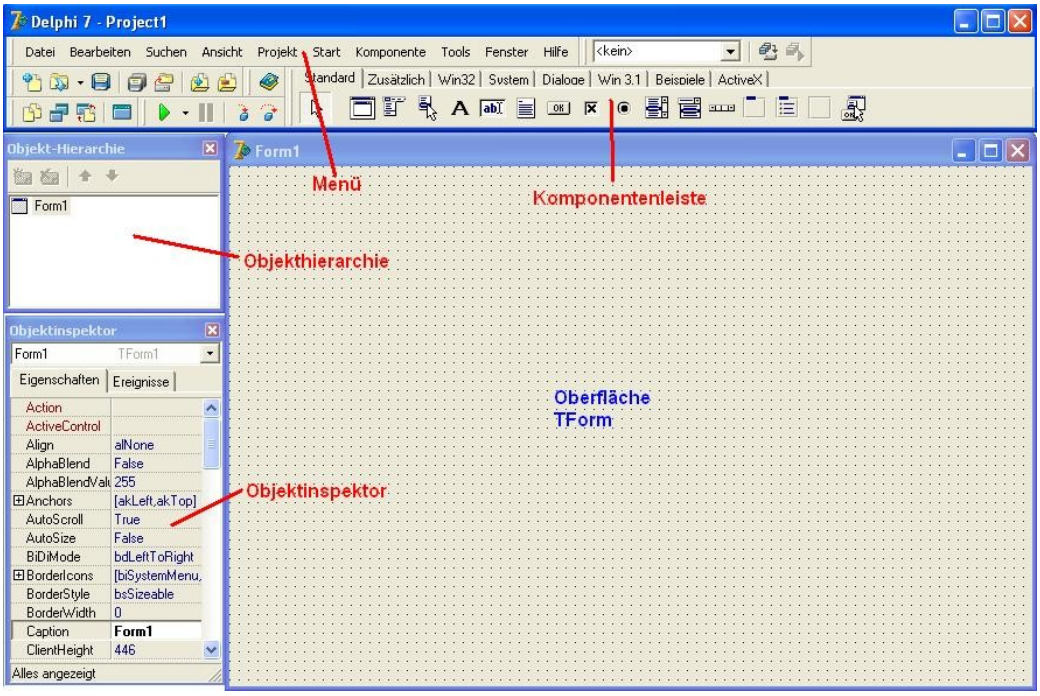


Abb. 27: Ansicht der Delphi-Entwicklungsumgebung

⚠	Es ist darauf zu achten, dass Delphi-Programme nicht nur aus einer einzelnen Datei bestehen. Vielmehr werden beim Anlegen eines neuen Programms die Daten in einem gesamten Projekt gespeichert.

Zu diesem Projekt gehören im Wesentlichen die folgenden sechs Dateien:

Datei	Inhalt
Projekt1.dpr	Diese Datei enthält alle für die Ausführung des Programms wichtigen Daten.
Projekt1.exe	Diese Datei kann als eigenständige Datei ausgeführt werden, sie ist die dem späteren Nutzer zugängliche Datei.
Projekt1.res	ermittelt die, für die Erstellung notwendigen, Speicherbelegungen
Projekt1.dof	enthält die Optionen des Delphi-Projekts
Unit1.dcu	enthält die für den Rechner übersetzten (compilierten) Units
Unit1.pas	enthält den Quelltext der Units

Daher ist es besonders wichtig jedes Projekt (also die zu diesem Projekt gehörenden Dateien) in einem gesonderten Verzeichnis abzulegen.

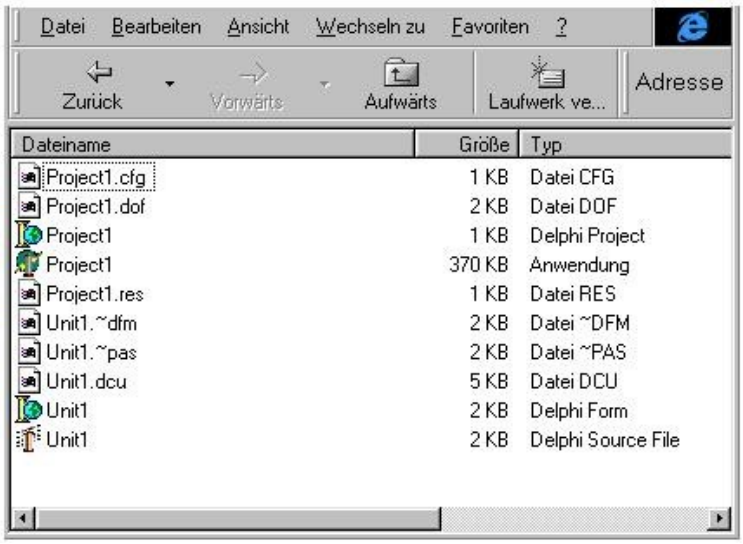


Abb. 28: Ansicht eines Ordners mit den Dateien eines Projekts

In der Darstellung sind noch weitere Dateien erkennbar, auf die wir hier aber vorerst nicht näher eingehen wollen.

⚠	**Löschen Sie grundsätzlich keine Dateien aus diesem Ordner (eine Ausnahme bildet dabei die Datei *.exe). Auch darf der Name einer solchen Datei nie über den Dateimanager geändert werden. Beides kann zu unerwarteten Problemen führen.**

Will man den Dateien einen anderen Namen geben (z.B. statt Project1 - Beispiel1) dann sollte das nur beim ersten Speichern der Daten in dem Ordner erfolgen.

Programmieraufgabe 1

An dieser Stelle wird es Zeit zum ersten Mal ein Delphi-Projekt zu erstellen. Gehen Sie dazu wie folgt vor:

1. Erstellen Sie zunächst den Ordner in dem Sie das Beispiel 1 speichern wollen.
2. Starten Sie dann die Programmierung mit Delphi (entweder mit dem Icon auf dem Desktop oder über Start - Programme - Borland Delphi7 - Delphi7)
3. Speichern Sie Ihr Projekt mit Datei - Alles Speichern wie in den nächsten Punkten beschrieben.
    a. Speichern Sie die UNIT1 unter dem Namen: Beispiel.pas
    b. Speichern Sie das PROJECT1 unter dem Namen Beispiel1.dpr
4. Achten Sie beim Speichern darauf, dass Umlaute und Sonderzeichen nicht erlaubt sind.

Wenn Sie dies alles erledigt haben, wechseln Sie noch einmal in den entsprechenden Ordner (ohne Delphi zu schließen) und prüfen Sie, welche Dateien hier erstellt wurden. Sie werden feststellen, dass hier wesentlich mehr Dateien vorhanden sind als nur die beiden Beispiel.pas und Beispiel1.dpr. Dagegen wurde aber eine ausführbare *.exe Datei noch nicht erstellt.
Wechseln Sie noch einmal zu Delphi und starten Sie Ihr "Programm" mit dem grünen Pfeil.

Sie haben hier zwar noch keine einzige Zeile Programm geschrieben, es steht Ihnen aber trotzdem ein vollständiges Windows-Formular zur Verfügung.
Schließen Sie dieses Programm wie in Windows üblich und wechseln Sie anschließend noch einmal in Ihren Ordner. Sie werden feststellen, dass jetzt eine ausführbare Datei Beispiel1.exe vorhanden ist. Diese Datei kann unabhängig vom Delphi-Programm gestartet werden. Probieren Sie es aus. Unser erstes Programm hat damit seinen Zweck erfüllt.

3.2. Wie entsteht ein Programm?

Bevor man mit dem eigentlichen Programmieren beginnt, ist es immer günstig, sich die die folgenden Fragen zu durchdenken und ggf. die Vorüberlegungen schriftlich zu formulieren. Dies vermeidet viele Fehler während der Programmierung und erspart überflüssiges Ausprobieren und Fehlersuchen:

1. Analyse der Aufgabenstellung (was soll eigentlich mit dem Programm erreicht werden) bzw. was will ich selbst mit diesem Programm erreichen?
2. Welche Komponenten benötige ich zur Ausführung der Aufgabe?
    a. Ein- und Ausgabekomponenten
    b. Komponenten die bestimmte Aktionen auslösen
    c. Welche Namen und Beschriftungen sollen die Komponenten erhalten
3. Welche Variablen benötige ich für die Programmierung, welche sinnvollen Namen (Bezeichner) gebe ich diesen? (z.B. statt x den Namen Eingabe1 verwenden)
4. Wie sehen die Algorithmen zur Implementierung der Aufgaben aus?

Wenn diese vier Fragen beantwortet sind, kann man zum eigentlichen Teil der Programmierung übergeben:

5. Erstellen eines neuen Projekts
6. Bereitstellen der notwendigen Komponenten und ggf. Vergeben aussagekräftiger Namen für diese Komponenten
7. Gestalten der Oberfläche (Farben, Beschriftungen, Grafiken)
8. Implementierung der Quelltexte
9. Erstellen der Dokumentation

Damit ist die Programmerstellung aber noch nicht abgeschlossen. Jetzt kommt der häufig schwierigste Teil der Programmerstellung: Testen und Fehlerbehebung. Dabei sind die folgenden Fragestellungen zu untersuchen:

10. Sind noch Syntaxfehler vorhanden?
    (Diese lassen sich oft leicht beheben, wenn man auf die entsprechenden Compilermeldungen achtet).
11. Reagiert das Programm nach meinen Vorstellungen?
    (werden also die richtigen Werte ausgegeben) Diese Fehler werden vom Compiler meist nicht angezeigt, müssen also im wahrsten Sinne des Wortes im Quelltext gesucht werden.)
12. Reagiert das Programm auf alle zugelassenen Eingabewerte richtig?
    Es ist natürlich schwierig alle zugelassenen Daten einzugeben, aber einige "extreme" Werte lassen sich doch ausprobieren.

    BEISPIEL:

    Es wurde die Division zweier Zahlen programmiert. Auf Aufgaben der Form 4 : 2 oder 16,4 : 4 reagiert das Programm richtig. Was aber passiert, wenn die Aufgabe 4 : 0 eingegeben wird?

3.3. Entwurfs- und Laufzeitmodus

Bei der Programmierung muss weiterhin darauf geachtet werden, dass Änderungen am Quelltext nur vorgenommen werden sollten, wenn sich das Programm im Entwurfsmodus befindet.

Kennzeichen Entwurfsmodus:

- Objektinspektor ist erkennbar
- Rasterung der Oberfläche vorhanden

Kennzeichen Laufzeitmodus:

- Keine Rasterung der Oberfläche
- Programm reagiert ggf. auf Eingaben
- Objektinspektor nicht vorhanden

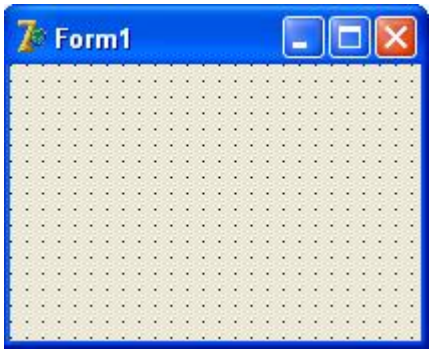


Abb. 30: Oberfläche im Entwurfsmodus

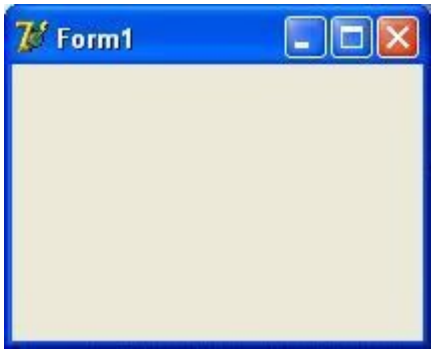


Abb. 31: Oberfläche im Laufzeitmodus

Programmieraufgabe 2

Öffnen Sie noch einmal die Programmieraufgabe 1 aus Kapitel 3.1. Starten Sie das Programm und vergleichen Sie die Unterschiede zwischen dem Entwurfs- und Laufzeitmodus.
Zum Öffnen des Programms gehen Sie wie folgt vor:

1. Starten Sie Delphi.
2. Wählen Sie im Menü Datei den Punkt "Projekt öffnen" aus.
3. Wechseln Sie in das Verzeichnis und öffnen Sie Beispiel1.dpr. Die Unit Beispiel.pas öffnet sich dabei automatisch und muss nicht gesondert ausgewählt werden.

3.4. Aufbau der Entwicklungsumgebung

3.4.1. Die Menüleiste

Abb. 32: Ansicht des Delphi-Menüs

Ähnlich wie in anderen Programmen (z.B. Excel, Word...) existiert auch in der Delphi-Entwicklungsumgebung eine Menüleiste in der wichtige Funktionen der Delphi-Programmierung eingestellt bzw. Bearbeitungsschritte vorgenommen werden können. Einige der wichtigsten Funktionen seien hier kurz beschrieben:

Menü Datei

Funktion	Beschreibung
a) Neu	hiermit können neue Anwendungen erstellt, neue Formulare in ein Projekt aufgenommen bzw. neue Units deklariert werden
b) Projekt öffnen	öffnet ein Projekt mit allen dazugehörigen Dateien (es werden aber nicht immer alle diese Dateien angezeigt)
c) Neu öffnen	zeigt eine Liste der zuletzt bearbeiteten Projekte an
d) Projekt speichern unter	mit dieser Funktion kann ein gesamtes Projekt gespeichert werden (neuer Name)
e) Alles speichern	speichert alle zum Projekt gehörenden Dateien
f) Alles schließen	schließt alle zum Projekt gehörenden Dateien
g) Drucken	druckt entweder das Formblatt oder den Quelltext der Unit (je nachdem welcher Teil des Projekts gerade aktiv ist)

Menü suchen

sucht nach vom Nutzer festgelegtem Text innerhalb des Quelltextes der aktuellen UNIT oder im gesamten Projekt.

Menü Ansicht

Hier können unter anderem Units bzw. Formblätter, die zwar in dem Projekt eingebunden wurden, aber nicht aktiv sind, aufgerufen werden.

Menü Start

Mit dem Menü Start kann ein Delphi-Projekt insgesamt oder im Einzelschritt abgearbeitet werden. Es stehen hier auch Optionen für die Suche nach logischen Fehlern, die vom Compiler nicht erkannt werden, zur Verfügung.

3.4.2. Die Komponentenleiste

Abb. 33: Ansicht der Komponentenleiste

Rechts unterhalb der Menüleiste befindet sich die Komponentenleiste. Hier sind alle standardmäßig vorgegebenen Komponenten enthalten. Sie wird in mehrere verschiedene Komponentenlisten unterteilt (z.B. Standard, Zusätzlich, System, Dialoge...).
Elemente der Komponentenlisten können auf die Formblätter gelegt und mit Quelltexten versehen werden. Die einzelnen Komponenten werden einfach durch Anklicken ausgewählt. Will man die gleiche Komponente mehrmals auf dem Formblatt platzieren, so drückt man zusätzlich auf die Umschalttaste auf der Tastatur und positioniert die Komponente entsprechend mehrmals. Dieser Vorgang wird erst durch Klicken auf eine andere Komponente oder auf den Pfeil links in der Komponentenliste beendet.

3.4.3. Die Objekt-Hierarchie

Abb. 34: Ansicht einer Objekt-Hierarchie

Links unterhalb der Menüleiste befindet sich ein Fenster für die Objekt-Hierarchie. Hier werden alle auf dem Formblatt vorhandenen Komponenten mit ihrem Namen angezeigt. Weiterhin kann man erkennen, welche Unterkomponenten ggf. einer anderen Komponente zugeordnet sind. Im Beispiel liegen auf dem Formblatt eine Edit-Komponente, eine Label-Komponente und eine Komponente TabControl. Auf dieser Komponente ist noch eine weitere Komponente Memo enthalten.

3.4.4. Der Objektinspektor

Unterhalb der Objekt-Hierarchie ist der Objekt-Inspektor zu finden. Klickt man zunächst auf eine Komponente, dann werden im O-I alle änderbaren Eigenschaften dieser Komponente angezeigt. Weiterhin kann man hier auch auswählen auf welche Ereignisse eine Komponente reagieren soll (Ereignisseite).

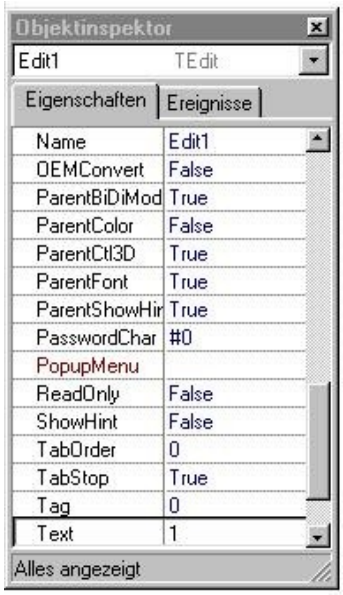


Abb. 35: Ansicht des Objektinspektors für eine Komponenten TEdit

PROGRAMME / 002 / KOMPONENTEN.DPR

3.5. Eigenschaften und Ereignisse von TForm

Wie bereits im Thema 3.4. erwähnt können mit Hilfe des Objekt-Inspektors (O-I) Eigenschaften einer Komponente verändert werden. Im Folgenden werden die für uns zunächst wichtigsten Eigenschaften dieser Komponenten beschrieben. Weiterhin werden einige Ereignisse die mit diesen Komponenten erzeugt werden können dargelegt. Weitere Eigenschaften und Ereignisse werden in den folgenden Kapiteln behandelt bzw. finden Sie in der Delphi-Hilfe.

3.5.1. Eigenschaften von TForm

Die Klasse TForm ist die für uns wichtigste Klasse in Delphi. Mit Hilfe von TForm wird das Formblatt erzeugt. Dieses Formblatt dient der Aufnahme der Komponenten und ist daher für die Funktionalität von Delphi unerlässlich. Einige der wichtigsten Eigenschaften der Komponente werden im Folgenden näher erläutert. Weitere Eigenschaften finden Sie in der Delphi-Hilfe.

Eigenschaft	Erläuterung
Caption:	gibt die Beschriftung im obersten Teil des Formulars an (z.B. von Form1 in Beispiel 1)
Color:	ändert die Farbe des Formulars (z.B. von clBtnFace in clAqua)
Cursor:	ändert die Form des Cursors (diese Änderung wird erst zur Laufzeit aktiviert, probieren Sie es aus)
Font:	ändert die Schriftarten, Farben und Schriftstile der auf dem Formblatt vorhandenen Komponenten (legen Sie dazu ein Label auf die Oberfläche und beobachten Sie die Auswirkungen)
Hint:	zeigt Kurzhinweise zu einer Komponente an, schreiben Sie hier also irgendeinen Text (diese Änderung wird erst zur Laufzeit aktiviert). Dazu muss aber noch zusätzlich die Eigenschaft ShowHint (weiter unten im O-I) auf true gesetzt werden.

Name:	legt den Namen des Formulars fest. Standardmäßig lautet dieser Name Form1. Der Name dient zur internen Verwaltung bzw. während der Implementierungsphase (Programmierung) zum Aufrufen des Formulars. Der Name darf nur einmal vergeben werden. Wird der Name geändert sollte der Begriff Form_ ... verwendet werden. Hiermit kann man spätere Verwechslungen ausschließen.
ShowHint:	vgl. Bemerkungen zu Hint
Visible:	legt die Sichtbarkeit des Formulars zur Laufzeit fest. Diese Eigenschaft sollte nur dann geändert werden, wenn während der Programmausführung anderweitig auf diese Eigenschaft zugegriffen werden kann. Lassen Sie diese Eigenschaft zunächst in jedem Fall auf den standardmäßig voreingestellten Wert true.
WindowState:	legt fest in welcher Größe das Formular nach Programmstart dargestellt wird. Probieren Sie es aus.

Die folgende Darstellung zeigt ein Formblatt zur Entwurfs- und zur Laufzeit.

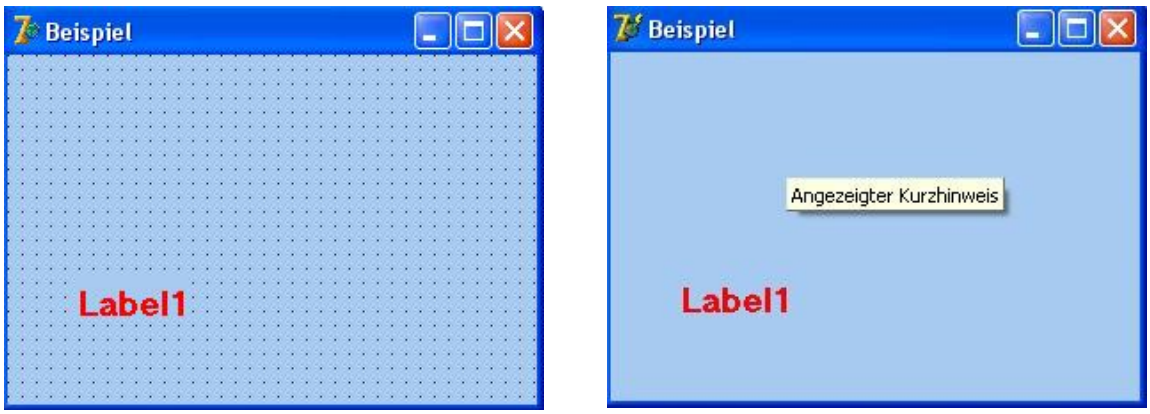


Abb. 36 und 37: Formblatt zur Entwurfs- und Laufzeit

Weitere Eigenschaften des Formblattes finden Sie in der Delphi-Hilfe.
Viele dieser Eigenschaften treten auch bei anderen Komponenten wieder auf. Es wird dann nicht noch einmal im Einzelnen darauf eingegangen.

Programmieraufgabe 3

Erzeugen Sie zunächst eine neue Anwendung. Nehmen Sie anschließend die nachfolgenden Änderungen an Form1 vor. Dazu klicken Sie im Objektinspektor auf der Eigenschaftsseite (diese sollte standardmäßig vorgegeben sein) und ändern die Eigenschaft entsprechend ab:

Caption	von "Form1" in "Eigenschaften des Formblattes"
Color	von "clBtnFace" in "clyellow"
Cursor	von "crDefault" in "crHandPoint"
Font	auf die drei Pünktchen klicken und Eigenschaften nach Belieben einstellen.
Hint	Schreiben Sie hier einen beliebigen Text
ShowHint	von "false" auf "true"
WindowState	von "wsNormal" auf "wsMaximized"

Starten Sie Ihre Anwendung und beobachten Sie die Auswirkung der vorgenommenen Änderungen.

PROGRAMME / 003 / FORMBLATT.DPR

3.5.2. Ereignisse von TForm

Bisher haben Sie nur Eigenschaften der Komponenten geändert. Sie haben dabei noch keine einzige Zeile Programmtext geschrieben und damit aber schon eine Menge erreicht. Hier werden Sie das erste Mal einige kurze Programmtexte schreiben, mit denen Sie einige Eigenschaften der Oberfläche zur Laufzeit verändern werden. Wechseln Sie zunächst auf die Ereignisseite des Objektinspektors. Hier finden Sie eine Vielzahl von Möglichkeiten Ihrem Formblatt Ereignisse zuzuordnen und damit bestimmte Aktionen auszulösen. Die wichtigsten Ereignisse dabei sind:

Ereignis	Löst folgende Aktivität aus
OnActivate	Dieses Ereignis wird bei der Aktivierung des Formulars ausgelöst.
OnClick	Ereignis wird durch Klicken auf die Komponente ausgelöst.
OnDblClick	Ereignis wird durch einen Doppelklick auf die Komponente ausgelöst.
OnKeyDown	Ereignis wird ausgelöst, wenn eine Taste gedrückt wird.
OnMouseMove	Ereignis wird ausgelöst, wenn sich die Maus über die Komponente bewegt

Weitere Ereignisse des Formblattes finden Sie in der Delphi-Hilfe.

Mit Hilfe dieser fünf Ereignisse soll nun ein erstes kleines Programm geschrieben werden. In diesem Programm werden einige Farbänderungen ausgeführt. Führen Sie die nachfolgenden Schritte genau aus. Beim Schreiben des Programmtextes achten Sie auf jedes einzelne Zeichen.

Hiermit haben wir bereits einige wichtige Eigenschaften eines Programms kennen gelernt.

- Werden Komponenten und Variablen Werte oder Eigenschaften zugeordnet, erfolgt dieses mit der so genannten Ergibt-Anweisung (**:=**). Wie wir später noch kennen lernen werden, unterscheidet sich diese Anweisung deutlich vom mathematischen Gleichheitszeichen.
- Einzelne Anweisungen werden in Delphi (und auch in vielen anderen Programmiersprachen) durch ein Semikolon getrennt.
- Bei der Programmierung kommt es oft auf einzelne Zeichen an (ein Deutschaufsatz kann auch gelesen und verstanden werden, wenn einige kleine Rechtschreibfehler auftreten. Bei auftretenden "Rechtschreibfehlern" in der Programmierung kann das Programm nicht mehr übersetzt werden.)

	Erzeugen Sie die Ereignisse immer mit der Ereignisseite des OI bzw. durch Doppelklick auf die Komponente. Andernfalls kann der Programmtext der Komponente nicht zugeordnet werden.

Programmieraufgabe 4

Mit Hilfe dieser fünf Ereignisse soll nun ein erstes kleines Programm geschrieben werden. In diesem Programm werden einige Farbänderungen ausgeführt. Führen Sie die nachfolgenden Schritte genau aus. Beim Schreiben des Programmtextes achten Sie auf jedes einzelne Zeichen.

1. Klicken Sie doppelt in die Spalte neben OnActivate.

Es werden damit zwei Dinge ausgelöst:

- Das Programm wechselt von der Oberfläche in die Unit (wenn Sie zur Oberfläche zurückkehren wollen, drücken Sie die Taste F12)
- Es erscheint nachfolgender vorbereiteter Programmcode:

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

end;

end.
```

Dieser vom Delphi-Programm vorbereitete Programmcode muss von uns noch zum Leben erweckt werden.

2. Ergänzen Sie den Programmcode wie folgt:

```
procedure TForm1.FormActivate(Sender: TObject);
begin
  form1.color := clred;
end;

end.
```

3. Wechseln Sie mit F12 zur Formularansicht.

Hier hat sich nichts geändert.

4. Starten Sie das Programm.

Wenn Sie alles richtig gemacht haben, ist Ihr Formular nun rot.
Die folgende Darstellung zeigt Ihr Formular zur Entwurfszeit und zur Laufzeit.

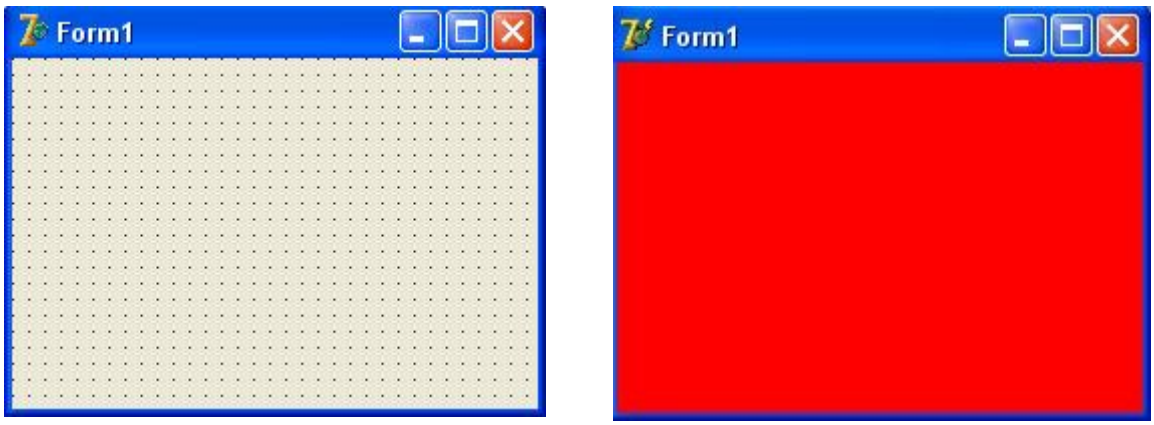


Abb. 38, 39: Formblatt zur Entwurfs- und Laufzeit

Was ist dabei passiert?

Mit dem kurzen Programmtext haben Sie auf eine Eigenschaft (nämlich die der Farbe) zugegriffen und dieser den Wert "rot" zugeordnet. Da Sie diesen Programmtext dem Ereignis OnActivate zugeordnet haben, wird die Farbe des Formblattes sofort bei der Aktivierung auf "rot" umgestellt. Der Begriff **procedure** zu Beginn des so genannten Quelltextes bedeutet, dass hier ein Programmteil beginnt. Man sagt auch: es wurde eine entsprechende Prozedur erzeugt.

5. Erzeugen Sie nun die Prozedur

OnClick (wie in Schritt 1).
Ergänzen Sie den entstehenden Programmtext (wieder zwischen begin und end; wie folgt:

```
form1.color := clgreen;
```

Wie Sie bestimmt bemerkt haben, erscheint nachdem Sie den Punkt nach Form1 eingegeben haben ein kleines Auswahlfenster, welches mit jedem Buchstaben weiter eingeschränkt wird. Hier steht Ihnen schon eine erste Delphi-Hilfe zur Verfügung. Dieses Auswahlfenster zeigt nämlich alle die Eigenschaften und Methoden an, die Sie an dieser Stelle auswählen können. Das erspart manchmal aufwendige Tipparbeit und Fehlersuche.

6. Erzeugen Sie nun die Prozedur

OnDblClick und ordnen Sie dem Formular dabei die Farbe gelb zu (`clyellow`). Ebenso können Sie die Prozeduren OnKeyDown und OnMouseMove erzeugen. Ordnen Sie diesen beiden Prozeduren die Farben blau (`clblue`) und hellgrün (`cllime`) zu.

7. Testen Sie Ihr Programm.

Sicher wird Ihnen jetzt auffallen, dass das Programm auf die Prozeduren OnClick und OnDblClick nicht mehr reagiert. Diese Annahme ist eigentlich falsch, das Problem liegt lediglich darin, dass Sie beim Klicken die Maus (wenn auch kaum wahrnehmbar) bewegen. Dies führt dazu, dass dann schon wieder die Prozedur OnMouseMove ausgelöst wird, die das Formular hellgrün färbt. Halten Sie die Maus einmal hoch und klicken Sie jetzt auf die Tasten.

8. Wir wollen daher die Prozedur OnMouseMove wie folgt abändern.

Wechseln Sie (falls noch nicht geschehen) mit F12 in die Ansicht der Oberfläche. Legen Sie eine Label-Komponente auf diese Oberfläche. Wechseln Sie wieder mit F12 in die Unit.
Suchen Sie die Prozedur OnMouseMove. Löschen Sie den von Ihnen geschriebenen Quelltext in dieser Prozedur. Schreiben Sie dafür den folgenden Programmtext:

```
label1.caption := 'Jetzt wird die Farbe nicht mehr geändert';
```

Testen Sie das Programm.

PROGRAMME / 004 / EIGEN_FORM.DPR

3.6. Ein- und Ausgabekomponenten

Für die Ein- und Ausgabe von Daten nutzt man unter anderem die folgenden Komponenten:

Komponente	Bedeutung	Zeichen
Label	dienen nur der Ausgabe von Daten bzw. für Beschriftungen	
Edit	einzeilige Ein- und Ausgabe	
Memo	mehrzeilige Ein- und Ausgabe	
ScrollBar	Eingabe von Zahlenwerten in einem bestimmten Intervall	

Viele Eigenschaften und Ereignisse dieser Komponenten sind genauso zu behandeln wie die entsprechenden Eigenschaften des Formblattes. Auf diese Eigenschaften wird hier nicht noch einmal eingegangen. Es kommen aber auch neue Eigenschaften und Ereignisse hinzu.

Im folgenden Programmbeispiel finden Sie einige wichtige Eingabekomponenten. Diesen wurden teilweise auch Ereignisse zugeordnet.

PROGRAMME / 005 / EINGABE.DPR

3.6.1. Die Labelkomponente

Abb. 44: Label-Komponente auf der Oberfläche

In einer Label-Komponente kann zur Laufzeit nur Text ausgegeben werden. Sie eignet sich nicht zur Eingabe von Daten über die Tastatur. Andererseits kann die Label-Komponente aber gut zur Beschriftung der Oberfläche eingesetzt werden.

Eigenschaften

Eigenschaft	Beschreibung
Height	Legt die Höhe der Komponente fest (hat nichts mit der Schrifthöhe zu tun - die wird über Font gesteuert)
Width	Legt die Breite der Komponente fest (Im Normalfall müssen diese beiden Eigenschaften nicht eingestellt werden, sie ergeben sich automatisch aus der Schrifthöhe und der Textlänge.)
WordWrap	Standardmäßig ist diese Eigenschaft auf False gesetzt. Das heißt der Text (Caption) wird in einer Zeile ausgegeben. Will man einen Zeilenumbruch innerhalb des Textes erreichen, muss man diese Eigenschaft auf True setzen.

Weitere Eigenschaften der Label-Komponente finden Sie in der Delphi-Hilfe.

Ereignisse vgl. Formblatt

Weitere Ereignisse der Label-Komponente finden Sie in der Delphi-Hilfe.

Programmieraufgabe 5

Wir wollen im Folgenden mit Hilfe einer Label-Komponente einige Ereignisse auslösen und dabei einige Eigenschaften dieser Label-Komponenten ändern. Erstellen Sie ein neues Projekt und speichern Sie dieses in einem neuen Ordner. Legen Sie auf dieses Formblatt zunächst eine Labelkomponente. Weiterhin benötigen Sie noch sechs Button-Komponenten. Mit Hilfe der Button werden Sie einige Eigenschaften der Label-Komponenten zur Laufzeit verändern:

Button	Was geändert werden soll
Button1	Ändern des Textes einer Label-Komponente
Button2	Ändern der Schriftfarbe des Labels
Button3	Die Hintergrundfarbe des Labels wird verändert
Button4	Schriftgröße verändern
Button5	Label soll unsichtbar werden.
Button6	Label soll wieder sichtbar werden.

Erzeugen Sie nun die Prozedur Button1Click indem Sie doppelt auf diese Komponente klicken. Es erscheint wieder ein vorbereiteter Prozedurquelltext, den Sie wie folgt ergänzen sollen:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  label1.Caption := 'Der Text hat sich in dieser Komponente verändert!';
end;
```

Testen Sie Ihr Programm und beseitigen Sie die möglichen Syntaxfehler.

Erzeugen Sie nun die Prozedur Button2Click.
Ergänzen Sie den Quelltext:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  label1.Font.Color := clblue;
end;
```

Entsprechend gehen Sie für die Prozedur Button3Click vor:

```
procedure TForm1.Button3Click(Sender: TObject);
begin
  label1.Color := clyellow;
end;
```

Button4Click:

```
procedure TForm1.Button4Click(Sender: TObject);
begin
  label1.Font.Size := 14;
end;
```

Button5Click:

```
procedure TForm1.Button5Click(Sender: TObject);
begin
  label1.Visible := false;
end;
```

Ergänzen Sie die Prozedur Button6Click selbständig.

Im Folgenden wollen wir mit einigen weiteren Label-Komponenten Ereignisse auslösen. Legen Sie dazu drei weitere Label-Komponenten auf die Oberfläche.

Label	Prozedur	Auszulösende Aktion
Label2	Label2Click	Ändern der Formularfarbe
Label3	Label3MouseMove	Text in Label3 verändern
Label4	Label4DblClick	Label4 wird durchgestrichen

Erzeugen Sie die Prozeduren mit der Ereignisseite des Objektinspektors. Die Quelltexte der einzelnen Prozeduren werden hier nur noch in gekürzter Schreibweise dargestellt.

```
  form1.Color := cllime;

  label3.Caption := 'Der Text hat sich verändert';

  label4.Font.Style := [fsstrikeout];
```

PROGRAMME / 006 / LABELKOMPONENTE.DPR bzw. LABELKOMPONENTE.EXE

3.6.2. Die Editkomponente

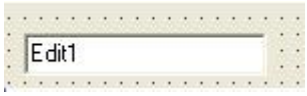


Abb. 45: Edit-Komponente auf der Oberfläche

Eine Edit-Komponente eignet sich zur einzeiligen Ein- und Ausgabe von Daten.

Eigenschaften

Eigenschaft	Beschreibung
MaxLength	gibt die maximale Anzahl der Zeichen an, die der Nutzer zur Laufzeit in diese Komponente eintragen kann.
ReadOnly	False bedeutet: Der Nutzer kann zur Laufzeit Daten in diese Komponente eintragen. True bedeutet: Der Nutzer kann zur Laufzeit keine Daten in diese Komponente eintragen.
Text	vergleichbar mit der Eigenschaft Caption bei anderen Komponenten (Caption steht bei Edit nicht zur Verfügung)

Ereignisse

Ereignis	Beschreibung
OnChange	Programm reagiert auf Änderungen in der Komponente (z.B. es wird ein Zeichen eingetragen)

Weitere Eigenschaften, Ereignisse und Methoden der Edit-Komponente finden Sie in der Delphi-Hilfe.

Programmieraufgabe 6

Erstellen Sie zunächst wieder ein neues Projekt. Mit diesem Projekt sollen einige Eigenschaften einer Edit-Komponente demonstriert werden. Außerdem ordnen wir den Komponenten verschiedene Prozeduren zu. Dazu benötigen wir eine Edit-Komponente und vier Button.

Button	Was geändert werden soll
Button1	In Edit1 soll ein Text ausgegeben werden.
Button2	Die Hintergrundfarbe von Edit1 soll verändert werden.
Button3	Die Schriftfarbe von Edit1 wird geändert.
Button4	Der Schriftstil und die Schriftgröße von Edit1 werden geändert.

Erzeugen Sie nacheinander die OnClick Prozeduren zu den einzelnen Button und geben Sie jeweils die folgenden Quelltexte ein.

```
edit1.Text := 'NEUER TEXT';

edit1.Color := clyellow;

edit1.Font.Color := clred;
edit1.Font.Name := 'Courier New';
edit1.Font.Size := 17;
```

Im Folgenden sollen einigen Edit-Komponenten zusätzlich Ereignisse zugeordnet werden. Wir benötigen dazu drei weitere Edit-Komponenten auf der Oberfläche.

Edit	Ereignis	Einstellungen im OI	Aktion
Edit2	OnChange	Eigenschaft MaxLength von Edit2 auf 5 abändern	Eingetragener Text soll in Edit1 übertragen werden.
Edit3	OnExit		Eingetragener Text soll in Edit1 übertragen werden.
Edit4	OnKeyUp		Text soll wieder gelöscht werden.

Die einzelnen Ereignisse werden dabei wieder über die Ereignisseite des OI erzeugt. Die zu ergänzenden Quelltexte sehen folgendermaßen aus:

```
edit1.Text := edit2.Text;

edit1.Text := edit3.Text;

edit1.Text := edit1.Text + edit4.Text;
edit4.Text := '';
```

Testen Sie Ihr Programm und stellen Sie dabei fest, wann die Ereignisse OnExit und OnKeyUp ausgelöst werden.

PROGRAMME / 007 / EditKomponente.DPR

3.6.3. Die Memokomponente

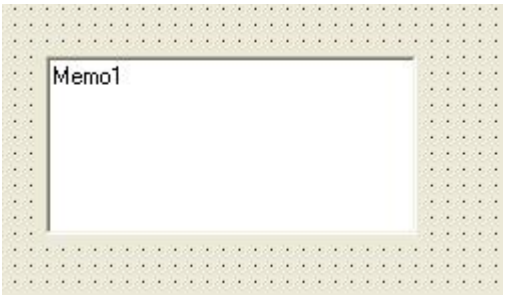


Abb. 46: Memo-Komponente auf der Oberfläche

Eine Memo-Komponente eignet sich zur mehrzeiligen Ein- und Ausgabe von Text.

Eigenschaften

Eigenschaft	Beschreibung
Lines	in eine Memo Komponente können mehrere Zeilen Text geschrieben werden. Mit einem Doppelklick auf Lines öffnet sich ein entsprechendes Fenster in dem die entsprechenden Daten geschrieben werden können.
ScrollBars	bei längerem Text können entsprechende Scrollbars links bzw. unten an die Komponente angefügt werden.

Weitere Eigenschaften, Ereignisse und Methoden der Memo-Komponente finden Sie in der Delphi-Hilfe.

Programmieraufgabe 7

Im folgenden Programm werden Sie in eine Memo-Komponente Text eintragen. Legen Sie dazu zwei Memo-Komponenten und zwei Button auf die Oberfläche.

Memo	Button	Aktion
Memo1	Button1	Hier soll vorbereiteter Text in die Memo-Komp. übertragen werden.
Memo2	Button2	Hier soll der Text aus Memo2 in Memo1 übertragen werden.

Erzeugen Sie die Prozedur Button1Click und geben Sie den nachfolgenden Quelltext ein:

```
memo1.Clear;
memo1.Lines.Add('Text wird');
memo1.Lines.Add('in diese');
memo1.Lines.Add('Memo-Komponente');
memo1.Lines.Add('geschrieben');
```

Sie können natürlich auch anderen Text eintragen lassen. Der Text kann auch aus noch mehr Zeilen bestehen. Testen Sie dann Ihr Programm.

Erzeugen Sie dann die Prozedur Button2Click und geben Sie wieder den folgenden Quelltext ein:

```
memo1.Clear;
text := memo2.Lines.Text;
memo1.Lines.Add(text);
```

PROGRAMME / 008 / MemoKomponente.DPR

3.6.4. Die Komponente Scrollbar

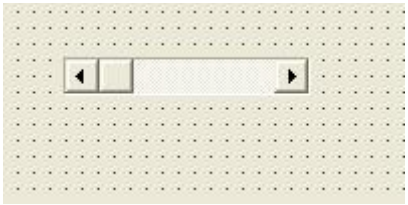


Abb. 47: ScrollBar-Komponente auf der Oberfläche

Mit einer ScrollBar können abhängig von der Position des Zeigers Integer-Zahlen in einem bestimmten Bereich eingegeben werden.

Eigenschaften

Eigenschaft	Beschreibung
Kind	legt fest ob die Komponente horizontal oder vertikal ausgerichtet ist.
Max, Min	legt den Bereich fest, aus dem die Zahlen der Komponente ausgewählt werden können. Es handelt sich dabei immer um ganze Zahlen. Standardmäßig ist festgelegt: 0 <= n <= 100

Weitere Eigenschaften, Ereignisse und Methoden der ScrollBar-Komponente finden Sie in der Delphi-Hilfe.

Programmieraufgabe 8

Das folgende Programmbeispiel zeigt Möglichkeiten zur Nutzung einer ScrollBar-Komponente. Sie werden mit den Zahlen, die über diese Komponenten eingegeben werden auch rechnen. Dabei treten Befehle auf, die Sie noch nicht kennen. Übernehmen Sie daher den Quelltext besonders sorgfältig. Zunächst benötigen wir nur eine Komponente ScrollBar und ein Label. Nehmen Sie für die ScrollBar folgende Einstellungen vor:

Eigenschaft	ändern auf
Max	200
Min	-200
SmallChange	5
Kind	sbVertical

Klicken Sie dann doppelt auf diese Komponente. Damit wird die Standardprozedur ScrollBar1Change erzeugt. Geben Sie den nachfolgenden Quelltext ein:

```
label1.Caption := inttostr (scrollbar1.Position);
```

Testen Sie Ihr Programm. Wenn alles richtig funktioniert, dann können Sie zum nächsten Schritt übergehen. Über eine ScrollBar werden Zahlen eingegeben. Mit diesen Zahlen kann man natürlich auch rechnen. Dies soll mit dem nächsten Programmteil auch geschehen. Legen Sie dazu zwei weitere ScrollBar-Komponenten und drei weitere Label auf die Oberfläche. Erzeugen Sie die Prozedur ScrollBar2Change und geben Sie den folgenden Quelltext ein:

```
procedure TForm1.ScrollBar2Change(Sender: TObject);
var su : integer;
begin
  label2.Caption := inttostr (scrollbar2.Position);
  label3.Caption := inttostr (scrollbar3.Position);
  su             := scrollbar2.Position + scrollbar3.Position;
  label4.Caption := inttostr (su);
end;
```

Die gleiche Prozedur müssen Sie noch einmal für die ScrollBar3 schreiben.

PROGRAMME / 009 / ScrollbarKomponente.DPR

3.7. Komponenten zum Auslösen von Ereignissen

Die eben besprochenen Komponenten dienen im Wesentlichen der Ein- und Ausgabe der Daten. Es können aber auch bestimmte Ereignisse mit diesen Komponenten ausgelöst werden. Sie haben es ausprobiert.
Die nun folgenden Komponenten sind im Wesentlichen dafür gedacht bestimmte Ereignisse auszulösen. Teilweise ist es auch möglich (wenn auch nicht üblich) Daten über diese Komponenten auszugeben.

Zu diesen Komponenten gehören unter anderem:

Komponente	Bedeutung	Zeichen
Button	vielfältig einsetzbare Komponente zum Auslösen von Ereignissen	
BitBtn	Button, mit vorbelegten Standardaktionen	
RadioGroup	Möglichkeit zur Auswahl zwischen mehreren Varianten	
ComboBox	Möglichkeit zur Auswahl zwischen mehreren Varianten	

3.7.1. Die Buttonkomponente

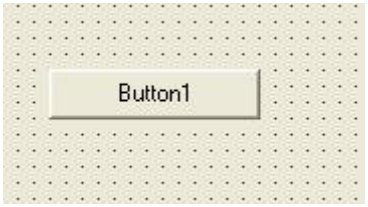


Abb. 52: Button-Komponente auf der Oberfläche

Die Button-Komponente ist die am häufigsten benutzte Komponente zum Auslösen von Ereignissen.

Eigenschaften

Eigenschaft	Beschreibung
Font	Wie bei anderen Komponenten kann von einem Button über Font die Schriftart geändert werden. Eine Einstellung der Schriftfarbe führt dabei aber zu keinem sichtbaren Ergebnis.

Weitere Eigenschaften und Ereignisse der Button-Komponente finden Sie in der Delphi-Hilfe.

Will man der Komponente das Ereignis OnClick zuordnen muss man nicht unbedingt über die Ereignisseite des Objektinspektors gehen. Das Ereignis OnClick ist dieser Komponente standardmäßig zugeordnet. Zum Auslösen der Prozedur reicht es daher aus, doppelt auf die Komponente zu klicken. In den Beispielen der vorigen Kapitel haben Sie auf diese Art die Prozeduren erzeugt. (Vielen Komponenten ist ein solches Standardereignis zugeordnet, dies ist nicht immer OnClick).

Programmieraufgabe 9

Mit einem Button können aber noch mehr als nur das OnClick-Ereignis ausgelöst werden. Einige davon werden Sie im Folgenden ausprobieren. Sie werden außerdem kennen lernen, wie Sie zwei Button auf sehr einfache Weise die gleiche Prozedur zuordnen können, ohne diese zweimal schreiben zu müssen. Sie benötigen dazu auf der Oberfläche insgesamt fünf Button-Komponenten.

Button	Ereignis	Aktion
Button1	OnClick	Änderung der Oberflächenfarbe
Button2	OnEnter	Änderung der Oberflächenfarbe
Button3	OnMouseMove	Änderung der Oberflächenfarbe
Button4	OnKeyDown	Ausgabe eines Textes auf den Button
Button5	OnClick	Zuordnung der Prozedur von Button1

Die Prozeduren zu den Button1 bis Button4 sollten Sie jetzt ohne Quelltextvorgabe schreiben können.
Um die Prozedur von Button1 dem Button5 zuzuordnen gehen Sie folgendermaßen vor:

1. Markieren Sie Button5 durch Einfachklick.
2. Wechseln Sie zur Ereignisseite des Objektinspektors.
3. Klicken Sie einmal in die Spalte neben dem Ereignis OnClick.

Abb. 53: Ereignisseite Objektinspektor

4. Mit dem kleinen Pfeil rechts klappen Sie ein Menü auf, mit dem Sie bereits vorhandene Prozeduren auswählen, die diesem Button zugeordnet werden können.

Abb. 54: Zuordnen einer vorhandenen Prozedur

5. Wählen Sie die Prozedur Button1Click aus. Das war's.

PROGRAMME / 010 / ButtonKomponente.DPR

3.7.2. Die Komponente BitBtn

Abb. 55: Darstellung eines BitBtn mit der Eigenschaft Schließen

Ein BitBtn ist ein Button, mit dem voreingestellte Ereignisse ausgelöst werden können. Die Art des BitBtn wird dabei über die Eigenschaft "Kind" eingestellt. BitBtn können ansonsten wie normale Button benutzt werden. Im Gegensatz zu normalen Button können Sie aber zusätzlich die Farbe der Beschriftung ändern.

Eigenschaften

Eigenschaft	Beschreibung
Font	Sie können die Schriftfarbe der Komponente ändern
Glyph	Auswahl einer Grafik auf dem Button
Kind	Einstellen des vordefinierten Ereignisses
Layout	Gibt an, an welcher Stelle die Grafik eingebunden wird

Weitere Eigenschaften und Ereignisse der BitBtn-Komponente finden Sie in der Delphi-Hilfe.

Programmieraufgabe 10

Hier ein einfaches Beispiel zur Verwendung eines BitBtn.

Sie möchten einen BitBtn zum Schließen des Programms benutzen. Legen Sie daher eine Komponente BitBtn (aus Komponentenliste "Zusätzlich") auf die Oberfläche. Wechseln Sie dann auf die Eigenschaftenseite des Objektinspektors und wählen Sie die Eigenschaft „Kind" aus. Stellen Sie diese Eigenschaft auf „bkClose" ein. Der Button verändert sofort sein Aussehen (vgl. Abbildung Kapitel 3.7.2.).

Starten Sie das Programm und klicken Sie auf diesen Button.

Im Programmbeispiel finden Sie noch weitere dieser Button. Diese zeigen aber im Moment noch keine direkte Auswirkung.

PROGRAMME / 011 / BitbtnKomponente.DPR

3.7.3. Die Komponente RadioGroup

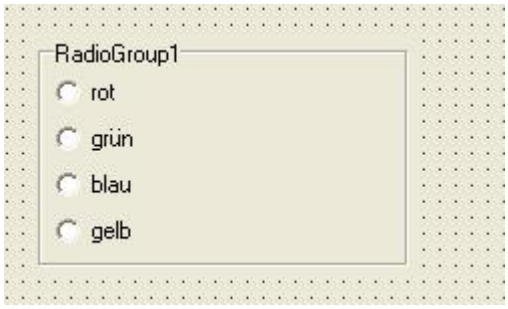


Abb. 56: RadioGroup-Komponente auf der Oberfläche

Eine RadioGroup bietet eine Auswahl für den Nutzer an. Stellen Sie sich vor, Sie möchten dem Nutzer die Auswahl der Hintergrundfarbe des Formulars selbst überlassen. Dies ist sicher mit Hilfe mehrerer Button-Komponenten möglich, aber doch recht aufwendig. Mit einer RadioGroup ist diese Auswahl übersichtlicher ausführbar.

Eigenschaften und Ereignisse der RadioGroup finden Sie in der Delphi-Hilfe.

Programmieraufgabe 11

Im Folgenden sollen mit einer RadioGroup mehrere Farbänderungen der Oberfläche ausgelöst werden. Sie benötigen dazu lediglich eine RadioGroup auf der Oberfläche.
Um eine solche Auswahl zu ermöglichen, ist es notwendig die RadioGroup zunächst mit entsprechendem Text zu versehen.
Markieren Sie die RadioGroup und klicken Sie dann doppelt im Objektinspektor auf die Eigenschaft "Items". Es erscheint der so genannte String-Listen-Editor.

Abb. 57: Ansicht des StringListenEditors mit eingetragenen Daten

Geben Sie hier den nachfolgenden Text ein:

```
rot
grün
blau
gelb
```

Klicken Sie auf OK und beobachten Sie, wie sich die RadioGroup-Komponente verändert hat. Nun wollen wir dieser RadioGroup natürlich auch noch Ereignisse zuordnen, denn mit dem Eintragen der Daten in "Items" weiß das Programm noch nicht, was es damit machen soll. Erzeugen Sie die Prozedur OnClick indem Sie doppelt auf die RadioGroup klicken (OnClick ist also wieder das Standardereignis). Ergänzen Sie den Programmcode wie folgt:

HINWEIS:

Es treten Befehle im Quelltext auf, die Sie noch nicht kennen, davon wollen wir uns aber nicht stören lassen.

```
procedure TForm1.RadioGroup1Click(Sender: TObject);
begin
  case radiogroup1.ItemIndex of
    0 : form1.Color := clred;
    1 : form1.Color := clgreen;
    2 : form1.Color := clblue;
    3 : form1.Color := clyellow;
  end;
end;
```

Testen Sie das Programm. Je nachdem welchen Auswahlpunkt Sie jetzt anklicken wird das Formular andersfarbig angezeigt. Wie Sie sicher schon vermuten, stehen die Zahlen 0 ... 3 für den jeweiligen Auswahlpunkt, der über die Eigenschaft "Itemindex" angesprochen wird.

AUFGABE:

Ändern Sie die Bezeichnungen der Farben innerhalb des Programms, z.B. wie folgt:

```
0 : form1.Color := cllime;
1 : form1.Color := claqua;
2 : form1.Color := clwhite;
3 : form1.Color := clblack;
```

Starten Sie das Programm erneut. Sie merken, dass die Farbgestaltung ausschließlich vom Quelltext abhängig ist, die Bezeichnungen in "Items" spielen dabei keine Rolle. Ändern Sie nun die Bezeichnungen in Items so ab, dass die Farben wieder passen. Das vollständige Programm finden Sie im Abschnitt ComboBox.

3.7.4. Die Komponente ComboBox

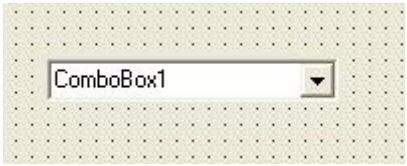


Abb. 58: ComboBox-Komponente auf der Oberfläche

Eine ComboBox funktioniert ähnlich wie eine RadioGroup. D.h. über Items können wieder mehrere Auswahlpunkte definiert werden. Eigenschaften und Ereignisse der ComboBox finden Sie wieder in der Delphi-Hilfe.

Informieren Sie sich in der Delphi-Hilfe über Eigenschaften, Methoden und Ereignisse der folgenden Komponenten. Überlegen Sie auch, wofür diese Komponenten sinnvoll eingesetzt werden können.

- ListBox
- Panel
- MaskEdit
- StringGrid
- CheckListBox
- ValueListEditor
- LabelEdEdit
- ColorBox
- Timer

Programmieraufgabe 12

Öffnen Sie das Projekt mit der RadioGroup aus dem vorhergehenden Abschnitt (Programmieraufgabe 11). Legen Sie zusätzlich eine Komponente Combobox auf die Oberfläche. Wie bei der RadioGroup müssen Sie auch hier den StringListenEditor öffnen. Geben Sie hier wieder die Farben der RadioGroup ein (rot, grün, blau, gelb). Im Gegensatz zur RadioGroup wird diese Änderung aber erst zur Laufzeit sichtbar. Klicken Sie wieder doppelt auf die ComboBox. Es wird die Standardprozedur OnChange erzeugt. Geben Sie nachfolgenden Quelltext ein:

```
procedure TForm1.ComboBox1Change(Sender: TObject);
begin
  case combobox1.ItemIndex of
    0 : form1.Color := clred;
    1 : form1.Color := clgreen;
    2 : form1.Color := clyellow;
    3 : form1.Color := clblue;
  end;
end;
```

Starten Sie das Programm und überprüfen Sie die Änderungen.

Eine ComboBox kann also immer dann eingesetzt werden, wenn auf dem Formblatt nicht genügend Platz für eine RadioGroup vorhanden ist. Im Allgemeinen sollte die Auswahl über die RadioGroup bevorzugt werden.

Das folgende Programmbeispiel zeigt die eben besprochenen Prozeduren für RadioGroup und ComboBox. Es wird außerdem eine weitere Möglichkeit zur Auslösung einer Farbänderung mit Hilfe einer ColorBox dargestellt.

PROGRAMME / 012 / RadioUndCombo.DPR

Fragen und Aufgaben
3. Einführung in die Programmierung

Grundlagen

1. Geben Sie die Unterschiede in der Arbeitsweise eines Compilers und eines Interpreters an.
2. Nennen Sie wichtige Teile der Delphi-Entwicklungsumgebung.
3. Nennen Sie einige wichtige Projektdateien und deren Inhalt.
4. Nennen Sie die wesentlichen Schritte zur Erarbeitung eines Programms.
5. Begründen Sie die Notwendigkeit eines Programmtests.
6. Woran kann man erkennen, ob sich das Programm im Entwicklungs- oder im Laufzeitmodus befindet?
7. Der Name einiger Projektdateien soll geändert werden. Wie gehen Sie vor?
8. Nennen Sie einige Eigenschaften des Formblattes.
9. Finden Sie mit der Delphi-Hilfe heraus, wann die nachfolgenden Ereignisse des Formblattes ausgelöst werden:
    a. OnClose
    b. OnKeyPress
10. Suchen Sie in der Komponentenleiste weitere Komponenten, mit denen eine Ein- und Ausgabe von Daten realisiert werden kann.
11. Informieren Sie sich mit der Delphi-Hilfe über die Nutzung einer StringGrid-Komponente. Welche Vorteile bietet diese Komponente gegenüber Edit- und Memokomponenten?
12. Nennen Sie Eigenschaften der folgenden Komponenten:
    a. Edit
    b. Button
    c. Label
    d. RadioGroup

Programmieraufgaben

13. Erstellen Sie eine neue Delphi-Application. Starten Sie diese Application mit dem grünen Pfeil. Beobachten Sie, wie sich die Darstellung des Formulars dabei ändert.

14. Suchen Sie in den Komponentenlisten die folgenden Komponenten und legen Sie diese auf die Oberfläche:
    a. Button
    b. Label
    c. RadioGroup
    d. Shape
    e. Panel
    f. Image
    g. StringGrid
    h. ScrollBar
    i. MainMenu

15. Gestalten Sie eine Oberfläche mit den folgenden Komponenten:
    a. 5 Label
    b. 4 Button
    c. 7 Edit
    d. 3 ScrollBar

16. Wie ändern sich die Bezeichnungen der Komponenten in Aufgabe 15?

17. Starten Sie das Programm aus Aufgabe 15 mit dem grünen Pfeil.

18. Löschen Sie einige der Komponenten aus Aufgabe 15.

19. Erstellen Sie ein Delphi-Programm mit den folgenden Komponenten:
    a. 1 Button
    b. 1 Label
    c. 1 Edit

    Beim Klicken auf den Button sollen die nachfolgenden Texte in die Ausgabekomponenten ausgegeben werden:
    Label: 'Hier können keine Daten eingegeben werden.'
    Edit: 'Geschafft!'

20. Lösungen/003/001/Project1.dpr
    Erstellen Sie ein Delphi-Programm mit den folgenden Komponenten:
    a. 1 Button
    b. 1 Label
    c. 1 Edit

    Beim Klicken auf den Button soll der Text aus der Edit-Komponente in die Label-Komponente übertragen werden.

21. Erstellen Sie ein Delphi- Programm mit den folgenden Komponenten:
    a. 1 Label
    b. 1 Edit

    Beim Eingeben von Daten in die Edit-Komponente sollen diese Änderungen sofort in der Label-Komponente sichtbar werden.

22. Erstellen Sie ein Delphi-Programm mit zwei Button. Beim einfachen Klicken auf den ersten Button soll das Formular grün werden. Beim Überstreichen des 2. Buttons soll das Formular gelb werden.

23. Lösungen/003/002/Project1.dpr
    Erstellen Sie mit einer RadioGroup und 5 Label-Komponenten ein Programm, bei dem je nachdem, welcher Auswahlpunkt angeklickt wird ein kurzer Text in die Label-Komponente ausgegeben wird.
    Bezeichnen Sie die Items mit Label1 ... Label5.

24. Ändern Sie das Programm aus Aufgabe 23 so ab, dass sich außerdem auch noch die Farbe der Label-Komponenten ändert.

25. Lösungen/003/003/Project1.dpr

Schreiben Sie ein Delphi-Programm mit einer ComboBox die folgende Ereignisse auslöst:
Es soll die Farbe des Formulars geändert werden.
Es soll ein Text in eine Edit-Komponente ausgegeben werden.
Es soll die Sichtbarkeit eines Labels auf "false" gesetzt werden.
Es soll die Sichtbarkeit des Labels aus c) wieder auf true gesetzt werden.

4. Variablenkonzept

4.1. Variablen in der Mathematik und in der Informatik

In der Mathematik sprechen wir immer dann von einer Variablen, wenn wir statt einer Zahl einen Buchstaben schreiben. (z.B. bei Gleichungen a, b; bei Funktionen x, y). Wir können nacheinander bestimmte Zahlenwerte in eine Gleichung einsetzen und überprüfen ob dadurch eine wahre oder eine falsche Aussage entsteht. Ein Ausdruck der Form a = a + 1 ist in der Mathematik sofort eine falsche Aussage und macht daher wenig Sinn. Andererseits stellen Ausdrücke wie z.B. x = 1 und 1 = x in der Mathematik den gleichen Sachverhalt dar.
Etwas anders ist es in der Programmierung. Hier werden einzelnen Variablen Werte zugewiesen. Ein Ausdruck der Form a := a + 1 ergibt für den Computer durchaus einen Sinn, wohingegen der Ausdruck 1 := x vom Compiler mit einer Fehlermeldung beantwortet wird.

Was passiert bei einer solchen Wertzuweisung?

Nehmen wir das Beispiel a := a + 1.
Vor der Ausführung hat die Variable einen bestimmten Wert angenommen. Dieser Wert ist im Arbeitsspeicher des Computers abgelegt. (Nehmen wir einmal an, dieser Wert sei 3).

Mit dem Beginn der Anweisung:	`a :=`	weiß der Computer, dass der Variablen a ein neuer Wert zugeordnet werden soll (das ist immer die linke Seite der Anweisung).
die rechte Seite der Anweisung:	`a + 1`	bedeutet für den Computer: lies den Wert a aus dem Speicher (wir hatten uns auf 3 geeinigt), addiere zu diesem Wert 1 (also 3 + 1 = 4) und ordne diesen neuen Wert der linken Seite zu. Schreibe diesen neuen Wert in den Speicher. Nach Ablauf der Anweisung steht also im Speicher die Zahl 4 für die Variable a.

Man kann einer Variablen natürlich auch nur eine Zahl zuordnen. Dies würde wie folgt aussehen:
`b := 25;`

AUFGABE:

Geben Sie bei den folgenden Wertzuweisungen an, welchen Wert die Variablen nach jedem Schritt des Programmdurchlaufs angenommen haben.

a)	b)	c)

a := 4;
b := 5;
a := a + b;
	a := 2;
	a := a + 2;
	a := a + a;
		a := 16;
		b := 2;
		c := 5;
		a := a * b - c;
		a := a / 3;

In den Aufgaben wird auch deutlich, dass man einer Variablen, bevor man sie das erste Mal auf der rechten Seite verwendet, einen Anfangswert zuordnen muss. Dies nennt man auch

Initialisieren der Variablen.

4.2. Einfache Variablentypen

In der Programmierung unterscheidet man im Wesentlichen 3 Arten von Variablentypen

- die einfachen Variablentypen
- die strukturierten Variablentypen
- die Zeigertypen

Wir wollen uns hier zunächst mit den einfachen Variablen befassen. Die strukturierten Variablen und die Zeigertypen werden in späteren Themen gesondert behandelt.

4.2.1. Zahlen in der Programmierung

In der Mathematik unterscheiden wir mehrere Zahlenbereiche (natürliche Zahlen, ganze Zahlen, rationale Zahlen, reelle Zahlen). Ähnliche "Zahlenbereiche" gibt es auch in der Programmierung. Der wesentliche Unterschied zwischen den Zahlenbereichen in der Programmierung und Zahlenbereichen in der Mathematik besteht darin, dass die Computerzahlen immer eine endliche Menge darstellen (auch wenn die Menge noch so groß ist).

Hier eine Auflistung der wichtigsten Zahlentypen (diese Auflistung ist bei weitem nicht vollständig, repräsentiert aber die in der Anwendung am häufigsten benutzten Typen. Viele weitere Typen finden Sie in der Delphi-Hilfe.)

Datentypen für ganze Zahlen (Integer-Typen):

Typ	Bereich
Integer	abzählbarer Typ, entspricht den ganzen Zahlen in der Mathematik Wertebereich: - 2 147 483 648 ... 2 147 483 647
Byte	kurze ganze Zahl Wertebereich: 0 ... 255
Int64	vorzeichenbehaftete ganze Zahl Wertebereich: -2^63 ... 2^63-1

Datentypen für rationale Zahlen (Real-Typen):

Typ	Bereich
Real	klassischer Typ für Gleitkommazahlen (doppelt genaue Gleitkommazahl) Wertebereich: 5,0*10^-324 ... 1,7*10^308 Genauigkeit: 15 bis 16 Stellen
Extended	extremgenaue Gleitkommazahl Wertebereich: 1,9*10^-4951 ... 1,1*10^4932 Genauigkeit: 19 bis 20 Stellen

Nun kann man sich fragen, warum z.B. für ganze Zahlen mehrere Typen vorhanden sind. Das hat ganz einfach mit dem Speicherplatz zu tun, den eine Zahl belegt. Definiert man z.B. die Zahl 1 als Int64 wird für diese Zahl wesentlich mehr Speicherplatz benötigt, als für die Zahl 1 im Bereich Byte. Wenn man also von vornherein weiß dass eine Zahl nicht größer wird als 255 kann man Speicherplatz sparen, indem man diese statt als Integer als Byte deklariert.

HINWEIS:

Zahlen mit Komma werden im Programmtext immer in der Form 2.3 (zwei Punkt drei) geschrieben.

4.2.2. Textvariablen

Textvariablen dienen in der Programmierung der Speicherung und Verarbeitung von Zeichenketten. Diese Zeichen werden mit Hilfe des ASCII-Kodes kodiert. Daher hat z.B. das Zeichen '1' im Computer eine andere Kodierung als die Zahl 1. Wurde also eine Variable als Textvariable deklariert, dann kann mit diesem Zeichen nicht gerechnet werden, selbst dann nicht wenn es sich bei diesen Zeichen um Ziffern handelt.

	Beachten Sie, dass mit den Eingabekomponenten Edit und Memo nur Zeichenketten eingegeben werden können.

Variablentypen für Zeichenketten

Typ	Bereich
ShortString	Zeichenkette mit max. 255 Zeichen
AnsiString	Zeichenkette mit unbegrenzter Länge
String	entspricht je nach Compilereinstellung dem ShortString bzw. dem AnsiString

Die Länge eines Strings kann vom Programmierer begrenzt werden. Soll vom Programm lediglich eine Zeichenkette der Länge 20 Zeichen (einschließlich der Leerzeichen) verarbeitet werden, dann schreibt man `string [20].`
Der Nutzer kann hier zwar mehr als 20 Zeichen eingeben, vom Programm werden aber nur die ersten 20 Zeichen des Strings verarbeitet.
Soll ein String im Programmtext verwendet werden, dann muss dieser in Hochkomma ' ' geschrieben werden.

BEISPIEL:

```
name := 'Lieschen Müller';
```

4.2.3. Wahrheitswerte

Eine besondere Klasse der Variablen sind die Wahrheitswerte, diese können nur genau zwei Werte annehmen, nämlich "true" und "false". Sie haben diese Werte bereits benutzt als Sie z.B. die Sichtbarkeit einer Label-Komponente verändert haben.

Variablentyp für Wahrheitswerte

Typ	Bereich
Boolean	Kann genau zwei Werte TRUE (wahr) und FALSE (falsch) annehmen

4.2.4. Deklaration und Initialisierung von Variablen

Jede Variable, die Sie in einem Programm verwenden ist gekennzeichnet durch

- Name (Bezeichner)
- Typ
- Wert
- Speicheradresse

Wenn wir ein Programm schreiben ist es daher notwendig die benutzten Variablen zu deklarieren. Das heißt, Sie müssen dem Programm vor der Verwendung mitteilen, welche Variablen Sie benutzen wollen. Diese Deklaration erfolgt vor dem begin in der Prozedur. Dabei legen Sie den

Namen und den Typ der Variablen fest. Vom Programm wird dadurch der benötigte Speicherplatz für diese Variable reserviert und es "merkt" sich die entsprechende Speicheradresse.

BEISPIEL:

Sie wollen in einem Programm die Variablen a und b vom Typ Integer und die Variable name vom Typ String benutzen. Die entsprechende Deklaration muss dann wie folgt aussehen.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  a, b : integer;
  name : string;
begin
  ...
```

Nun werden vom Programm also Speicherplätze für die Integer-Werte und einen String bereitgestellt.

Die Deklaration hat insgesamt die folgende Syntax:

var	gibt an, dass jetzt eine Variablendeklaration erfolgt
`a, b`	sind die Namen der Variablen
`: integer;`	nach dem Doppelpunkt folgt der Typ, den die Variablen haben sollen.

Hiermit stellt sich jetzt die Frage, welche Namen eine Variable überhaupt annehmen darf. Dazu sind die nachfolgenden Regeln zu beachten:

- Ein Variablenname darf nur einmal in der Prozedur deklariert werden. (Dabei unterscheidet das Pogramm nicht zwischen der Groß- und Kleinschreibung, d.h. a und A symbolisieren die gleiche Variable)
- Ein Variablenname muss immer mit einem Buchstaben beginnen
- Umlaute und Sonderzeichen sind nicht zugelassen.
- Innerhalb der Variablen dürfen Ziffern und das Zeichen _ vorkommen

Beispiele für gültige Variablen:

`a, b, a1, a2, a_1, a_2, a11, ab1, name, vorzeichen, wert, position ...`

Beispiele für ungültige Variablen:

`1a, 2c, für, x$y, f-g, z(1), ...`

Wie wir oben bereits besprochen haben, wird für jede Variable nach der Initialisierung Speicherplatz zur Verfügung gestellt. Das heißt aber noch lange nicht dass dieser Speicherplatz leer sein muss. Oftmals befinden sich hier noch irgendwelche Daten, die nicht mehr benötigt werden. Wenn man nun mit dieser Variablen arbeiten will, wird dieser Speicherplatz aufgerufen. Hat das Programm hier noch keine Daten neu eingeschrieben, wird der vorhandene "Speichermüll" gelesen. Das kann zu vielen Fehlern führen und den Programmierer, der den Fehler sucht schlicht in den Wahnsinn treiben.

	Daher muss jede Variable vor der ersten Benutzung initialisiert werden.

Dies bedeutet nichts anderes, als dass der Variablen ein Anfangswert zugeordnet wird.

BEISPIEL:

```
procedure TForm1.Button1Click(Sender: TObject);
var
   a : integer;
begin
   a := 1;
   ...
```

4.2.5. Zuweisungskompatibilität von Variablen

Manchmal reagiert der Compiler auf eine Wertzuweisung der Form a := b + 1 mit einer Fehlermeldung, obwohl doch die Syntax dieser Anweisung richtig ist. Wo liegt der Fehler?
Hier hilft es, sich die Fehlermeldung des Compilers genau durchzulesen. Oftmals findet man eine Meldung der Form "Incompatible Typen ...". Der Fehler liegt also in der Variablendeklaration. Es ist z.B. nicht möglich einer Integer-Variablen einen Wert vom Typ Real zuzuordnen.
Welche Zuweisungen sind aber erlaubt? Die folgende Tabelle soll einen Anhaltspunkt für mögliche Zuweisungsverträglichkeiten liefern.

linke Seite	rechte Seite	Kompatibel?
Real	Integer	JA
Integer	Real	Nein

4.3. Nutzung der Ergibt-Anweisung für Berechnungen

Wir erinnern uns:

Will man einer Variablen einen Wert zuweisen, dann muss auf der linken Seite der Variablenname und auf der rechten Seite die entsprechende Berechnung bzw. Zuordnung stehen. Die Berechnung muss in der Regel den gleichen Variablentyp ergeben, mit der die Variable deklariert wurde. Beide Seiten werden durch := getrennt.

4.3.1. Rechnen mit Real-Zahlen

In diesem Kapitel werden Sie nun endlich auch mit Zahlen rechnen, indem Sie einen kleinen Taschenrechner entwickeln. Dieser wird zwar vorerst nur die vier Grundrechenoperationen umfassen, aber für den Anfang ist dies schon ausreichend. Dabei stoßen Sie aber gleich auf ein neues Problem:

Wie können Zahlen eingegeben werden?

Aus den vorigen Kapiteln wissen Sie, dass in Edit-Komponenten nur Text eingegeben werden kann. Dabei werden auch einzelne Ziffern nur als Zeichen, aber nicht als Zahlen aufgefasst. Innerhalb des Programms kann mit den Zahlen zwar ganz normal gerechnet werden. Problematisch wird aber die Ein- und Ausgabe der Zahlen. Der Nutzer soll ja schließlich seine eigenen Zahlen für die Berechnung eingeben können und das Ergebnis soll dann auch in einer Ausgabekomponente sichtbar werden. Es ist zwar möglich Zahlen über mehrere ScrollBar-Komponenten einzugeben, diese Methode ist dann aber wohl doch sehr ungewöhnlich und ineffektiv. Die Zahlen, die daher vom Nutzer in eine Edit-Komponente eingegeben werden, müssen demzufolge erst in eine "richtige" Zahl umgewandelt werden. Dazu benutzen wir beim Zahlentyp Real die Standardfunktion `Strtofloat`. Standardfunktionen werden ähnlich wie die Funktionen auf Ihrem Taschenrechner verwendet, Sie geben den Wert ein und drücken eine Taste, was intern passiert ist uns dabei (vorerst) egal.

Der Funktionsaufruf sieht ziemlich kompliziert aus. Wenn wir diesen etwas "zerpflücken" kann man ihn sich besser merken:

`str`	Wandle einen String
`to`	in eine
`float`	Real-Zahl um.

Andersherum verfahren wir, wenn eine berechnete Zahl wieder in eine Ausgabekomponente ausgegeben werden soll. Die dazugehörige Funktion lautet: Floattostr, also:

`float`	Wandle eine Real-Zahl
`to`	in einen
`str`	String um.

Deutlicher wird dies alles wenn wir es uns in einer kurzen Programmsequenz anschauen:

```
procedure TForm1.Button1Click(Sender: TObject);
var a, b, c : real;                 // Variablendeklaration
begin
  a := strtofloat (edit1.Text);     // liest eine Zahl aus Edit1
  b := strtofloat (edit2.Text);     // liest eine Zahl aus Edit2
  c := a + b;                       // berechnet c
  edit3.Text := floattostr (c);     // gibt c in Edit3 aus
end;
```

In diesem Programmtext wurde auch eine einfache Möglichkeit zum Kommentieren von Quelltexten verwendet. Texte hinter zwei Schrägstrichen werden vom Compiler ignoriert. Sie helfen aber oft zum Verständnis eines Programms.

Programmieraufgabe 13

Jetzt ist an der Zeit endlich mit dem Taschenrechner zu beginnen.

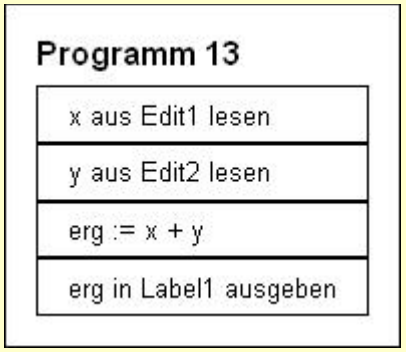
Erstellen Sie zunächst wieder ein neues Projekt. Legen Sie die Komponenten nach der folgenden Tabelle auf die Oberfläche und ändern Sie die Eigenschaften entsprechend ab.

Komponente	Eigenschaft	Einstellung
Edit1	Text	1
Edit2	Text	2
Label1	Caption	Ergebnis
Button1	Caption	+
Button2	Caption	-
Button3	Caption	*
Button4	Caption	/

Gestalten Sie Ihre Oberfläche weiter aus, indem Sie Schriftgrößen und Farben nach Ihren Vorstellungen verwenden. Die fertige Oberfläche könnte z.B. folgendermaßen aussehen:

Abb. 59: Beispiel für eine fertige Oberfläche

Erzeugen Sie nun das OnClick-Ereignis des Button1. Mit diesem Button soll die Addition ausgeführt werden. Struktogramm und Quelltext sehen dazu wie folgt aus:




```
procedure TForm1.Button1Click(Sender: TObject);
var x, y, erg : real;
begin
  x   := strtofloat(edit1.Text);
  y   := strtofloat(edit2.Text);
  erg := x + y;
  label1.Caption := floattostr (erg);
end;
```

Es sollte Ihnen nun möglich sein die drei fehlenden Struktogramme und Prozeduren allein zu erstellen. Das vollständige Programm finden Sie hier:

PROGRAMME / 013 / Taschenrechner_1.DPR

4.3.2. Rechnen mit Integer-Zahlen

Das Rechnen mit Integer-Zahlen erfolgt prinzipiell wie bei den Real-Zahlen. Da die Variablen hierbei aber als Integer (Ganzzahlig) definiert werden, dürfen natürlich keine Kommazahlen vom Nutzer eingegeben werden. Die Rechenoperationen Addition, Subtraktion und Multiplikation können mit den bekannten Operationszeichen ausgeführt werden, da die Ergebnisse stets wieder ganzzahlig werden.

Etwas anderes ist es mit der Division. Das Ergebnis einer Division zweier ganzer Zahlen kann natürlich auch eine gebrochene Zahl werden.

Daher sind bei der Arbeit mit Integer-Typen die folgenden Regeln zu beachten:

- Die Umwandlung von einem String in eine Integer-Zahl erfolgt mit StrToInt (was steckt dahinter?).
- Die Umwandlung von einer Integer-Zahl in einen String erfolgt mit IntToStr.
- Während die Addition, Subtraktion und Multiplikation ganzer Zahlen immer ausführbar ist, können bei der Division zweier Integer-Zahlen (sofern das Ergebnis wieder einer Integer-Variablen zugeordnet werden soll) Probleme auftreten. Daher kann das Operationszeichen "/" hier nicht mehr angewendet werden.
- Wir merken uns:
  - Soll das Ergebnis der Division zweier Integer-Zahlen einer Real-Variablen zugeordnet werden, dann verwenden wir "/".
  - Soll das Ergebnis der Division dagegen einer Integer-Variablen zugeordnet werden muss div verwendet werden. Div gibt dabei den ganzzahligen Anteil der Division zurück.

BEISPIEL:

Nach Ausführung von `a := 9 div 4` hat a den Wert 2.

Manchmal möchte man aber auch wissen, wie groß der Rest bei einer Division ist, dann kann man z.B. rechnen:

```
a    := 9 div 4;
rest := 9 - 4*a;
```

Dieses kurze Programm berechnet zwar den Rest der Division, aber ziemlich umständlich. Leichter geht es mit der Anweisung `mod`.

```
rest := 9 mod 4;
```

Programmieraufgabe 14

Der Taschenrechner soll jetzt um die folgenden Funktionen erweitert werden:

- Berechnen einer Division zweier natürlicher Zahlen.
- Anzeigen des Restes bei dieser Division.

Öffnen Sie dazu das Projekt Taschenrechner aus dem vorigen Abschnitt (Programmieraufgabe 13) und legen Sie eine weitere Button-Komponente auf die Oberfläche und erzeugen Sie die Prozedur OnClick. Ergänzen Sie den Quelltext wie folgt:

```
procedure TForm1.Button5Click(Sender: TObject);
var x, y, erg, rest : integer;
begin
  x := strtoint (edit1.Text);
  y := strtoint (edit2.Text);
  erg := x div y;
  rest := x mod y;
  label1.Caption := inttostr (erg) + ' Rest ' + inttostr (rest);
end;
```

AUFGABE:

Erstellen Sie das Struktogramm zu dieser Prozedur.

Im Programm wurde außerdem eine Verknüpfung von Strings ausgeführt. Mit diesen werden Sie sich im nächsten Abschnitt näher beschäftigen.

PROGRAMME / 014 / Taschenrechner_2.DPR

4.3.3. Operationen auf String-Variablen

Mit String-Variablen lässt sich nur eine Operation sinnvoll ausführen. Man kann String-Variablen "addieren". Besser sagt man hier, es werden zwei Strings miteinander verknüpft. Was passiert bei so einer Verknüpfung?

Wir stellen uns vor, die Variablen a, b und c wurden als String deklariert.
Anschließend wurden die folgenden Programmzeilen eingegeben:

```
a := 'Ich heiße ';
b := 'Lieschen Müller.';
c := a + b;
```

Wie Sie sicher richtig vermuten hat c jetzt den Wert: 'Ich heiße Lieschen Müller.'
Natürlich könnte man jetzt auch schreiben:

```
c := c + ' und wohne in Haldensleben.'
```

Übrigens müssen Strings beim Lesen aus Eingabekomponenten nicht erst in einen String umgewandelt werden. (logisch oder?)

Im vorigen Programmbeispiel trat die folgende Zeile auf:

```
label1.Caption := IntToStr (erg) + ' Rest ' + IntToStr (rest);
```

Hierbei handelte es sich ebenfalls um die Verknüpfung mehrerer Strings. Achten Sie immer darauf, dass Zahlen zunächst in einen String umgewandelt werden müssen.

4.3.4. Nutzung von Wahrheitswerten

Oftmals muss man auch zwei Werte auf die verschiedenste Art miteinander vergleichen. Hierbei spielen die Wahrheitswerte eine entscheidende Rolle. Auf diese Thematik kommen wir aber noch einmal intensiver zurück, wenn wir uns mit logischen Ausdrücken beschäftigen.

4.4. Lokale und globale Variablen

Schon wieder zwei seltsame Begriffe, die sich aber recht einfach erklären lassen:

In manchen Programmen braucht man für verschiedene Prozeduren immer wieder die gleichen Variablen, die dazu auch noch bestimmte Werte aus der vorhergehenden Prozedur benötigen. Nun haben aber Variable die innerhalb einer Prozedur deklariert wurden (lokale Variable), das Problem, dass sie den Wert nach Durchlauf durch diese Prozedur einfach wieder "vergessen" (man sagt auch der Speicherplatz wird wieder freigegeben).

Anders verhält es sich mit den globalen Variablen, diese werden bereits viel weiter oben im Programm-Quelltext (nämlich bereits vor der ersten Prozedur deklariert). Die Variablendeklaration erfolgt analog zur bisher behandelten Deklaration. Wir können diesen Variablen aber noch zusätzlich einen so genannten Startwert zuordnen. Dieser Startwert, das sagt schon der Name, wird beim Starten des Programms der Variablen zugeordnet (es handelt sich also um eine Initialisierung der Variablen).

Sollten wir hier aber die Initialisierung vergessen oder nicht wollen, so ist das nicht weiter schlimm, denn globale Variablen werden vom Programm automatisch initialisiert (Zahlen mit 0, Strings mit '', Wahrheitswerte mit false). In anderen Themen werden wir uns ausführlicher damit beschäftigen.

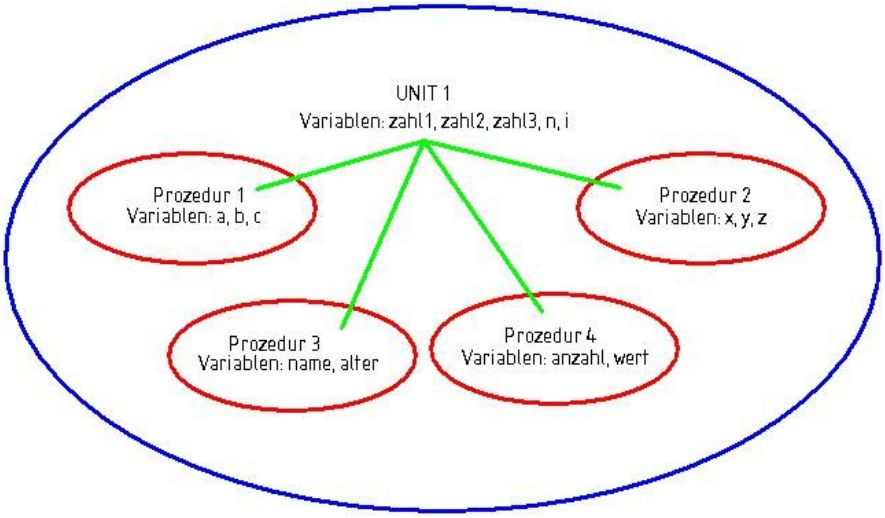


Abb. 61: Lokale und globale Variablen

In der Grafik ist diese Problematik dargestellt. Die Variablen zahl1, zahl2, zahl3, n und i sind globale Variablen und können daher von allen Prozeduren genutzt werden.

Die Variablen a, b, c stehen nur der Prozedur 1 zur Verfügung und können daher von den anderen Prozeduren nicht genutzt werden.

Die Deklaration lokaler Variablen haben Sie bereits mehrmals vorgenommen. Die Deklaration globaler Variablen könnte wie folgt aussehen. (Ansicht einer Unit mit global deklarierten Variablen):

```
unit Unit1;

Interface

uses
  Windows, Messages, SysUtils, ...

type
  TForm1 = class(TForm)
  ...

private
  { Private-Deklarationen }

public
  { Public-Deklarationen }

end;

var
  Form1: TForm1;
  b : integer;        // globale Variable ohne Initialisierung
  c : integer = 5;    // globale Variable mit Initialisierung

implementation

{$R *.dfm}
```

Im Programmbeispiel werden lokale und globale Variablen mit und ohne Initialisierung ausgegeben. Sehen Sie sich den Quelltext dieses Programms genau an und achten Sie auf die verschiedenen Ausgaben der Daten.

PROGRAMME / 015 / VARIABLEN1.DPR

4.5. Kodierung von Zahlen und Zeichen

Jetzt wird es erst mal wieder sehr theoretisch.

In den letzten Kapiteln haben Sie sehr viel über Variablen, deren Deklaration und Verwendung gelernt. Wie aber verarbeitet ein Computer diese Variablen und warum besteht ein Unterschied zwischen dem Zeichen '1' und der Zahl 1?

Computer sind nur in der Lage zwei verschiedene Zustände: Strom fließt bzw. Strom fließt nicht zu unterscheiden. Entsprechend können alle Zahlen und Zeichen nur mit diesen beiden Zuständen vom Computer verarbeitet werden.

Dem Zustand "Strom fließt" ordnen wir im Folgenden die Ziffer 1 zu, dem Zustand "Strom fließt nicht" die Ziffer 0. Daraus ergibt sich, dass alle Zahlen und Zeichen, aber auch Farben und Grafiken nur mit Hilfe der Zeichen 0 und 1 kodiert werden können.

4.5.1. Binär- und Hexadezimalzahlen

Das Dezimalsystem

Im Dezimalsystem bildet die Zahl 10 die Grundlage für alle Zahlen und Berechnungen. Zur Darstellung einzelner Zahlen dienen dabei die Ziffern 0, 1, 2, ... 9 also insgesamt zehn Ziffern. Erst durch die Stellung innerhalb der gesamten Zahl wird dabei die Bedeutung der einzelnen Ziffer klar. So hat z.B. die Ziffer 3 in 13 eine andere Bedeutung als in 31.
Die Zahlen werden dabei entsprechend ihrer Stellung und Potenz betrachtet.

So bedeutet:
123 = 1 * 10^2 + 2 * 10^1 + 3 * 10^0

Das Dualsystem (Binärsystem)

Grundlage für das Verständnis der Arbeitsweise eines Computers bildet das duale Zahlensystem (oder auch Binärsystem).

Im Gegensatz zu dem Ihnen bekannten Dezimalsystem gibt es in diesem Zahlensystem nur zwei Grundziffern, die 0 und die 1. Alle Zahlen des Dezimalsystems können in dieses Dualsystem umgewandelt werden. Die beiden Ziffern werden oft auch als 0 und L geschrieben. Die Zahlen werden entsprechend der Potenzen von 2 in der Gesamtzahl angeordnet.

Es bedeutet also:

1010 = 1 * 2^3 + 0 *2^2 + 1 * 2^1 + 0 * 2^0
(= 10 im Dezimalsystem)

Hier wird bereits ersichtlich, dass es wichtig ist die Basis des Zahlensystems zu kennen, um eine klare Aussage über den entsprechenden Wert machen zu können.

Umwandlung einer Dezimalzahl in eine Dualzahl

Um eine Dezimalzahl in eine Dualzahl umzuwandeln gibt es zwei Möglichkeiten:

- Umwandlung durch Zerlegung
- Umwandlung durch Division mit Rest

BEISPIEL:

Umwandlung durch Zerlegung

Zuerst zerlegt man eine gegebene Dezimalzahl in eine Summe aus Zweierpotenzen:

Beispiel: 25 = 16 + 8 + 1

Schreibt dann die entsprechenden Zweierpotenzen auf:

Beispiel: 25 = 1*2^4 + 1*2^3 + 0*2^2 + 0*2^1 + 1*2^0

und setzt dann die Dualzahl zusammen:

Beispiel: $(25)_{10} = (11001)_2$ = LL00L

Umwandlung durch Division mit Rest

Hierbei wird die gegebene Dezimalzahl wiederholt durch 2 dividiert und der dabei auftretende Rest festgestellt.

Beispiel:
25 : 2 = 12 Rest 1
12 : 2 = 6 Rest 0
6 : 2 = 3 Rest 0
3 : 2 = 1 Rest 1
1 : 2 = 0 Rest 1

Es müssen jetzt nur noch die berechneten "Reste" in umgekehrter Reihenfolge aufgeschrieben werden.

Beispiel: LL00L

Rechnen mit binären Zahlen

Um Aufgaben mit den Grundrechenoperationen im Dezimalsystem lösen zu können (ohne Taschenrechner) hat man die Aufgaben jeweils auf eine bestimmte Anzahl von Grundaufgaben der Addition und Multiplikation zurückgeführt. Diese Grundaufgaben kann man dann auch in einer Tabelle zusammenfassen. Für die Addition würde diese folgendermaßen aussehen:

+	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9	10
2	2	3	4	5	6	7	8	9	10	11
3	3	4	5	6	7	8	9	10	11	12
4	4	5	6	7	8	9	10	11	12	13
5	5	6	7	8	9	10	11	12	13	14
6	6	7	8	9	10	11	12	13	14	15
7	7	8	9	10	11	12	13	14	15	16
8	8	9	10	11	12	13	14	15	16	17
9	9	10	11	12	13	14	15	16	17	18

Um zu addieren sucht man sich die gewünschten Ziffern in der oberen Zeile bzw. in der linken Spalte und findet das Ergebnis an der entsprechenden Stelle innerhalb der Tabelle. Bei der Addition ergeben diese Grundaufgaben bereits 100 einzelne Aufgaben.

Addition von Dualzahlen

Auch für das Rechnen mit Dualzahlen lässt sich eine entsprechende Tabelle mit Grundaufgaben zusammenstellen:

+	**0**	**L**
0	0	L
L	L	L0

Hier hat man aber im Gegensatz zum Dezimalsystem nur noch 2 * 2 = 4 Grundaufgaben.

Schriftliches Addieren von Dualzahlen

Es soll die Aufgabe: L00LLL + LLL0L gelöst werden.

Natürlich wäre es jetzt möglich beide Zahlen zunächst in eine Dezimalzahl umzuwandeln, diese Dezimalzahlen zu addieren und dann das Ergebnis wieder in eine Dualzahl umzurechnen. Dies ist aber ein Weg der vom Computer natürlich nicht gegangen werden kann. Wie lösen wir diese Aufgabe aber mit den Dualzahlen?

Zunächst einmal schreiben wir die Werte entsprechend ihrer Potenz untereinander:

```
     L00LLL
+     LLL0L
```

Nun kann mit Hilfe der Tabelle der Grundaufgaben addiert werden. Ähnlich wie beim Zehnersystem erhalten wir auch hier Überträge, die in der nächsten Stelle beachtet werden müssen.

```
   L00LLL
    LLL0L
   LLLLL
  L000L00
```

Darstellung des Rechenweges:

1. L + L = L0
   0 schreiben, L als Übertrag in die nächste Spalte
2. (L + 0) + L = L + L = L0
   0 schreiben, L als Übertrag in die nächste Spalte
3. (L + L) + L = LO + L = LL
   L schreiben L als Übertrag in die nächste Spalte
4. (0 + L) + L= L0
   0 schreiben, L als Übertrag in die nächste Spalte
5. (0 + L) + L = L + L = L0
   0 schreiben, L als Übertrag in die nächste Spalte
6. L + L = L0
   L0 schreiben

Das Hexadezimalsystem

Wozu noch ein Zahlensystem?

... werden Sie sich jetzt fragen. Wie Sie sicher bereits bemerkt haben, werden die Dualzahlen ziemlich lang und unübersichtlich. Um eine kürzere und übersichtlichere Schreibweise der Kodierung im Computer zu erhalten, wird das Hexadezimalsystem verwendet. Dieses Zahlensystem bietet gegenüber den Dezimalzahlen folgende Vorteile:

- Die Zahlen werden noch kürzer.
- Eine zweistellige Hexadezimalzahl entspricht genau einem Byte Speicherbelegung.
- Die Umwandlung einer Binärzahl in eine Hexadezimalzahl ist sehr einfach.

Dualzahlen entsprechen in der Computertechnik genau 1 bit. Ein Bit bedeutet, dass genau ein Zustand (Strom fließt oder fließt nicht) eintritt. Mit diesem einen Bit lassen sich aber nur genau zwei Werte repräsentieren. Das ist natürlich für die Darstellung von Ziffern, Buchstaben und anderen Zeichen viel zu wenig. In der Computertechnik werden daher insgesamt 8 solcher einzelnen Bits zusammengefasst = 1 Byte. Das heißt eine 8-stellige Dualzahl repräsentiert alle möglichen Werte. Für bestimmte Kodierungen (z.B. Farben) werden auch noch längere Kombinationen verwendet.

BEISPIEL: 0L0LL00L ➔ Codierung für das Zeichen "Y"

Auf diese Weise lassen sich insgesamt 256 (von 00000000 bis LLLLLLLL) Zeichen darstellen. Für den Computer ergeben diese Kodierungen einen Sinn. Leider sind diese Zeichen für den Menschen nur schwer lesbar, da die Kodierung sehr lang erscheint. Andererseits ist die Umwandlung in eine Dezimalzahl (bzw. auch die Umwandlung aus einer Dezimalzahl in die entsprechende Kodierung) sehr mühselig. Um die Übersicht über die Zeichen zu gewährleisten und die Darstellung der Zeichen auf eine kürzere "Zahl" zu begrenzen nutzt man in der Informatik sehr häufig das Hexadezimalsystem. Die Basis dieses Zahlensystems bildet die Zahl 16. (eine Potenz von 2: 2^4 = 16).

Die Ziffern des Hexadezimalsystems

Um eine hexadezimale Zahl darzustellen benötigen wir auch 16 Ziffern. Für die ersten 10 Ziffern nutzt man dabei die bekannten Ziffern 0, 1, 2, ... 9. Die fehlenden sechs Ziffern werden in alphabetischer Reihenfolge mit den Großbuchstaben A, B, C, D, E, F aufgefüllt.

Es bedeutet daher:

Hexadezimal	Dezimal	Hexadezimal	Dezimal
0	0	8	8
1	1	9	9
2	2	A	10
3	3	B	11
4	4	C	12
5	5	D	13
6	6	E	14
7	7	F	15

Umwandeln einer Hexadezimalzahl in eine Dezimalzahl

Welchen Dezimalwert hat die Zahl $(1AE2)_{16}$?

Wir gehen wieder entsprechend der Stellen vor:

= 1 * 16^3 + 10 * 16^2 + 14 * 16^1 + 2 * 16^0
= 1* 4096 + 10 * 256 + 14 * 16 + 2 * 1
= 4096 + 2560 + 224 + 2
= 6882

Umwandeln einer Dezimalzahl in eine Hexadezimalzahl

Für diese Umwandlung sollte grundsätzlich das Divisionsverfahren eingesetzt werden. Dies erspart viel Rechenaufwand.

Beispiel: 6882 soll in eine Hexadezimalzahl umgewandelt werden:

```
6882 : 16 = 430   Rest  2          2
 430 : 16 =  26   Rest 14          E
  26 : 16 =   1   Rest 10          A
   1 : 16 =   0   Rest  1          1
```

Abschließend die sich ergebenden Reste wieder in umgekehrter Reihenfolge aufschreiben:

$(6882)_{10} = (1AE2)_{16}$

Umwandeln einer Hexadezimalzahl in eine Dualzahl

Um eine Hexadezimalzahl in eine Dualzahl umzuwandeln könnte man natürlich diese erst in eine Dezimalzahl umwandeln und diese dann wieder in die Dualzahl. Es geht aber wesentlich einfacher. Dies ergibt sich aus der Tatsache, dass das Hexadezimalsystem als Basis eine Potenz der 2 besitzt.

BEISPIEL:

Es ist die Hexadezimalzahl 3B in eine entsprechende Dualzahl umzuwandeln.

Dazu wird jede Stelle der Hexadezimalzahl einzeln betrachtet. Also zuerst die 3 und dann das B.

$(3)_{16} = (3)_{10} = (00LL)_2$
$(B)_{16} = (11)_{10} = (L0L0)_2$

Insgesamt ergibt sich

$(3B)_{16} = (00LL\ L0L0)_2$

Man kann Hexadezimalzahlen also **STELLENWEISE** in Dualzahlen umwandeln.

Tabelle der Umwandlung Hexadezimal in Dual:

Hexadez.	Dual	Dezimal	Hexadez.	Dual	Dezimal
0	0000	0	8	1000	8
1	0001	1	9	1001	9
2	0010	2	A	1010	10
3	0011	3	B	1011	11
4	0100	4	C	1100	12
5	0101	5	D	1101	13
6	0110	6	E	1110	14
7	0111	7	F	1111	15

BEISPIEL 2: $(BC)_{16} = (L0LL \mid LL00)_2$

Man kann sich also den Umweg über eine Dezimalzahl ersparen.

Umwandlung einer Dualzahl in eine Hexadezimalzahl

Auch für die Umwandlung einer Dualzahl in eine Hexadezimalzahl kann man die Tabelle entsprechend verwenden. Man teilt dazu die gegebene Dualzahl zunächst von rechts ausgehend in entsprechende Viererblöcke ein (ggf. sind noch Nullen zu ergänzen) und wandelt diese Blöcke mit Hilfe der Tabelle in Hexadezimalzahlen um:

```
Beispiel 1:   LL00LL00   =  LL00 | LL00  =  CC
Beispiel 2:   L000LLLL   =  L000 | LLLL  =  8F
Beispiel 3:   LL0L0L     =  00LL | 0L0L  =  35
```

4.5.2. Kodierung ganzer Zahlen

Ganze Zahlen mit n >= 0 (Vorzeichenlose ganze Zahlen)

Ganze Zahlen nehmen je nach Typ verschiedene große Speicherbereiche ein. Dementsprechend unterscheiden sich die Definitionsbereiche der Zahlen. Die Speicherbereiche entsprechen dabei immer einer vollen Bytezahl (1Byte, 2Byte, ...).

Variablentyp	Speicherplatz	Definitionsbereich
Byte	8 Bit = 1 Byte	0..255
Word	16 Bit = 2 Byte	0..65535
Cardinal oder Longword	32 Bit = 4 Byte	0..4294967295

Die Kodierung erfolgt dabei über die entsprechende Umwandlung der Zahlen in eine Dualzahl. Evtl. fehlende Stellen werden dabei durch 0 ergänzt.

BEISPIEL:

Wie wird die Zahl 105 als Byte-Typ gespeichert?

Dazu wandeln Sie zunächst die Zahl in eine Dualzahl um:

$(105)_{10} = (LL0L00L)_2$

Da diese Zahl erst 7Bit umfasst müssen Sie an der 1. Stelle eine 0 ergänzen. Die Kodierung der Zahl sieht daher folgendermaßen aus:

0 L L 0 L 0 0 L

Entsprechend mehr Nullen müssten Sie ergänzen, wenn diese Zahl in den beiden anderen Typen dargestellt werden sollen.

Ganze Zahlen (Vorzeichenbehaftete ganze Zahlen)

Variablentyp	Speicherplatz	Definitionsbereich
Shortint	8 Bit = 1 Byte	-128..127
Smallint	16 Bit = 2 Byte	-32768..32767
Integer oder Longint	32 Bit = 4 Byte	-2147483648..2147483647
Int64	64 Bit = 8 Byte	-2^63..2^63-1

Die Kodierung muss bei diesen Zeichen mit gewährleisten, dass die negativen Zahlen erkannt werden. Wie soll dies geschehen? Schließlich kann der Computer ein Minus Zeichen nicht erkennen. Die in diesen Bereichen auftretenden positiven ganzen Zahlen werden dabei wieder wie gewohnt kodiert. Dabei ergibt sich, dass das erste Bit immer den Wert 0 annimmt. Wäre es daher möglich, für negative Zahlen einfach dieses erste Bit auf 1 zu ändern und den Rest der Zahl "normal" darzustellen? Probieren wir es an einem einfachen Beispiel aus.

Wir kodieren auf diese Weise die beiden Zahlen 8 und -10 als Shortint.

$(8)_{10} = (0\ 0\ 0\ 0\ L\ 0\ 0\ 0)_2$
$(-10)_{10} = (L\ 0\ 0\ 0\ L\ 0\ L\ 0)_2$

Für die Addition dieser beiden Zahlen ergibt sich:

8 + (-10) = -2
0 0 0 0 L 0 0 0 + L 0 0 0 L 0 L 0 = L 0 0 L 0 0 L 0 (-14)

Wir sehen, dass diese Variante mit einem "Vorzeichenbit" zu Fehlern bei Berechnungen führt. Ein anderes Problem wäre die Darstellung der Null. Es gäbe dann zwei Möglichkeiten. Dies ist aber nicht mehr eindeutig.

Verwendete Methode

Die verwendete Methode zur Darstellung negativer Zahlen ist zwar nicht so ganz einfach, aber beide Probleme lassen sich damit beheben.
Die Darstellung der negativen Zahl -10 ergibt sich dabei wie folgt:

0 0 0 0 1 0 1 0	Es wird zunächst die positive Zahl +10 kodiert.
1 1 1 1 0 1 0 1	Diese Zahl wird invertiert.
+ 1	dazu wird 1 addiert.
1 1 1 1 0 1 1 0	Das Ergebnis ergibt die negative Zahl -10.

Die beiden Probleme der anderen Methode sind hiermit behoben. Probieren Sie es aus.

4.5.3. Kodierung von Real-Zahlen

Bei der Kodierung von Real-Zahlen ergibt sich zusätzlich ein Problem zur Kodierung der Nachkommastellen. Die Zahlenbereiche im Einzelnen:

Wie aus der Tabelle zu ersehen ist, spielt im Bereich der Real-Zahlen neben dem Definitionsbereich die Genauigkeit der Zahlen eine wesentliche Rolle.

Variablentyp	Speicherplatz	Definitionsbereich	Genauigkeit
Real48	6 Byte	2.9 x 10^-39 ... 1.7 x 10^38	11-12
Single	4 Byte	1.5 x 10^-45 ... 3.4 x 10^38	7-8
Real oder Double	8 Byte	5.0 x 10^-324 ... 1.7 x 10^308	15-16
Extended	10 Byte	3.6 x 10^-4951 ... 1.1 x 10^4932	19-20
Comp	8 Byte	-2^63+1 ... 2^63 -1	19-20
Currency	8 Byte	-922337203685477.5808 ... 922337203685477.5807	19-20

Eine 4 Byte Single Zahl könnte (hexadezimal) folgendermaßen kodiert sein:

4 2 8 C E 9 F C

Daraus ergibt sich der folgende Binärcode:

0100 0010 1000 1100 1110 1001 1111 1100

Welche rationale Zahl steckt dahinter?

Dazu stellen wir den Binärcode zunächst etwas "anders" zusammen.

0 | 10000101 | 00011001110100111111100

Dabei ist das erste Zeichen für das Vorzeichen der Zahl verantwortlich(hier werden Vorzeichenbit verwendet). Die nächsten 8 Bit repräsentieren einen Exponenten und die letzten 23 Bit eine Mantisse.

Berechnung:

aus dem Vorzeichenbit 0 ergibt sich eine positive rationale Zahl.

Der Exponent ist eine positive ganze Zahl und wird daher wie gewohnt berechnet:

Exp = 1 * 2^8 + 1 * 2^2 + 1 * 2^0 - 127 = 133 - 127 = 6

Die Berechnung der Mantisse erfolgt über negative Exponenten der Basis 2.

Man = 2^-4 + 2^-5 + 2^-8 + 2^-9 + 2^-10 + 2^-12 + 2^-15 + 2^-16 + ... + 2^-21 +1
= 1,248400688...

Die Zahl ergibt sich dann aus: 1,248400688 * 2^6 = 79,897644...

Auf die Kodierung einer rationalen Zahl wollen wir hier verzichten.

4.5.4. Kodierung von Zeichen

Um Daten zwischen zwei Computern mit unterschiedlichen Anwendungsprogrammen und Betriebssystemen austauschen zu können, ist es notwendig neben der Kodierung der Zahlen die Kodierung der Zeichen international festzulegen. Dies ergibt sich schon allein durch die Nutzung des Internets. Ein solches universelles Kodierungssystem für Texte ist der ASCII-Code. Für Grafiken und andere Dateiformate gibt es darüber hinaus noch weitere Kodierungsmöglichkeiten. Wir wollen uns hier etwas näher mit der ASCII-Kodierung beschäftigen:

ASCII = American Standard Code for Information Interchange

Mit Hilfe des ASCII-Codes wird jedem Zeichen eine eindeutige Bitfolge zugeordnet. Man muss dabei aber beachten, dass es für einige Zeichen eine spezifische Ländereinstellung gibt. Eine Tabelle mit den Zeichen der ASCII-Kodierung mit der deutschen Ländereinstellung finden Sie in Ihrem Tafelwerk. Mit Hilfe dieser Tabelle kann man die entsprechende binäre Verschlüsselung eines Zeichens ermitteln.

BEISPIEL:

In welcher Art ist das Wort Informatik kodiert?
Wir entnehmen dazu die Kodierungen der einzelnen Buchstaben aus der Tabelle:

I = 0L00 L00L
n = 0LL0 LLL0
f = 0LL0 0LL0
o = 0LL0 LLLL
r = 0LLL 00L0
m = 0LL0 LL0L
a = 0LL0 000L
t = 0LLL 0L00
i = 0LL0 L00L
k = 0LL0 L0LL

Dies ergibt die folgende Bitfolge (natürlich ohne die Freizeichen dazwischen):

0L00 L00L 0LL0 LLL0 0LL0 0LL0 0LL0 LLLL 0LLL 00L0 0LL0 LL0L 0LL0 000L 0LLL 0L00 0LL0 L00L 0LL0 L0LL

Diese Bitfolge ergibt für den Rechner aber nur in Zusammenhang mit einem Anwendungsprogramm bzw. einer entsprechenden Variablendeklaration einen Sinn. Es könnte sich letztendlich auch um Zahlen oder um eine Folge innerhalb einer Grafik handeln. In den Fällen wird die Bitfolge auch anders interpretiert.
Moderne Programmiersprachen bieten auch die Möglichkeit Zeichen und Zeichenketten in die entsprechende ASCII-Kodierung umzuwandeln und umgekehrt.

Umwandlung ASCII-Kodierung in String

Um eine ASCII-Kodierung in einen String umzuwandeln benutzt man die Standardfunktion `chr`. Diese Funktion wandelt einen Integer-Wert (Typ Byte verwenden) in das entsprechende Zeichen um.

Programmieraufgabe 15

Die Kodierung von Zeichen werden Sie im folgenden Programm ausprobieren. Sie benötigen zwei Button- und zwei Edit-Komponenten. Mit der ersten Prozedur wandeln Sie eine Integer-Zahl in das entsprechende Zeichen um.

```
procedure TForm1.Button1Click(Sender: TObject);
var z : byte;
begin
  z := strtoint (edit1.text);
  edit2.Text := chr (z);
end;
```

Übernehmen Sie diesen kurzen Programmtext in eine Delphi-Anwendung. Geben Sie dann in Edit1 nacheinander einige positive ganze Zahlen ein und vergleichen Sie die Ausgabe mit der ASCII-Kodierung in Ihrem Tafelwerk.

Mit der zweiten Prozedur wandeln Sie ein Zeichen in den entsprechenden Integer-Wert der ASCII-Kodierung um.

```
procedure TForm1.Button2Click(Sender: TObject);
var z     : char;
    text  : string;
    ASCII : integer;
begin
  text := edit2.text;
  z := text [1];
  ASCII := ord (z);
  edit1.Text := inttostr(ASCII);
end;
```

Ergänzen Sie Ihr Programm entsprechend. Testen Sie dieses wieder mit Ihrer ASCII-Tabelle im Tafelwerk.

PROGRAMME / 016 / ASCII_Zeichen.DPR

Fragen und Aufgaben
4. Variablen

Grundlagen

1. Worin unterscheiden sich Variablen für Integer- und Real-Zahlen.
2. Welche der folgenden Zuweisungen führen nicht zu einer Fehlermeldung? Begründen Sie Ihre Entscheidung.
    a. Einer Integer-Variablen wird ein Real-Wert zugeordnet.
    b. Einer Real-Variablen wird ein Integer-Wert zugeordnet.
    c. Einer String-Variablen wird ein Real-Wert zugeordnet.
    d. Einer String-Variablen wird mit Hilfe von INTTOSTR eine Integer-Variable zugeordnet.
    e. Einer String-Variablen wird mit Hilfe von INTTOSTR eine Real-Variable zugeordnet.
    f. Einer Integer-Variablen wird mit Hilfe von STRTOFLOAT eine String-Variable zugeordnet.
3. Worin unterscheiden sich Textvariablen, die als String , String [3] und Char deklariert wurden?
4. Zur Ein- und Ausgabe von Datums- und Zeitwerten wird der Variablentyp TDATETIME verwendet. Informieren Sie sich dazu in der Delphi-Hilfe.
5. Informieren Sie sich über den Variablentyp TColor.
6. Wandeln Sie die nachfolgenden Dualzahlen in eine Dezimalzahl um:
    a. LL00LL
    b. LLLL
    c. LLL000LLL
    d. LLLLL
    e. L0L0L0L0
    f. LL00LL00LL
7. Wandeln Sie die nachfolgenden Dezimalzahlen in eine Dualzahl um:
    a. 13
    b. 24
    c. 48
    d. 77
    e. 121
    f. 308

    Nutzen Sie dazu das Zerlegungsverfahren und das Divisionsverfahren.
8. Berechnen Sie die folgenden Summen von Dualzahlen. Überprüfen Sie die Ergebnisse, indem Sie alle Zahlen anschließend in Dezimalzahlen umwandeln.
    a. LL00LL + LLL
    b. L0L0L0 + LL00LL
    c. LL0L0LL + L00LLL
    d. LLL00LL + L0L0LL0
    e. LL0L + LLL + LLLL
    f. L00L + L0L0 + LL00
    g. L0L0L0 + LLLLL +L000L + L000L
    h. L0L0L0 + L0LL + LLL000 + LL0LL
    i. LLLLLLL + LLLLLLL + LLLLLLL + LLLLLLL
9. Geben Sie die nachfolgenden Dezimalzahlen als Hexadezimalzahlen und anschließend als Dualzahlen an:
    a. 200
    b. 150
    c. 364
    d. 1 234
    e. 6 531
    f. 13 168
10. Wandeln Sie die Dualzahlen aus Aufgabe 6 und 7 in Hexadezimalzahlen um.

11. Kodieren Sie die nachfolgenden Zahlen in 8 Bit Darstellung mit einem vorzeichenbehafteten Typ:
    a. 24
    b. -36
    c. 105
    d. -100
    e. 0
    f. -2
12. Welchen Wert haben die nachfolgenden Hexadezimal kodierten Real-Zahlen?
    a. 10531A0B
    b. 41966FF0
13. Erstellen Sie eine Bitfolge für die folgenden Wörter im ASCII-Code.
    a. Computer
    b. ASCII-Code
    c. Dualzahl
    d. Hexadezimalzahl
14. Die folgenden Bitfolgen repräsentieren jeweils ein Wort im ASCII-Code. Ermitteln Sie das entsprechende Wort.
    a. 0L0L 00LL 0LL0 00LL 0LL0 L000 0LLL 0L0L 0LL0 LL00 0LL0 0L0L
    b. 0L00 0LL0 0LL0 0L0L 0LLL 00L0 0LL0 L00L 0LL0 0L0L 0LL0 LLL0
    c. 0L0L 0LLL 0LL0 L00L 0LL0 LLL0 0LL0 0L00 0LL0 LLLL 0LLL 0LLL 0LLL 00LL
15. Erklären Sie die Unterschiede zwischen dem Zeichen "1", der Zahl 1 als vorzeichenlose ganze Zahl und der Zahl 1 als rationale Zahl vom Typ "Single".

Programmieraufgaben

16. Erstellen Sie Delphi-Programme mit denen im Bereich der Real-Zahlen die nachfolgenden Terme für beliebige Zahlen a, b und c berechnet werden können.
    a. $2x\ (a + 1)$
    b. $(y^2 + z^2)\ (1 - x)$
    c. $x^3 + 0{,}2x^2 - 1{,}5x + 8{,}2$
    d. $(b - 0{,}7) / [(c - e)\ (b2 + 2)]$

17. In den folgenden kurzen Programmauszügen sind Fehler enthalten. Korrigieren Sie diese.

a)
```
procedure ...
var a, b = integer;
...
```

b)
```
procedure ...
var a, b : integer;
...
begin
  a := 2.1;
  b := 5;
  ...
```

c)
```
procedure ...
var  sx, sy : real;
...
begin
  sx := edit1.text;
  sy := 2sx;
  ...
```

d)
```
procedure ...
var a, b : integer;
...
begin
  a := 2;
  b := 5;
  b := a / b;
  ...
```

18. Erstellen Sie Delphi-Programme mit denen im Bereich der Integer-Zahlen die nachfolgenden Terme für beliebige Zahlen a, b und c berechnet werden können.
    a. a * (b + c)
    b. (a + b)*2 + (b + c)*2
    c. 3a + 4b + 5c
    d. (a + b) / (b + c)
    e. $a^3 + b^2 + c$
    f. a - b / c

19. Lösungen/004/004/Project1.dpr
    Es soll ein Programm für einen Schüler der Grundschule implementiert werden, welches die Division natürlicher Zahlen mit Rest realisiert. Für die Eingabe der Daten sind Edit-Komponenten zu verwenden, die Ausgabe soll in einem Label erfolgen.

20. Erstellen Sie ein Delphi-Programm, mit dem die vier Grundrechenoperationen ausgeführt werden können. Die Auswahl der Operation soll über eine RadioGroup erfolgen. Die Ein- und Ausgabe der Daten über Edit-Komponenten.

21. Lösungen/004/005/Project1.dpr
    Für den Physikunterricht sollen Sie ein Programm erstellen, welches die Umrechnung von km/h in m/s und umgekehrt realisiert. Achten Sie auf entsprechende Beschriftungen der Oberfläche.

22. Lösungen/004/006/Project1.dpr
    Erstellen Sie ein Delphi-Programm, welches den Flächeninhalt und den Umfang eines Rechteckes berechnet. Vom Nutzer sind die Längen der beiden Seiten einzugeben.

23. Erstellen Sie ein Delphi-Programm, welches bei vorgegebenem Kreisradius dessen Umfang und Flächeninhalt berechnet.

24. Lösungen/004/007/Project1.dpr
    Erstellen Sie ein Delphi-Programm, welches die Flächeninhalte von Rechteck, Quadrat, Dreieck und Kreis berechnet. Die Auswahl der Figur soll über eine ComboBox erfolgen. Nutzen Sie für die Eingabe nur 2 Edit-Komponenten.

25. Lösungen/004/008/Project1.dpr
    Erstellen Sie ein Delphi-Programm, welches die Längenänderung eines Körpers bei Temperaturänderung berechnet. Es sollen wenigstens 5 Stoffe vorgegeben werden, deren Auswahl über eine RadioGroup erfolgt. Die Auswahl der Temperaturänderung soll über eine ScrollBar im Bereich von 0 bis 200K vorgenommen werden. Die Ausgangslänge ist in eine Edit-Komponente einzugeben.

5. Standardfunktionen und Standardprozeduren

5.1. Begriff

Stellen Sie sich einmal vor, Ihr Taschenrechner kann nur +, -, * und / rechnen. Ihm fehlen aber z.B. die Wurzeltaste, die Tasten für die trigonometrischen Funktionen u.s.w. Jetzt sollen Sie so etwas aber auch noch berechnen. Pech gehabt, oder?
Ähnlich verhält es sich auch mit dem Programmieren. Nun gibt es zwar keine Taste für die Wurzel, aber man kann diese mit ganz bestimmten Funktionen, die bereits fest installiert sind, aufrufen. Solche Funktionen und Prozeduren sind also standardmäßig vorhanden und für die verschiedensten Problemstellungen nutzbar.
Es ist an dieser Stelle nicht sinnvoll alle diese Funktionen und Prozeduren im Einzelnen zu behandeln (zum einen gibt es viel zu viele und zum anderen sind viele davon für so spezielle Aufgabenbereiche vorgesehen, dass wir deren Sinn gar nicht verstehen würden.) Wer sich alle diese Funktionen und Prozeduren anschauen möchte muss einfach mal in der Hilfe nachschauen.
Im Folgenden einige der wichtigsten Funktionen und Prozeduren.

5.2. Arithmetische Funktionen und Prozeduren

5.2.1. ABS

Inhalt:	berechnet den absoluten Betrag einer Zahl		
Syntax	`abs (x)`	x :	Integer oder Real
		Ergebnis:	Integer oder Real

BEISPIELE: abs (-3) = 3
abs (2.4) = 2,4

5.2.2. EXP

Inhalt:	Berechnet e^x e = 2,71828...		
Syntax	`exp (x)`	x :	Real
		Ergebnis:	Real

BEISPIELE: exp (2.1) = 8,16616...

5.2.3. INTPOWER

Inhalt:	Berechnet x^y		
Syntax	`intpower (x, y)`	x :	Real
		y :	Integer
		Ergebnis:	Real

BEISPIELE: intpower (2.1, 3) = 9,261

5.2.4. LN

Inhalt:	Berechnet ln x Basis e = 2,71828...		
Syntax	`ln (x)`	x :	Real
		Ergebnis:	Real

BEISPIELE: ln (3.5) = 1,25276...

5.2.5. LOG10

Inhalt:	Berechnet $\log_{10}x$ Basis 10		
Syntax	`log10 (x)`	x :	Real
		Ergebnis:	Real

BEISPIELE: log10 (2.5) = 0,3979...

HINWEIS: In die uses-Liste muss zusätzlich die Unit "MATH" aufgenommen werden.

5.2.6. LOG2

Inhalt:	Berechnet $\log_2x$ Basis 2		
Syntax	`log2 (x)`	x :	Real
		Ergebnis:	Real

BEISPIELE: log2 (2.5) = 1,3219...

HINWEIS: In die uses-Liste muss zusätzlich die Unit "MATH" aufgenommen werden.

5.2.7. MAX, MIN

Inhalt:	Gibt den größeren (kleineren) von zwei Werten zurück		
Syntax	`max (x, y)` `min (x, y)`	x, y :	Integer oder Real
		Ergebnis:	Integer oder Real

BEISPIELE: max (3 , 4) = 4
min (3 , 4) = 3

HINWEIS: In die uses-Liste muss zusätzlich die Unit "MATH" aufgenommen werden.

5.2.8. PI

Inhalt:	Gibt den Wert für π zurück.		
Syntax	`pi`	Ergebnis:	Real

BEISPIELE: u := 2*PI * 5 = 31,415...
(Kreisumfang mit r = 5)

5.2.9. POWER

Inhalt:	Berechnet x^y		
Syntax	`power (x, y)`	x :	Real
		y :	Real
		Ergebnis:	Real

BEISPIELE: power (2.1, 3.2) = 10,742

HINWEIS: In die uses-Liste muss zusätzlich die Unit "MATH" aufgenommen werden.

5.2.10. ROUND

Inhalt:	Rundet einen Real-Wert auf einen Integer-Wert.		
Syntax	`round (x)`	x :	Real
		Ergebnis:	Integer

BEISPIELE: Round (2.36) = 2

5.2.11. ROUNDTO

Inhalt:	Rundet eine Zahl auf eine beliebige Zehnerpotenz.		
Syntax	`roundTo (x, y)`	x :	Real oder Integer
		y :	Integer
		Ergebnis:	Real oder Integer

BEISPIELE: roundTo (85.635, -1) = 85,6
roundTo (107.25, 1) = 110

HINWEIS: In die uses-Liste muss zusätzlich die Unit "MATH" aufgenommen werden.

5.2.12. SQR

Inhalt:	Bildet das Quadrat einer Zahl (x^2)		
Syntax	`sqr (x)`	x :	Real oder Integer
		Ergebnis:	Real oder Integer

BEISPIELE: sqr (1.5) = 2,25

5.2.13. SQRT

Inhalt:	Berechnet die Quadratwurzel.		
Syntax	`sqrt (x)`	x :	Real
		Ergebnis:	Real

BEISPIELE: sqrt (6.25) = 2,5

5.2.14. TRUNC

Inhalt:	Wandelt eine Real-Zahl in eine Integer-Zahl um.		
Syntax	`trunc (x)`	x :	Real
		Ergebnis:	Integer

BEISPIELE: trunc (3.6) = 3

Das Programmbeispiel stellt die Wirkungsweise der oben besprochenen arithmetischen Funktionen dar.

PROGRAMME / 017 / Arithmetische_Funktionen.PPR

5.3. Datumsfunktionen

5.3.1. DATE

Inhalt:	Gibt das aktuelle Datum zurück (Computereinstellung)		
Syntax	`date`		

5.3.2. DATETIMETOSTR

Inhalt:	Wandelt einen Datumswert in einen String um.		
Syntax	`Datetimetostr (x)`	x :	TDateTime
		Ergebnis:	String

5.3.3. STRTODATETIME

Inhalt:	Wandelt einen String in einen Datumswert um		
Syntax	`Strtodatetime (x)`	x :	String
		Ergebnis:	TDateTime

5.3.4. TIME

Inhalt:	Gibt die aktuelle Uhrzeit zurück.		
Syntax	`Time`	Ergebnis:	TDateTime

Das folgende Programmbeispiel gibt die aktuelle Uhrzeit und das aktuelle Datum aus.

PROGRAMME / 018 / Datumsfunktionen.DPR

5.4. Dialogfelder

5.4.1. Inputbox

Abb. 62: Inputbox

Über eine Inputbox können zur Laufzeit Daten vom Nutzer abgefragt werden. Dieser abgefragte Wert wird dann einer Variablen zugeordnet bzw. steht sofort für Berechnungen zur Verfügung.

Der Quelltext für die oben abgebildete Inputbox lautet:

```
a := inputbox ('Geben Sie einen Wert ein','Hier eingeben','xxx');
```

Soll kein Wert vorgegeben werden, wird statt 'xxx' nur ' ' geschrieben.

5.4.2. Messagebox

Abb. 63: Messagebox

Über eine MessageBox können zur Laufzeit Meldungen an den Nutzer ausgegeben und damit das Programm zu unterschiedlichen weiteren Verläufen veranlasst werden. Dabei stehen dem Programmierer unterschiedliche Möglichkeiten für die Schaltflächen zur Verfügung.

Der Quelltext für die oben abgebildete MessageBox lautet:

```
application.MessageBox('Hier kann der Nutzer auswählen', 'Titel der
Messagebox', MB_ABORTRETRYIGNORE);
```

Bezeichnungen der Schaltflächen

Bezeichnung der Schaltfläche	Erläuterung
MB_ABORTRETRYIGNORE	Das Meldungsfenster enthält drei Schaltflächen: Abbruch, Wiederholen und Ignorieren.
MB_OK	Das Meldungsfenster enthält eine Schaltfläche: OK. Dies ist die Voreinstellung.
MB_OKCANCEL	Das Meldungsfenster enthält zwei Schaltflächen: OK und Abbrechen.
MB_RETRYCANCEL	Das Meldungsfenster enthält zwei Schaltflächen: Wiederholen und Abbrechen.
MB_YESNO	Das Meldungsfenster enthält zwei Schaltflächen: Ja und Nein.
MB_YESNOCANCEL	Das Meldungsfenster enthält drei Schaltflächen: Ja, Nein und Abbrechen.

5.4.3. MessageDlg

Abb. 64: MessageDlg

Über eine MessageDlg können zur Laufzeit Meldungen an den Nutzer ausgegeben und damit das Programm zu unterschiedlichen weiteren Verläufen veranlasst werden. Außerdem werden dabei entsprechende Grafiken im Meldungsfenster mit angezeigt. Dem Programmierer stehen unterschiedliche Möglichkeiten für die Schaltflächen und die Grafiken zur Verfügung.
Der Quelltext für die oben abgebildete MessageDlg lautet:

```
MessageDlg('Dies ist eine MessageDLG', mtWarning,[mbYes,mbNo], 1);
```

Für die Grafiken stehen die folgenden Möglichkeiten zur Verfügung:

Bezeichnung der Grafik	Erläuterung
mtWarning	Ein Meldungsfeld mit einem gelben Ausrufezeichen.
mtError	Ein Meldungsfeld mit einem roten Stoppschild.
mtInformation	Ein Meldungsfeld mit einem blauen "i".
mtConfirmation	Ein Meldungsfeld mit einem grünen Fragezeichen.
mtCustom	Ein Meldungsfeld ohne Grafik. Als Titel des Feldes wird der Name der Anwendung verwendet.

Bezeichnungen der Schaltflächen

Bezeichnung der Schaltfläche	Erläuterung
mbYes	Eine Schaltfläche mit dem Text 'Ja'
mbNo	Eine Schaltfläche mit dem Text 'Nein'
mbOK	Eine Schaltfläche mit dem Text 'Ok'.
mbCancel	Eine Schaltfläche mit dem Text 'Abbrechen'.
mbAbort	Eine Schaltfläche mit dem Text 'Abbruch'
mbRetry	Eine Schaltfläche mit dem Text 'Wiederholen'
mbIgnore	Eine Schaltfläche mit dem Text 'Ignorieren'
mbAll	Eine Schaltfläche mit dem Text 'Alle'
mbNoToAll	Eine Schaltfläche mit dem Text 'Alle Nein'
mbYesToAll	Eine Schaltfläche mit dem Text 'Alle Ja'
mbHelp	Eine Schaltfläche mit dem Text 'Hilfe'

5.4.4. ShowMessage

Abb. 65: ShowMessage

Showmessage ist die einfachste Form der Mitteilung an einen Nutzer.
Der Quelltext für die oben abgebildete ShowMessage lautet:

```
Showmessage ('Hier kann ein kurzer Text eingegeben werden');
```

Im Programm werden die eben gezeigten Dialogkomponenten dargestellt.

PROGRAMME / 019 / Dialogfelder.DPR

5.5. Zahlenkonvertierungen

Vergleiche hierzu 4.3. Nutzung der Ergibt-Anweisung für Berechnungen

5.6. Erzeugen von Zufallszahlen

5.6.1. RANDOMIZE

Inhalt:	Initialisiert den Zufallsgenerator. Randomize sollte immer vor dem ersten Aufruf von Random verwendet werden.		
Syntax	`Randomize`		

5.6.2. RANDOM

Inhalt:	Erzeugt eine Zufallszahl im Bereich 0 <= x < 1 (wenn kein n angegeben) oder im Bereich 0 <= x < n (bei Angabe eines n)		
Syntax	`Random (n)`	n :	Integer
		Ergebnis:	Real oder Integer

Im Programmbeispiel werden Zufallszahlen im Bereich der Real-Typen und der Integer-Typen dargestellt.

PROGRAMME / 020 / Zufallszahlen.DPR

5.7. Trigonometrische Funktionen

5.7.1. SIN, COS, TAN

Inhalt:	Berechnet werden die bekannten Winkelfunktionen. Der Eingabewert muss dabei in Bogenmaß angegeben werden.		

Syntax	`sin (x)`
	`cos (x)`
	`tan (x)`
		x :	Real (Wert in Bogenmaß)

		Ergebnis:	Real

5.7.2. ARCSIN, ARCCOS, ARCTAN

Inhalt:	Umkehrfunktionen der Winkelfunktionen. Der Rückgabewert wird dabei in Bogenmaß ausgegeben.		

Syntax	`arcsin (x)`
	`arccos (x)`
	`arctan (x)`
		x :	Real (Wertebereich beachten)

		Ergebnis:	Real (Wert in Bogenmaß)

HINWEIS: In die uses-Liste muss zusätzlich die Unit "MATH" aufgenommen werden.

Im Programm werden die Funktionen sin, cos und tan dargestellt. Die Eingabe der Daten erfolgt in Grad. Daher ist es zunächst notwendig diese Eingabedaten in rad umzuwandeln.

PROGRAMME / 021 / Trigonometrie.DPR

Fragen und Aufgaben
5. Standardfunktionen und -prozeduren

Grundlagen

1. Was verstehen wir unter Standardfunktionen bzw. Standardprozeduren.
2. Sie wollen in einem Programm die Standardfunktion LOG10 benutzen. Worauf müssen Sie dabei achten?
3. Wofür lassen sich Dialogfelder sinnvoll einsetzen?
4. Worauf muss beim Einsatz von trigonometrischen Funktionen geachtet werden?
5. Geben Sie die Terme in der Notation von Delphi an:
    a. Das Ergebnis einer Division zweier Real-Zahlen mit maximal zwei Stellen nach dem Komma.
    b. 3 sinx (x liegt als Gradzahl vor)
    c. Die Länge der Hypotenuse eines rechtwinkligen Dreiecks bei gegebenen Katheten.
    d. Eine zufällige ganze Zahl zwischen -10 und 10.
    e. Die Berechnung einer beliebigen Potenz von 2.

Programmieraufgaben

6. Erstellen Sie ein Delphi-Programm, welches bei Eingabe des Radius eines Kreises dessen Umfang und Flächeninhalt berechnet. Der Wert für ☐ ist dabei über die Standardfunktion aufzurufen.

7. Lösungen/005/009/Project1.dpr
   Erstellen Sie ein Delphi-Programm, welches das Würfeln mit einem idealen Würfel simuliert. Bei jedem Button Klick ist eine neue Zahl anzuzeigen.

8. Erstellen Sie ein Delphi-Programm für die Multiplikation und Division zweier Real-Zahlen. Das Ergebnis soll auf zwei Stellen nach dem Komma gerundet werden.

9. Lösungen/005/010/Project1.dpr
   Erstellen Sie ein Delphi-Programm mit dem ein rechtwinkliges Dreieck berechnet werden kann. Einzugeben sind die Längen der Katheten. Es sind die Innenwinkel und die Hypotenuse zu berechnen.

10. Lösungen/005/011/Project1.dpr
    Erstellen Sie ein Delphi-Programm, mit dem man bei einem bestimmten Anlagebetrag und eingegebenen Zinssatz den Endbetrag nach einer bestimmten (vom Nutzer vorgegebenen) Zeit berechnen kann.

11. Lösungen/005/012/Project1.dpr
    Die Halbwertzeit gibt an, nach welcher Zeit die Hälfte der vorhandenen radioaktiven Kerne eines bestimmten Isotops zerfallen ist. Die Halbwertzeit für C-14 beträgt ca. 5700 Jahre. Erstellen Sie ein Programm, welches bei Angabe des vorhandenen C-14 Gehaltes eines archäologischen Fundes dessen Alter bestimmt.

12. Erstellen Sie ein Programm zum Üben des kleinen Einmaleins. Die Aufgaben sollen über einen Zufallsgenerator erzeugt werden. Durch Klicken auf einen ersten Button soll eine neue Aufgabe generiert werden. Beim Klicken auf einen zweiten Button soll das Ergebnis der gestellten Aufgabe angezeigt werden.

6. Umsetzung der Algorithmen-strukturen mit Delphi 7

6.1. Einführung in die Boolesche Algebra

6.1.1. Vergleichsoperatoren

In der Mathematik ist es uns möglich Zahlen miteinander zu vergleichen. Diese Vergleichsoperatoren finden wir auch in der Programmierung wieder. Es können aber hier nicht nur Zahlen miteinander verglichen werden, sondern z.B. auch Strings (also Texte). Es bedeuten:

Operator	Bedeutung	Beispiel	Ergebnis
=	gleich	z.B. 7 = 8	Rückgabe: false
<>	ungleich	z.B. 7 <> 8	Rückgabe: true
<	kleiner als	z.B. 7 < 8	Rückgabe: true
<=	kleiner gleich	z.B. 7 <= 8	Rückgabe: true
>	größer	z.B. 7 > 8	Rückgabe: false
>=	größer gleich	z.B. 7 >= 8	Rückgabe: false

6.1.2. Einführung in die boolesche Algebra

Und noch ein wenig Theorie.

Manchmal muss man solche Vergleiche auch miteinander verknüpfen. Diese Verknüpfungen erfolgen über logische Ausdrücke. Hierbei müssen die Eingaben in der Regel weitere logische Ausdrücke oder Vergleichsoperatoren darstellen. Prinzipiell ist es auch möglich diese Vergleichsoperatoren auch auf Zahlen und Zeichen anzuwenden. Wir werden dieses ausprobieren.

BEISPIELE:

Man will wissen, ob eine Zahl innerhalb eines bestimmten Bereiches liegt: 7 <= x <= 10.
Dazu muss überprüft werden, ob diese Zahl x >= 7 und gleichzeitig x <= 10 ist. Dies erfolgt dann über die logische Verknüpfung UND (AND).

Wir wollen wissen, ob eine Zahl nicht in einem Bereich liegt: x soll nicht zwischen 7 und 10 liegen. Dazu müssen wir überprüfen, ob diese Zahl x < 7 ist oder x > 10. Dies erfolgt über die logische Verknüpfung ODER (OR).

Da im zweiten Beispiel eine Zahl nicht gleichzeitig kleiner als 7 und größer als 10 sein kann, kann hier auch die logische Verknüpfung ENTWEDER - ODER (XOR) verwendet werden.

Mit diesen Beispielen befinden wir uns bereits mitten in der booleschen Algebra, bei der es sich letztendlich um logische Verknüpfungen handelt. Wann sind solche Verknüpfungen insgesamt wahr (true) bzw. falsch (false)?

NOT

NOT kehrt den Wahrheitswert einer Aussage um.

Eingangswert	Ausgangswert
true	false
false	true

BEISPIEL: NOT (7 < 8)
Da 7 < 8 eine wahre Aussage ist, ist NOT (7<8) eine falsche Aussage.

AND

AND überprüft ob zwei Aussagen gleichzeitig wahr sind.

Der Ausdruck AND ist vergleichbar mit dem folgenden einfachen Stromkreis. Nur wenn beide Schalter geschlossen sind kann die Glühlampe aufleuchten.

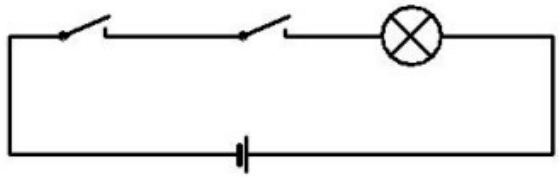


Abb. 66: AND-Schaltung

Eingangswert 1	Eingangswert 2	Ausgangswert
true	true	true
true	false	false
false	true	false
false	false	false

AND gibt also nur dann true zurück, wenn beide Eingangswerte true sind.

OR

OR überprüft, ob von zwei Aussagen mindestens eine wahr ist (es können auch beide wahr sein).

Der Ausdruck OR ist vergleichbar mit dem folgenden einfachen Stromkreis. Wenn mindestens ein Schalter geschlossen ist kann die Glühlampe aufleuchten.

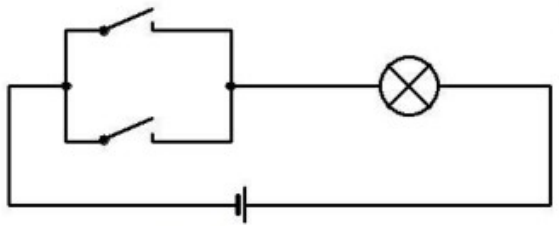


Abb. 67: OR-Schaltung

Eingangswert 1	Eingangswert 2	Ausgangswert
true	true	true
true	false	true
false	true	true
false	false	false

OR gibt also nur dann false zurück, wenn beide Eingangswerte false sind.

XOR

XOR überprüft, ob von zwei Aussagen genau eine wahr ist (es dürfen nicht beide wahr sein).

Eingangswert 1	Eingangswert 2	Ausgangswert
true	true	false
true	false	true
false	true	true
false	false	false

Verknüpfung logischer Ausdrücke

Die oben genannten logischen Ausdrücke können auch miteinander verknüpft werden. An dieser Stelle soll dabei nicht auf die dabei geltenden Vorrangregeln eingegangen werden. Wir einigen uns einfach darauf, was zuerst überprüft werden soll, wird in Klammern gesetzt.

BEISPIELE:

(4 < 8 AND 5 < 9) OR 7 > 10 Ergebnis: True
(4 < 8 XOR 5 < 9) AND 7 > 10 Ergebnis: False

Überprüfen Sie die in den Beispielen getroffenen Aussagen.

6.1.3. Die logischen Ausdrücke in der Programmierung

In der Programmierung hat man sich für diese vergleichenden Ausdrücke nichts Neues einfallen lassen. Die Vergleiche und Ergebnistabellen haben grundsätzlich das gleiche Aussehen. Es ist aber immer auf die richtige Syntax zu achten.

Die beiden Beispiele aus 6.1.2. würden im Programm folgendermaßen aussehen:

(4 < 8 AND 5 < 9) OR 7 < 10	Ergebnis: True
`((4 < 8) AND (5 < 9)) OR (7 > 10)`	

(4 < 8 XOR 5 < 9) AND 7 < 10	Ergebnis: False
`((4 < 8) XOR (5 < 9)) AND (7 > 10)`	

Es sind also nur zusätzlich die zu überprüfenden Eingangswerte in Klammern zu setzen.

Anwendung der logischen Ausdrücke auf Zahlen

Im Programm werden die logischen Ausdrücke auf Ganze Zahlen des Typs Byte angewendet. Die Ergebnisse dabei werden Ihnen seltsam vorkommen.

PROGRAMME / 022 / Logik.DPR

Um zu verstehen, warum die Ergebnisse so seltsam anmuten, vergleichen Sie den Binärcode der Zahlen mit den einzelnen logischen Ausdrücken. Die 0 wird dabei als "FALSE" aufgefasst, die 1 als "TRUE".

6.2. Programmverzweigungen

Sicher haben Sie sich schon gefragt, wozu dieser ganze Aufwand mit den logischen Ausdrücken. In der Programmierung kommt es oft vor, dass abhängig vom Eingangswert ganz bestimmte (unterschiedliche) nachfolgende Wege eingeschlagen werden sollen.

BEISPIEL:

Es soll die Wurzel aus einer Zahl berechnet werden. In Abhängigkeit vom Eingangswert (wenn x < 0) kann dieses Ergebnis aber nicht definiert sein. Um eine entsprechende Fehlermeldung des Compilers abzufangen erfolgt vorab die Überprüfung, ob x >= 0, damit die Berechnung ausgeführt werden kann.

Es soll also in unserem Beispiel das Ergebnis ausgegeben werden, wenn x >= 0 und eine Meldung, wenn der Wert kleiner ist als 0.

Um dies zu erreichen nutzen wir die verschiedenen Formen der Kontrollstrukturen. Verzweigungen werden immer dann eingesetzt, wenn in Abhängigkeit von einem Anfangswert, der eine oder andere Weg eingeschlagen werden soll. Man unterscheidet dabei die einfache und die vollständige Alternative, sowie die Fallauswahl.

vgl. 1.4. Algorithmenstrukturen

6.2.1. Die einfache Alternative

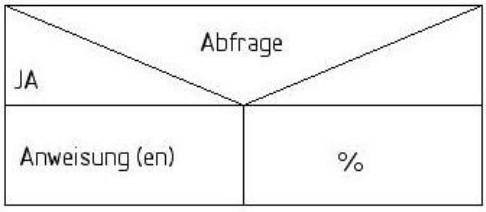


Abb. 68: Struktogrammsymbol für die einfache Alternative

Eine einfache Alternative ist ein Programmkonstrukt, bei dem in Abhängigkeit vom Ausgang einer logischen Abfrage ein bestimmter Programmzweig durchlaufen wird oder nicht. Ergibt die Überprüfung das Ergebnis TRUE, dann werden die nachfolgenden Anweisungen durchlaufen, im Falle der Antwort FALSE werden diese Anweisungen übersprungen.

In der Programmierung mit Delphi wird dafür eine

IF ... THEN - Anweisung benutzt

(ähnlich dem Wenn - Dann in der Tabellenkalkulation).

Die entsprechende Syntax sieht wie folgt aus:

```
IF (logischer Ausdruck) THEN
   Anweisung
```

oder sollen für den TRUE - Fall mehrere Anweisungen folgen

```
IF (logischer Ausdruck) THEN
 begin
   Anweisung 1;
   Anweisung 2;
   ...
 end;
```

Programmieraufgabe 16

Wenden wir uns nun noch einmal dem Wurzelziehen zu. Im Folgenden sollen Sie ein Programm zur Berechnung der Wurzel schreiben. Im Falle einer falschen Eingabe (also einer Zahl die kleiner ist als Null) soll zunächst der Betrag dieser Zahl gebildet werden. Ein verbaler Algorithmus könnte folgendermaßen aussehen: (Später verzichten wir auf verbale Algorithmen ganz und zeichnen gleich die Struktogramme).

1. Lies die Zahl x aus der Eingabekomponente
2. Wenn x kleiner als 0 ist dann bilde den Betrag von x
3. Berechne die Wurzel aus x
4. Gib das Ergebnis aus.

und das entsprechende Struktogramm:

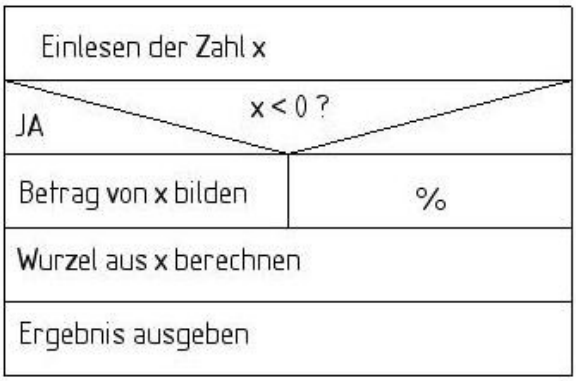


Abb. 69: Struktogramm Wurzel

Zur Bearbeitung des Programms benötigen Sie auf Ihrer (leeren) Oberfläche lediglich einen Button, eine Edit-Komponente zur Eingabe und ein Label zur Ausgabe des Ergebnisses.
Unser konkretes Beispiel sieht dann folgendermaßen aus:

```
{1} procedure TForm1.Button1Click (Sender : TObject);
{2} var x, wurzel : Real;
{3} begin
{4}   x := strtofloat (Edit1.Text);      // liest aus Edit1
{5}   if x < 0 then                      // überprüft ob x < 0
{6}     x := abs (x);                    // im Ja-Fall Betragsbildung
{7}   wurzel := sqrt (x);                // Berechnen der Wurzel
{8}   label1.caption := floattostr (x); // Ausgabe
{9} end;
```

BEISPIEL 2:

Über eine Edit-Komponente soll eine Zahl eingegeben werden. Vom Programm soll geprüft werden, ob diese Zahl zwischen 1 und 100 liegt. Ist dies der Fall, dann soll in einer zweiten Edit-Komponente der String 'JA' ausgegeben werden.

verbaler Algorithmus:

1. Lies die Zahl ein
2. Prüfe, ob die Zahl zwischen 1 und 100 liegt
3. Wenn ja, gib den String 'JA' aus.

AUFGABE: Erstellen Sie zu Beispiel 2 das Struktogramm.

Im Programm würde dies folgendermaßen aussehen:

```
{1} procedure TForm1.Button1Click (Sender : TObject);
{2} var x : Real;
{3} begin
{4}   x := strtofloat (Edit1.Text);    // liest aus Edit1
{5}   if (x <= 100) and (x >= 1) then // überprüft ob x im Bereich
{6}     edit2.text := 'JA';           // im Ja-Fall Ausgabe
{7} end;
```

Im Programm wird die Berechnung einer Wurzel mit einer ggf. vorherigen Betragsbildung dargestellt.

Erstellen Sie das 2. Beispielprogramm selbständig.

PROGRAMME / 023 / Wurzel1.DPR

6.2.2. Die vollständige Alternative

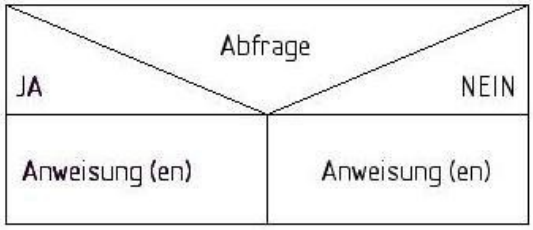


Abb. 70: Struktogrammsymbol für die vollständige Alternative

Eine vollständige Alternative gibt für jede Antwort der geführten Abfrage (also für den TRUE-Fall und für den FALSE-Fall) an, was konkret gemacht werden soll. Jeder Fall erhält also seine eigenen Anweisungen, die bei Eintreten ausgeführt werden sollen. Dies entspricht einer Anweisung der Form WENN Aussage gilt DANN führe dies aus ANSONSTEN führe jenes aus.

Die Syntax dieser vollständigen Alternative sieht wie folgt aus:

```
if ... then
   begin
     ...
   end
 else
   begin
     ...
   end;
```

HINWEISE:

- Vor ELSE darf kein Semikolon stehen.
- Steht nach THEN bzw. nach ELSE nur eine Anweisung so kann BEGIN und END weggelassen werden.
- Achten Sie auf eine konsequente Einrückung der Befehlszeilen, nur so kann die Übersichtlichkeit des gesamten Quelltextes erhalten bleiben.

Programmieraufgabe 17

Kommen wir noch einmal auf das Beispiel Wurzel aus x zurück. In manchen Fällen wollen wir bewusst eine Fehlermeldung ausgeben lassen. Dies soll aber nicht über den Compiler geschehen sondern über eine Ausgabe in der Edit-Komponente oder eine andere Mitteilung an den Nutzer (z.B. ShowMessage). Das Wurzelprogramm aus dem vorigen Abschnitt soll daher ergänzt werden.

Struktogramm des ergänzten Wurzelprogramms:

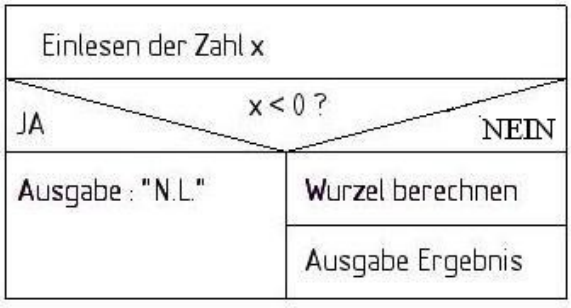


Abb. 71: Struktogramm für Wurzelprogramm II

Öffnen Sie das Programm noch einmal. Weitere Komponenten sind dabei nicht notwendig. Ergänzen Sie dann den Quelltext:

```
{01} procedure TForm1.Button1Click (Sender : TObject);
{02} var x, wurzel : Real;
{03} begin
{04}   x := strtofloat (Edit1.Text);          // liest aus Edit1
{05}   if x < 0 then                          // überprüft ob x < 0
{06}     label1.caption := 'Nicht möglich'    // im Ja-Fall Fehlermitteilung
{07}   else                                   // Ansonsten
{08}     begin
{09}       wurzel := sqrt (x);               // Berechnen der Wurzel
{10}       label1.caption := floattostr (x); // Ausgabe
{11}     end;
{12} end;
```

Ergänzen Sie das Beispiel 2 (Programmieraufgabe 18) so, dass für den Fall die Zahl liegt nicht im Intervall NEIN ausgegeben wird.

PROGRAMME / 024 / Wurzel2.DPR

6.2.3. Verschachtelung der IF-Anweisung

In vielen Fällen müssen diese IF-Anweisungen auch ineinander verschachtelt werden. Das heißt, innerhalb des Anweisungsteils treten neue Abfragen auf. Die Syntax dieser Anweisungen sieht dabei genauso aus, wie Sie es oben kennen gelernt haben. Bei der Verschachtelung von Anweisungen ist darauf zu achten, dass die Anweisungen, die zuletzt mit begin geöffnet wurden zuerst mit end beendet werden.

Ein Beispiel aus der Mathematik ist die Lösung der quadratischen Gleichung $ax^2 + bx + c = 0$. Um die Lösungsformel anwenden zu können, müssen Sie die Gleichung zunächst durch a teilen, dabei ist aber schon auf die Division durch Null zu achten. Die nachfolgende Darstellung zeigt die einzelnen Verschachtelungen farbig an.

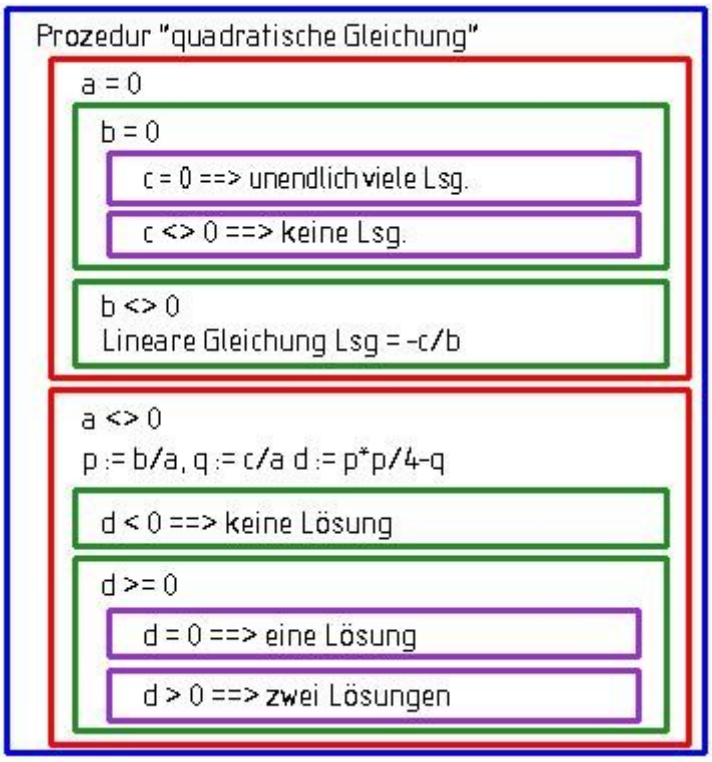


Abb. 72: Verschachtelung von Alternativen

Programmieraufgabe 18

Im Folgenden wollen wir ein Programm zur Lösung dieser quadratischen Gleichung entwickeln. Entwickeln Sie zunächst mit Hilfe der obigen Abbildung ein Struktogramm zur Lösung der Aufgabenstellung.

Das fertige Struktogramm finden Sie im Lösungsteil.

Für die Ausführung des Projekts sind neben der Anfertigung des Struktogramms weitere gründliche Vorüberlegungen notwendig:

Welche Komponenten werden benötigt?
Welche Variablen (Name und Typ) werden benötigt?

Vorschlag für die Komponenten:

Komponente	Name	benötigt für
Button	Button1	Ausführung der Berechnung
Edit	E_Eing_a	Eingabe des Wertes für a
Edit	E_Eing_b	Eingabe des Wertes für b
Edit	E_Eing_c	Eingabe des Wertes für c
Label	L_Lsg1	Ausgabe der 1. Lösung
Label	L_Lsg2	Ausgabe der 2. Lösung
Label	L_Mitteilung	Anzeige der Anzahl der Lösungen

Nutzen Sie weitere Label-Komponenten zur Gestaltung der Oberfläche und deren Beschriftung. Statt der oben angeführten Label-Komponenten lassen sich auch gut Panel-Komponenten für die Ausgabe benutzen.

Die fertige Oberfläche könnte dann in etwa so aussehen:

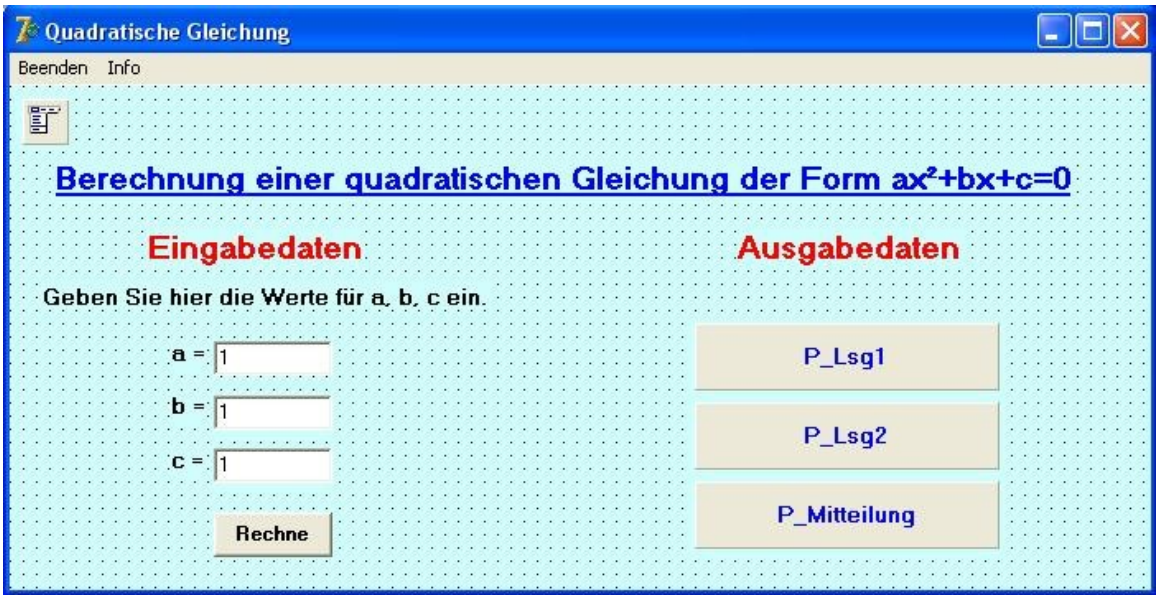


Abb. 73: Mögliche Oberflächengestaltung

Vorschlag für die Variablen:

Name	Typ	benötigt für
a	REAL	Eingabe Wert für a
b	REAL	Eingabe Wert für b
c	REAL	Eingabe Wert für c
p	REAL	Berechnung p in $x^2 + px + q$
q	REAL	Berechnung q in $x^2 + px + q$
D	REAL	Berechnung der Diskriminante
x1	REAL	Erste Lösung
x2	REAL	Zweite Lösung

Wichtige Hinweise

- Schließen Sie jedes begin sofort mit einem end ab und setzen Sie den nachfolgenden Programmtext dazwischen.
- Rücken Sie nachfolgende Verschachtelungen konsequent ein.
- Sparen Sie nicht mit Kommentaren in ihrem Programm.

Und nun zum Programm selbst. Versuchen Sie zunähst das Programm mit Hilfe Ihres Struktogramms selbst zu implementieren.

Den Programmquelltext finden im Lösungsteil.

PROGRAMME / 025 / QuadratischeGleichung.DPR

6.2.4. Die Fallauswahl

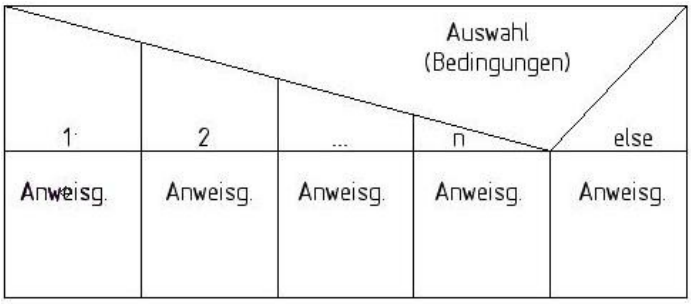


Abb. 74: Struktogrammsymbol für die Fallauswahl

Die Fallauswahl haben wir schon öfter unbewusst eingesetzt. Sie wurde immer dann benutzt, wenn eine RadioGroup oder eine ComboBox verwendet wurde. Je nach Auswahl durch den Nutzer wurden dabei andere Aktionen durch das Programm ausgelöst. Die allgemeine Syntax lautet:

```
case Bedingung of
   Auswertung 1 : Anweisung(en) 1;
   Auswertung 2 : Anweisung(en) 2;
   ...
   Auswertung n : Anweisung(en) n
 else
   Anweisung(en);
 end;
```

HINWEISE:

- Werden für eine Auswertung mehrere Anweisungen notwendig, so sind diese noch einmal in BEGIN und END zu schachteln.
- Für den Fall, dass keine der Auswertungen 1 bis n eintreffen, kann ein entsprechender ELSE-Zweig vorgesehen werden (kein Semikolon vor ELSE).
- Obwohl die CASE - Anweisung nicht mit BEGIN anfängt, muss trotzdem ein abschließendes END geschrieben werden.

Bei der Auswertung einer RadioGroup wird zunächst überprüft, welcher Punkt (ITEMINDEX) angeklickt wurde. Da die Zählung der ITEMs standardmäßig mit 0 beginnt, erfolgen die entsprechenden Auswertungen über diese entsprechende Nummer. Eine ELSE-Anweisung ist hier im Allgemeinen nicht notwendig, da jedem möglichen ITEMINDEX Anweisungen zugeordnet werden.

Die quadratische Gleichung wird hier mit einer Fallauswahl gelöst.

PROGRAMME / 026 / QuadratischeMitCase.DPR

6.3. Zyklen (Programmwiederholungen)

Zyklen werden immer dann eingesetzt, wenn ein bestimmter Programmteil mehrmals durchlaufen werden soll. Es handelt sich dabei um Wiederholungen. Es werden drei Arten von Zyklen unterschieden:

- Zählzyklen
- abweisende Zyklen
- nichtabweisende Zyklen

Zyklen können ineinander verschachtelt werden. Sie können auch Alternativen enthalten.
Bei der Arbeit mit Zyklen ist immer darauf zu achten, dass diese terminierend sind. Das heißt, Zyklen müssen nach einer endlichen Anzahl von Durchläufen vom Programmablauf beendet werden können. Daher ist hier eine besonders genaue Planung der Algorithmen unbedingt notwendig.

vgl. 1.4. Algorithmenstrukturen

6.3.1. Die Zählschleife

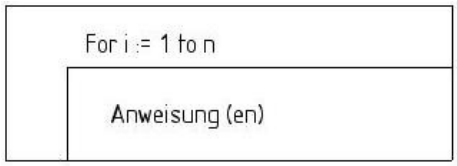


Abb. 75: Struktogrammsymbol für eine Zählschleife

Die allgemeine Syntax einer solchen Zählschleife sieht folgendermaßen aus:

```
for zählvariable := anfang to ende do
  begin
    anweisung 1
    anweisung 2
    ...
  end;
```

HINWEISE:

- Wenn innerhalb der FOR-Schleife nur eine Anweisung zu finden ist, kann BEGIN und END auch entfallen.
- Es ist darauf zu achten, dass die Zählvariable innerhalb der Schleife zwar verwendet werden kann, sie darf aber unter keinen Umständen verändert werden. D.h. man darf dieser Zählvariablen innerhalb der Schleife keine Werte zuweisen.
- In unserem Beispiel haben wir das Aufwärtszählen benutzt. Es besteht aber auch die Möglichkeit zum Abwärtszählen, hierfür muss dann geschrieben werden:

```
for zählvariable := ende downto anfang do
```

For-Schleifen (und auch alle anderen Schleifenformen) können ähnlich wie Alternativen ineinander geschachtelt werden. Auch dabei gilt die Regel:

- Die Schleife, die zuletzt begonnen wurde, wird zuerst beendet. Außerdem ist darauf zu achten, dass jede Schleife eine andere Schleifenvariable benötigt.

Im Folgenden werden die verschiedenen Auswirkungen der Nacheinanderausführung und der Schachtelung zweier FOR-Schleifen demonstriert:

PROGRAMME / 027 / Zaehlschleife.DPR

Programmieraufgabe 19

Problemstellung:

Es soll die Summe der Zahlen von 1 bis 100 gebildet werden. Dazu soll aber nicht die berühmte Gauß'sche Formel eingesetzt werden. Nun könnte man das einfach umsetzen, indem man im Programm schreibt 1 + 2 + 3 + ... + 100. Dies wäre aber zum einen recht mühsam, und zum anderen ist damit nur diese eine Aufgabe lösbar. Schon für die Summe von 1 bis 99 müsste ein neues Programm geschrieben werden. Dies ist aber nicht der Sinn der Programmierung.

Hier bietet sich eine Zählschleife an.
Dabei wird in unserem Beispiel ein bestimmter Programmteil von 1 bis 100 durchlaufen und bei jedem Durchlauf jeweils die Summe mit der aktuellen Zählvariablen gebildet. Ein verbaler Algorithmus könnte wie folgt aussehen:

1. Lies den Endwert der Summierung ein.
2. Setze die Summe auf 0 (Initialisierung einer Variablen)
3. Durchlaufe die folgende Schleife von 1 an bis zu diesem Endwert.
4. Addiere zur aktuellen Summe die Zählvariable und ordne das Ergebnis der Summe zu
5. Gib die aktuelle Summe aus.

Das Struktogramm

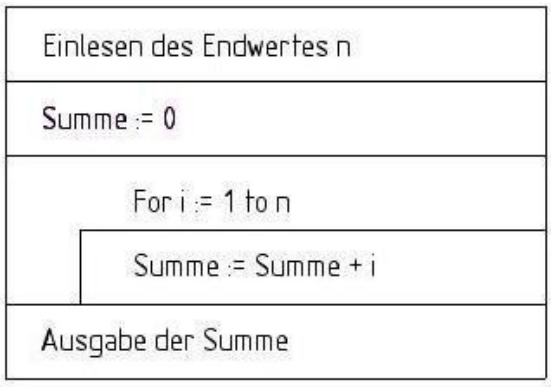


Abb. 76: Struktogramm für die Summierung von n Zahlen

Zur Umsetzung des Programms wird also zusätzlich eine Zählvariable benötigt. Da man "Zählen" nur im Bereich der Ganzen Zahlen sinnvoll ausführen kann, muss diese Zählvariable also vom Typ Integer sein. Natürlich muss auch diese Variable deklariert werden.

Unser Programm würde dann wie folgt aussehen:

```
{ 1} procedure TForm1.Button1Click (Sender : TObject);
{ 2} var i  : integer;                // Zählvariable
{ 3}     n  : integer;
{ 4}     su : integer;                // Summe der Zählung
{ 5} begin
{ 6}   n  := strtoint (edit1.text);   // Endwert der Zählung
{ 7}   su := 0;                       // Initialisierung
{ 8}   for i := 1 to n do
{ 9}     begin
{10}       su := su + i;              // Summenbildung
{11}     end;
{12}   edit2.text := inttostr (su);
{13} end;
```

PROGRAMME / 028 / Summe_1.DPR

6.3.2. Abweisende Zyklen

In einer FOR-Schleife steht die Anzahl der Durchläufe von Anfang an fest. Manchmal kommt es aber auch vor, dass diese Anzahl nicht bekannt ist. Dann kann die FOR-Schleife nicht benutzt werden. Für diese Fälle gibt es zwei andere Möglichkeiten zur Berechnung, die so genannten abweisenden und die nichtabweisenden Zyklen.

Abweisende Zyklen sind Wiederholungen (Iterationen), bei denen die Abbruchbedingung vor Durchlauf der Schleife überprüft wird.

Struktogrammsymbol

Abb. 77: Struktogrammsymbol für abweisende Schleife

Syntax:

```
while Bedingung do
  begin
    Anweisung 1
    Anweisung 2
    ...
  end;
```

HINWEISE:

- Eine WHILE-Schleife wird solange durchlaufen, bis die Abbruchbedingung nicht mehr erfüllt ist.
- Erfolgt nur eine Anweisung innerhalb der Schleife kann auf BEGIN und END verzichtet werden.
- Es ist darauf zu achten, dass die Abbruchanweisung innerhalb des Programms auf jeden Fall nach endlich vielen Schritten erreicht wird. Ansonsten gerät das Programm in eine Endlosschleife. Dieser Programmierfehler ist oft nur sehr schwer zu finden.

Programmieraufgabe 20

Problemstellung:

Es soll solange die Summe der ersten n natürlichen Zahlen gebildet werden, bis die Summe größer ist als 100. Da wir hier nicht wissen, wie viele Zahlen zu summieren sind, kann die FOR-Schleife nicht eingesetzt werden (man könnte es natürlich ausprobieren, aber wir wollen das Problem ja vom Computer lösen lassen).

Struktogramm zur Problemlösung:

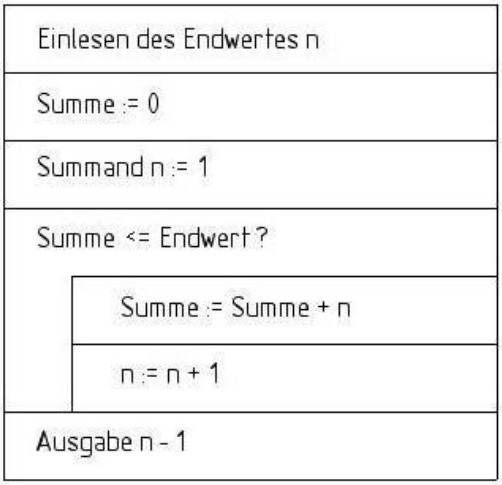


Abb. 78: Struktogramm Summierung bis zu einer Endsumme

Als Programm sieht das ganze folgendermaßen aus:

```
{ 1} procedure TForm1.Button1Click (Sender : TObject);
{ 2} var n     : integer;
{ 3}     su    : integer;
{ 4}     ende  : integer;
{ 5} begin
{ 6}   ende := strtoint (edit1.text);
{ 7}   su   := 0;
{ 8}   n    := 1;
{ 9}   while su < ende do
{10}     begin
{11}       su := su + n;
{12}       n  := n + 1;
{13}     end;
{14}   edit2.text := inttostr (n-1);
{15} end;
```

HINWEISE:

In Zeile 9 erfolgt die Abfrage, ob die Summe kleiner ist als der Endwert. Wird dies mit "JA" beantwortet, dann wird die Schleife durchlaufen, ansonsten erfolgt der Abbruch der Schleife. Erfolgt nur eine Anweisung innerhalb der Schleife kann auf BEGIN und END verzichtet werden. Es ist darauf zu achten, dass die Abbruchanweisung innerhalb des Programms auf jeden Fall nach endlich vielen Schritten erreicht wird. Ansonsten gerät das Programm in eine Endlosschleife. Dieser Programmierfehler ist oft nur sehr schwer zu finden.

PROGRAMME / 029 / Summe_2.DPR

6.3.3. Nichtabweisende Zyklen

Das gleiche Problem lässt sich auch mit einem so genannten nichtabweisenden Zyklus behandeln. Während bei einem abweisenden Zyklus die Bedingung am Anfang steht und dadurch der Zyklus nicht in jedem Fall durchlaufen werden muss, steht bei einem nichtabweisenden Zyklus die Bedingung erst am Ende der Anweisung, der Zyklus wird daher mindestens einmal durchlaufen.

Nichtabweisende Zyklen sind Zyklen, bei denen die Abbruchbedingung erst nach der Schleife überprüft wird.

Struktogrammsymbol

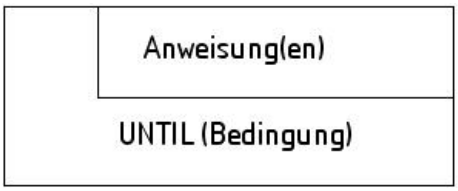


Abb. 79: Struktogrammsymbol für nichtabweisende Schleife

Syntax:

```
repeat
   Anweisung 1
   Anweisung 2
   ...
 until Bedingung;
```

HINWEISE:

- Eine REPEAT...UNTIL-Schleife wird solange durchlaufen bis die Abbruchbedingung erfüllt ist. (Beachten Sie den Unterschied zur WHILE-Schleife, die solange durchlaufen wird bis die Abbruchbedingung nicht mehr durchlaufen wird.)
- Anders als bei der For-Schleife oder bei der while-Schleife werden die Anweisungen hier nicht in BEGIN und END gefasst. Es werden alle Anweisungen zwischen repeat und until wiederholt abgearbeitet.
- Nach until wird die Abbruchbedingung angegeben. Es muss wieder darauf geachtet werden, dass die Schleife nach endlich vielen Schritten beendet wird.

Programmieraufgabe 21

Wir wollen die Problemstellung aus der Programmieraufgabe 20 jetzt mittels eines solchen nichtabweisenden Zyklus lösen.

Struktogramm:

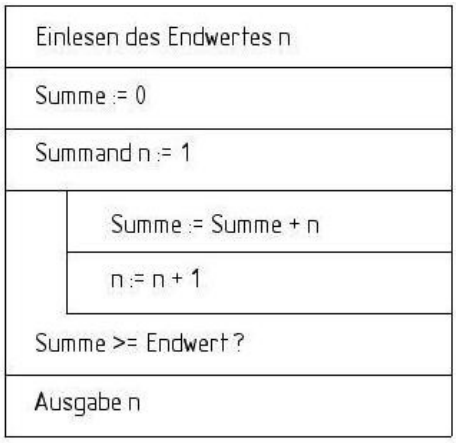


Abb. 80: Struktogramm Summierung bis zu einer Endsumme

Der Programmtext sieht dann folgendermaßen aus:

```
{ 1} procedure TForm1.Button1Click (Sender : TObject);
{ 2} var n    : integer;
{ 3}     su   : integer;
{ 4}     ende : integer;
{ 5} begin
{ 6}   ende := strtoint (edit1.text);
{ 7}   su   := 0;
{ 8}   n    := 1;
{ 9}   repeat
{10}     su := su + n;
{11}     n  := n + 1;
{12}   until su > ende;
{13}   edit2.text := inttostr (n);
{14} end;
```

PROGRAMME / 030 / Summe_3.DPR

6.4. Euklidscher Algorithmus

6.4.1. Mathematische Grundlagen

Mit Hilfe des Euklidschen Algorithmus kann von zwei gegebenen natürlichen Zahlen der größte gemeinsame Teiler (ggT) bestimmt werden. Dies erfolgt dabei nicht über die Primfaktorenzerlegung, die Ihnen aus unteren Klassenstufen bekannt ist, sondern über die Division mit Rest.

Darstellung des Euklidschen Algorithmus

BEISPIEL: Es soll der ggT der Zahlen 124 und 76 bestimmt werden:

Aufgabe	Ergebnis	Rest
124 : 76 =	1	48
76 : 48 =	1	28
48 : 28 =	1	20
28 : 20 =	1	8
20 : 8 =	2	4
8 : 4 =	2	0

Die angegebene Division wird dabei immer solange ausgeführt, bis der Rest 0 ergibt. Der ggT ist dann der letzte von 0 verschiedene Rest.

In unserem Beispiel ist der ggT = 4.

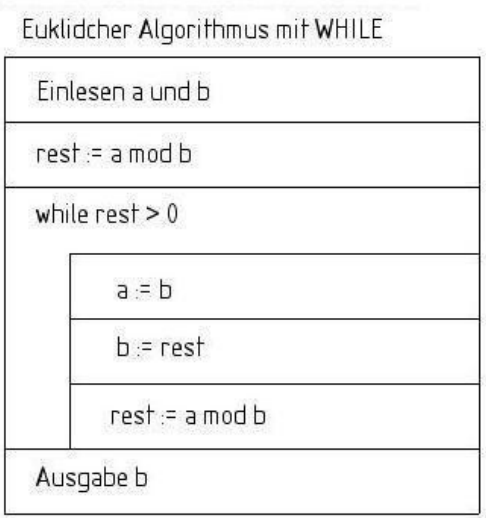
AUFGABE:

Ermitteln Sie auf diese Art und Weise den ggT der Zahlen:

- 24 und 36
- 65 und 91
- 84 und 96
- 144 und 233

6.4.2. Umsetzung des Euklidschen Algorithmus in ein Delphi Programm

Struktogramme



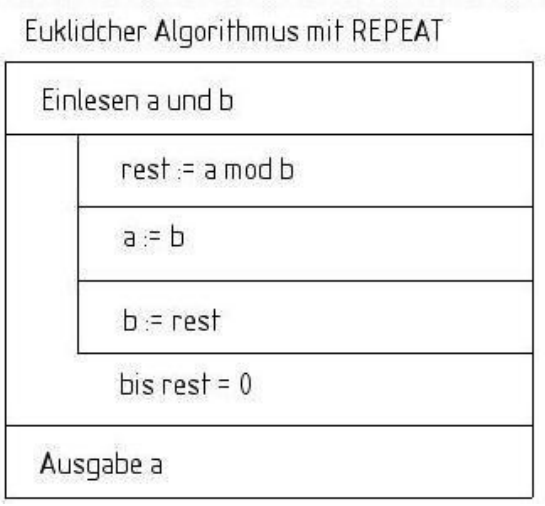


Abb. 81 und 82: Struktogramme für den Euklidschen Algorithmus

Quelltext des Euklidschen Algorithmus mit einer WHILE-Schleife:

```
procedure TForm1.Button1Click(Sender: TObject);
var a, b, rest : integer;
begin
  a    := strtoint (edit1.Text);
  b    := strtoint (edit2.Text);
  rest := a mod b;
  while rest <> 0 do
    begin
      a    := b;
      b    := rest;
      rest := a mod b;
    end;
  edit3.Text := inttostr (b);
end;
```

AUFGABE:

Implementieren Sie den Euklidschen Algorithmus mit einer repeat-Schleife.

Programmieraufgabe 22

Anwendungsbeispiele für die Nutzung des Euklidschen Algorithmus sind:

- Kürzen eines gemeinen Bruches
- Ermitteln des kleinsten gemeinsamen Vielfachen zweier natürlicher Zahlen
- Bestimmung von Verhältnissen (Proportionalitäten)

Im folgenden Projekt sollen mit Hilfe des Euklidschen Algorithmus der größte gemeinsame Teiler und das kleinste gemeinsame Vielfache zweier natürlicher Zahlen bestimmt werden. Sie benötigen hierfür vier Edit-Komponenten und einen Button.
Implementieren Sie zunächst die Berechnung des ggT mit einer REPEAT-Schleife.
Überlegen Sie dann, wie mit Hilfe des bestimmten ggT das kgV berechnet werden kann. Nutzen Sie dazu das Einführungsbeispiel. Ergänzen Sie Ihren Quelltext entsprechend.

Testen Sie Ihr Programm mit einfach nachvollziehbaren Beispielen.

Den vollständigen Quelltext finden Sie im Lösungsteil.

PROGRAMME / 031 / kgV_und_ggT.DPR

Fragen und Aufgaben
6. Umsetzung der Algorithmenstrukturen mit Delphi 7

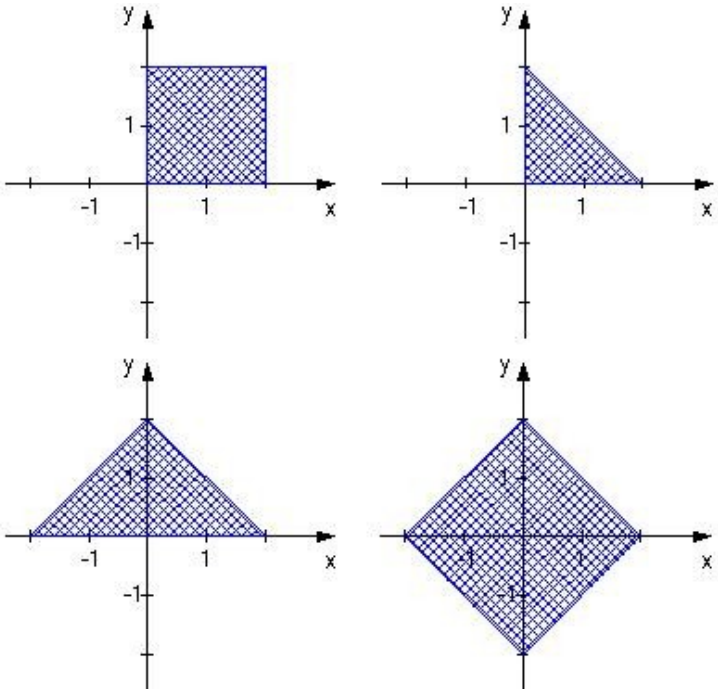
HINWEIS:
Zu allen Programmen ist zunächst ein Struktogramm anzufertigen.

Theoretische Grundlagen

1. Stellen Sie den Wahrheitswert der nachfolgenden Ausdrücke fest:
   a. (5 > 8) OR (5 = 8)
   b. (7 = 14/2) AND (8 *2 < 25)
   c. (9 < 4) XOR (5 < 8)
   d. (NOT (5 < 4)) XOR (4 < 6)
   e. (4 + 7 < 15) AND (5 + 8 < 15) AND (9 + 4 < 15)
   f. ((6 - 8 > 0) OR (8 - 6 > 0)) AND (3*3 < 10)

2. Stellen Sie für die nachfolgenden Problemstellungen einen logischen Ausdruck auf:
   a. Eine Zahl soll zwischen 1 und 50 liegen
   b. Eine Zahl soll nicht zwischen 2 und 5 liegen
   c. Eine Zahl soll kleiner sein als 100 oder zwischen 200 und 300 liegen
   d. Eine Zahl soll entweder zwischen 4 und 10 oder zwischen 25 und 30 liegen
   e. Die Summe zweier Zahlen soll nicht größer sein als 10, ihr Produkt dagegen soll größer sein als 20.

3. Stellen Sie die farbig schraffierten Punktmengen mit Hilfe eines logischen Ausdrucks dar.



4. Welcher Unterschied besteht zwischen einer einfachen und einer vollständigen Alternative?
5. Worin unterscheiden sich FOR und WHILE-Schleife?
6. Worin unterscheiden sich WHILE und REPEAT-Schleife?
7. Geben Sie Beispiele für Alternativen bzw. Zyklen aus Ihrem täglichen Leben an.

8. Geben Sie für die nachfolgenden verschachtelten FOR-Schleifen nacheinander die Werte der Laufvariablen i, j und k an.

a)
```
FOR i := 1 TO 4 DO
  BEGIN
    ...
  END;
FOR j := 1 TO 3 DO
  BEGIN
    ...
  END;
```

b)
```
FOR i := 1 TO 4 DO
  BEGIN
    FOR j := 1 TO 3 DO
      BEGIN
        ...
      END;
    FOR k := 8 DOWNTO 6 DO
      BEGIN
        ...
      END;
  END;
```

c)
```
FOR i := 4 DOWNTO 1 DO
  BEGIN
    FOR j := 1 TO 3 DO
      BEGIN
        FOR k := 8 DOWNTO 6 DO
          BEGIN
            ...
          END;
      END;
  END;
```

d)
```
FOR i := 1 TO 4 DO
  BEGIN
    FOR j := 0 TO i DO
      BEGIN
        ...
      END;
  END;
```

9. Die nachfolgenden Schleifen sind nicht terminierend. Geben Sie den Fehler an.

a)
```
n := 1
while n <> 0 do
  begin
    ...
    n := n + 1;
  end;
```

b)
```
n := 1
while n = 1 do
begin
  su := a + n;
  a := su;
end;
```

10. Berechnen Sie den ggT der folgenden Zahlen mit dem Euklidschen Algorithmus:
    a. 116 und 224
    b. 24 und 226
    c. 48, 72 und 144

Programmieraufgaben

11. Lösungen/006/013/project1.dpr
    Erstellen Sie ein Programm zur Kreisberechnung. Der Nutzer soll bei der Eingabe einer negativen Zahl mit einer MessageBox auf seinen Fehler aufmerksam gemacht werden.

12. Lösungen/006/014/project1.dpr
    Erstellen Sie ein Programm, welches entscheidet, ob aus drei vorgegebenen Seitenlängen ein Dreieck konstruierbar ist (Dreiecksungleichung).

13. Schreiben Sie ein Delphi-Programm, welches zur Auswertung einer Messreihe eingesetzt werden kann. Es sind dabei 10 Werte einzulesen. Das Einlesen der Werte soll über eine InputBox erfolgen. Auszugeben ist der Mittelwert dieser Messreihe.

14. Lösungen/006/015/project1.dpr
    Ändern Sie das Programm aus Aufgabe 13 so ab, dass die Anzahl der Messwerte vom Nutzer über eine Edit-Komponente angegeben werden kann.

15. Für einen eingegebenen Wert soll überprüft werden, ob dieser in einem bestimmten Bereich liegt. Erstellen Sie ein entsprechendes Programm.

16. Schreiben Sie ein Delphi-Programm mit dem der Betrag einer eingegebenen Zahl berechnet werden kann. (keine Standardfunktion verwenden).

17. Erstellen Sie ein Delphi-Programm zur Lösung der quadratischen Gleichung
    $x^2 + px + q = 0$.

18. Lösungen/006/016/project1.dpr
    Erstellen Sie ein Delphi-Programm zur Lösung der allgemeinen quadratischen Gleichung.

19. Simulieren Sie in einem Delphi-Programm das Würfeln mit einem idealen Würfel. Vom Nutzer ist die Anzahl der Würfe einzugeben. Es sind die Anzahlen der gewürfelten Einsen, Zweien... auszugeben.

20. Lösungen/006/017/project1.dpr
    Vom Nutzer sind drei beliebige ganze Zahlen einzugeben. Schreiben Sie ein Delphi-Programm, das diese drei Zahlen der Größe nach ordnet (in aufsteigender Reihenfolge).

21. Lösungen/006/018/project1.dpr
    Entwickeln Sie ein Spiel Zahlenraten. Über den Zufallsgenerator ist dabei eine Zahl zwischen 1 und 100 zu erzeugen. Über eine Edit-Komponente kann eine geratene Zahl eingegeben werden. Ist diese Zahl zu groß oder zu klein soll in einem Label eine entsprechende Ausgabe erfolgen. Bei richtig geratener Zahl soll die Anzahl der Versuche ausgegeben werden.

22. Ein Kopierladen bietet die folgenden Tarife:
    Tarif 1 : 5 Cent für eine einfache Kopie
    Tarif 2 : 10 Cent für Vergrößerungen auf A3
    Tarif 3 : 40 Cent für eine Farbkopie
    Tarif 4 : 80 Cent für die Vergrößerung einer Farbkopie auf A3
    Schreiben Sie ein Programm, das bei eingegebener Stückzahl den Preis berechnet.
23. Lösungen/006/019/project1.dpr
    Für einen vorgegebenen Geldbetrag ist die kleinste mögliche Anzahl an Münzen zu ermitteln.

24. Es ist ein Programm zu erstellen, welches für eine vom Nutzer eingegebene Zahl (INTEGER) überprüft, ob diese Zahl durch 2, 3, 4, ... 10 teilbar ist.

25. Lösungen/006/020/project1.dpr
    Erstellen Sie ein Spiel Würfeln. Es ist dabei solange eine Zufallszahl zwischen 1 und 6 zu generieren, bis die Summe mindestens 21 erreicht. Wird genau 21 erreicht, dann hat der Spieler gewonnen.

26. Lösungen/006/021/project1.dpr
    Erstellen Sie einen Trainer für das kleine Einmaleins. Die Faktoren sind über einen Zufallsgenerator zu ermitteln. Der Nutzer soll die Anzahl der zu übenden Aufgaben selbst vorgeben können. Das Ergebnis soll in eine Edit-Komponente eingegeben werden. Weiterhin soll eine Auswertung der einzelnen Aufgabe und der Übungsreihe erfolgen.

27. Lösungen/006/022/project1.dpr
    Entwickeln Sie mit Hilfe des Euklidschen Algorithmus eine Delphi-Anwendung, zum Rechnen mit gemeinen Brüchen. Es sollen dabei die vier Grundrechenoperationen implementiert werden. Die Ergebnisse sind so weit wie möglich zu kürzen. Das Kürzen ist dabei in einer gesonderten Prozedur durchzuführen.

28. Von zwei gegebenen natürlichen Zahlen ist das kleinste gemeinsame Vielfache zu bestimmen.

7. Prozeduren und Funktionen

7.1. Aufbau einer Delphi-Unit

7.1.1. Die wesentlichen Bestandteile einer Unit

Wird eine neue Delphi-Anwendung erstellt, so wird der dafür notwendige Quellcode sofort zur Verfügung gestellt und die notwendigen Dateien erzeugt. Dazu gehört unter anderem die entsprechende Unit (Unit1.pas).
Zu dieser Unit gehören die folgenden Bestandteile:

- Name der Unit
- der Interface-Teil
- der Typendeklarationsteil
- der Variablendeklarationsteil
- der Implementierungsteil
- das abschließende end.

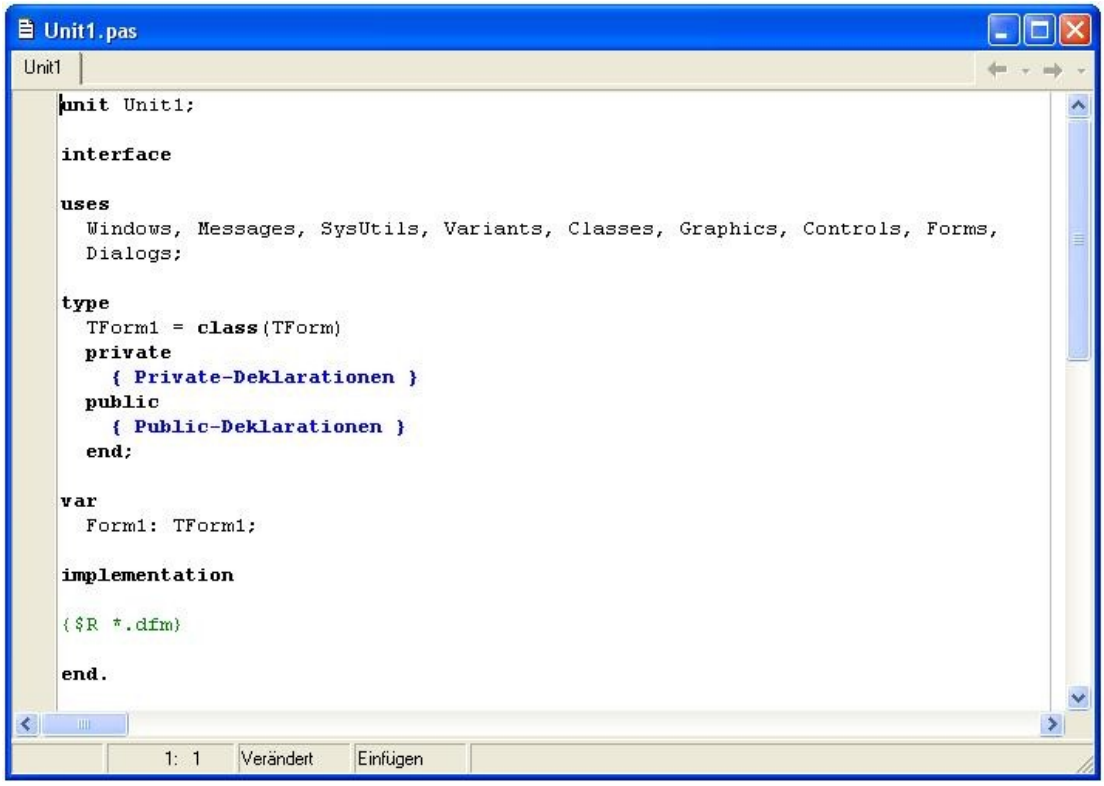


Abb. 84: Ansicht einer Unit

7.1.2. Der Name der Unit

Vom Programm erhalten neu eingebundene Units zunächst Namen der Form UNIT1, UNIT2, ...
Soll eine Unit einen anderen Namen bekommen, so kann man dies nur beim Speichern erreichen.

HINWEIS:

⚠	Versuchen Sie nie den Namen der Unit direkt im Quelltext zu ändern. Dies führt zu ungeahnten Fehlermeldungen.

Der Name einer Unit sollte immer mit U_ ... beginnen.

Dieser Name kann dabei nicht beliebige Zeichen umfassen. Sonderzeichen (Umlaute, ?, ...) sind nicht erlaubt. Auch Leerzeichen im Namen werden vom Compiler übel genommen.

Programmieraufgabe 23

Erstellen Sie zunächst ein neues Delphi-Projekt (entweder durch Starten von Delphi oder durch Datei / Neu / Anwendung).
Wechseln Sie mit F12 in die zugehörige UNIT1.

In der Zeile 1 finden Sie folgenden Quelltext:

```
unit  Unit1
```

Wir wollen dieser Unit1 nun den Namen U_Beispiel geben. Gehen Sie dazu wie folgt vor:

1. Sind Sie noch in der Unit? Ansonsten mit F12 wieder in die Unit wechseln.
2. Datei / Speichern unter
3. Es öffnet sich wieder der Speichern-Dialog.
4. Wechseln Sie in einen (neuen) Ordner.
5. Geben Sie der Datei den Namen U_Beispiel und klicken Sie auf ok.

Wie hat sich die Zeile 1 des Quelltextes verändert?

7.1.3. Schnittstellen Vereinbarungen

(Interface-Teil)
Aussehen beim Starten einer neuen Unit:

```
interface

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

Interface ist dabei das "Stichwort" für das Programm - jetzt kommen die Schnittstellen-Vereinbarungen.

Nach uses folgt eine Liste mit den bereits in das Programm eingefügten anderen Units (Units dürfen daher auf keinen Fall diese Namen erhalten). Aus dieser Liste dürfen Sie keine Daten löschen. Es handelt sich dabei um Standard-Units, die die Funktionalität von Delphi erst ermöglichen.

Es ist aber möglich weitere Units hier einzubinden.

Programmieraufgabe 24

Öffnen Sie dazu die Programmieraufgabe 23. Erstellen Sie dann mit Datei / Neu / Unit eine neue Unit für dieses Programm (dieser neuen Unit ist keinem Formular zugeordnet und sieht daher viel kürzer aus). Speichern Sie die Unit im gleichen Verzeichnis wie das Beispiel unter dem Namen U_Anhang. Damit diese Unit im Programm eingebunden wird, müssen Sie die Unit im Interface-Teil der Unit U_Beispiel mit aufnehmen. Ergänzen Sie dazu die uses-Liste wie folgt:

```
Interface

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

Da in der Unit U_Anhang aber noch kein weiterer Programmtext zu finden ist, hat die Unit für die Programmausführung noch keinerlei Bedeutung. Wir werden solche Units später mit Quelltext versehen.

Ein anderes Beispiel zum Einbinden einer Unit in ein Programm.

Öffnen Sie dazu noch einmal die Programmieraufgabe 23.
Erstellen Sie dann mit Datei / Neu / Formular ein weiteres Formblatt für das Projekt. Speichern Sie die dazugehörige Unit im gleichen Verzeichnis mit dem Namen U_Beispiel2. Fügen Sie diese Unit in die uses-Liste ein.

HINWEIS:

Nicht sichtbare Units (bzw. Formblätter) können mit Ansicht / Units (bzw. Formulare) wieder sichtbar gemacht werden.
Gehen Sie dann wie folgt vor:

1. Legen Sie auf das Formular 1 (Form1) zwei Buttons.
2. Legen Sie auf das Formular 2 (Form2) zwei Buttons.
3. Erstellen Sie zu Button1 in Form1 die Prozedur OnClick, indem Sie doppelt auf diesen Button klicken.
4. Schreiben Sie den nachfolgenden Quelltext zwischen begin und end;

```
begin
  form2.color := clred;
  form2.show;
end;
```

5. Starten Sie das Programm und klicken Sie auf Button1. Was passiert? Erklären Sie.
6. Erzeugen Sie die Prozedur zu Button1 in Form2 die OnClick Prozedur.
7. Ergänzen Sie den Quelltext wie folgt:

```
begin
  form1.color := cllime;
  form1.show;
end;
```

8. Starten Sie das Programm. Sie werden zunächst eine Fehlermeldung erhalten (aufmerksam lesen). Klicken Sie auf ok und starten Sie nochmals. Probieren Sie das Programm aus.

Das Programm hat die Unit U_Beispiel automatisch in die Unit U_Beispiel2 aufgenommen.
Wechseln Sie in die Unit U_Beispiel2 und suchen Sie die Stelle an der dieses passiert ist. (Dies ist eine weitere Möglichkeit Units einzubinden).

HINWEIS:

Eine Einbindung in der uses-Liste im Interface-Teil hätte hier zu einer anderen Fehlermeldung geführt. Damit hätte man einen so genannten Kreuzverweis implementiert und den beantwortet der Compiler mit einer Fehlermeldung.

Erstellen Sie nun die OnClick Prozedur zu Button2 in Form1 und geben Sie nachfolgenden Quelltext ein (begin und end wird nachfolgend weggelassen).

```
form1.Caption := 'Hallo';
form2.Caption := 'Formblatt 2';
```

Verfahren Sie ebenso mit Button2 in Form2 und geben Sie den folgenden Quelltext ein:

```
form1.Caption := 'Formblatt 1';
form2.Caption := 'Delphi lernen';
```

PROGRAMME / 032 / UNITS_VERWENDEN.DPR

7.1.4. Typen- und Variablendeklarationsteil

Im Typen- und Variablendeklarationsteil werden unter anderem die globalen Variablen deklariert. Weiterhin ist es hier auch möglich, eigene Variablentypen zu definieren (dazu später). Außerdem finden Sie hier alle auf dem Formblatt vorhandenen Komponenten mit ihrem Namen und die bereits vorhandenen Prozeduren wieder.
Der Typen- und Variablendeklarationsteil umfasst die folgenden Deklarationen:

- Deklaration der globalen Variablen
- Deklaration der globalen Konstanten
- Deklaration eigener Variablentypen

7.1.5. Implementationsteil

Im Implementationsteil sind zunächst alle lokal definierten Units deklariert (uses - Anweisung). Weiterhin sind hier alle Quelltexte der einzelnen Prozeduren zu finden.

7.2. Übergabe und Rückgabe von Werten

7.2.1. Grundsätzliches

Bisher haben wir die Prozeduren, die vom Programm selbst erzeugt werden, lediglich um ihre Funktionalität erweitert. Dabei haben wir verschiedenen Komponenten Ereignisse zugeordnet, z.B. Berechnungen, Änderungen von Eigenschaften von Komponenten. Wir haben uns nicht weiter darum gekümmert, was dabei eigentlich passiert. Es besteht in Delphi aber auch die Möglichkeit selbst deklarierte Prozeduren und Funktionen zu entwickeln.

BEISPIEL:

Sie sollen ein Programm für die Grundrechenarten mit gemeinen Brüchen entwickeln. Die Ergebnisse der Berechnungen sollen dabei so weit wie möglich gekürzt werden. Man kann das Kürzen nun in jeder einzelnen Prozedur mit einarbeiten, da es sich aber immer um die gleiche Vorgehensweise handelt, ist es günstig dieses Kürzen als eigene Prozedur zu entwickeln und diese als Unterprogramm der eigentlichen Button-Prozedur aufzurufen. Die nachfolgende Grafik verdeutlicht die entsprechenden Aufrufe:

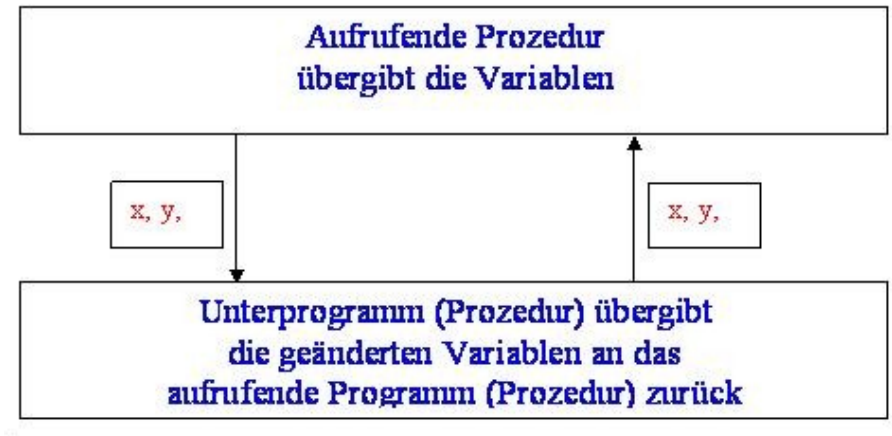


Abb. 85: Variablenübergabe 1

Prozeduren und Funktionen sind letztendlich nichts anderes als die Unterprogramme des Gesamtprogramms.

Beispiel für die Erstellung einer eigenen Prozedur:

```
...
type
  ...
  procedure Button1Click(Sender: TObject);
  procedure p_addition (var a, b, su : real);
  ...

implementation

{$R *.dfm}

procedure TForm1.p_addition (var a, b, su : real);
begin
  su := a + b;
end;

procedure TForm1.Button1Click(Sender: TObject);
var x, y, z : real;
begin
  x := strtofloat (edit1.Text);
  y := strtofloat (edit2.Text);
  p_addition (x, y, z);
  edit3.Text := floattostr (z);
end;
```

PROGRAMME / 033 / Prozeduren_1.DPR

7.2.2. Übergabe und Rückgabe von Werten

Es bestehen die folgenden Möglichkeiten zur Deklaration und Verwendung eigener Prozeduren:

```
procedure p_beispiel1;
procedure p_beispiel2 (a, b : real);
procedure p_beispiel3 (var a, b : real);
```

zu Beispiel 1:
An diese Prozedur werden keine Variablen (Werte) übergeben, die Prozedur gibt an das rufende Programm keine Daten zurück.
Unterprogramme dieser Art können eingesetzt werden, um z.B. bestimmte Komponenten zu initialisieren, Grafikbefehle auszuführen oder mit globalen Variablen zu arbeiten.

zu Beispiel 2:
Dieser Prozedur werden Werte aus dem rufenden Programm übergeben. Diese Werte können innerhalb des Unterprogramms verändert werden. Da die veränderten Daten an das rufende Programm nicht zurückgegeben werden, liegen diese im rufenden Programm unverändert vor.
Diese Art der Übergabe von Variablen nennt man auch:

call by value

zu Beispiel 3:
Dieser Prozedur werden Werte aus dem rufenden Programm übergeben. Diese Werte können innerhalb des Unterprogramms verändert werden. Nach Abarbeitung des Unterprogramms werden die veränderten Werte wieder an das rufende Programm zurückgegeben. Die Werte liegen dann also in veränderter Form im rufenden Programm vor.
Diese Art der Übergabe von Variablen nennt man auch:

call by reference

Prozeduren wie in Beispiel 2 und 3 können auch kombiniert werden. Im einführenden Programm hätte die Deklaration der Prozedur p_addition dann auch folgendermaßen aussehen können:

```
procedure p_addition (a, b : real; var su : real);
```

Darstellung der Übergabearten call by value und call by reference:

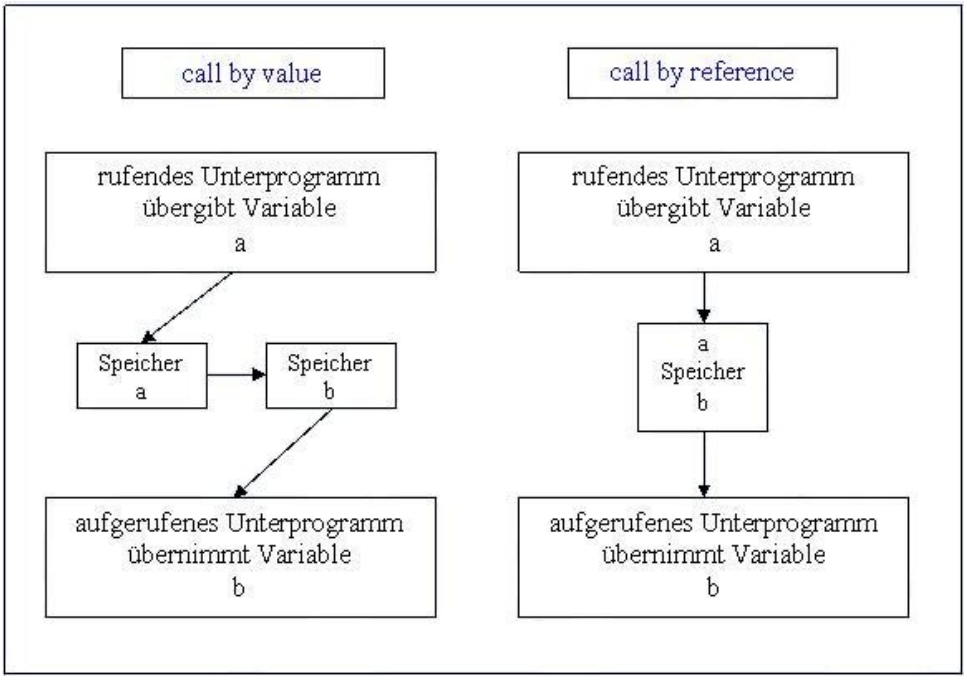


Abb. 86: call by value und call by reference

7.3. Deklaration und Implementation von Prozeduren

Will man eigene Prozeduren in einem Programm verwenden, dann muss man dabei zwei wesentliche Schritte beachten:

- Die Prozedur muss deklariert sein
- Die Prozedur muss implementiert sein

7.3.1. Deklaration von Prozeduren

Prozeduren können an verschiedenen Stellen des Programms deklariert werden.

- innerhalb der Liste der Prozeduren im Interface-Teil
- als Prozedur nach der Deklaration globaler Variablen und eigener Datentypen

Für die Implementierung ist folgendes zu beachten:
Werden Prozeduren innerhalb des Interface-Teils deklariert, dann muss die Implementation mit

```
procedure tform1.p_addition ...
```

beginnen.

Die Prozedur ist hier Bestandteil des Formblattes.

Erfolgt die Deklaration erst nach der Deklaration der globalen Variablen, dann beginnt die Implementation mit

```
procedure p_addition ...;
```

Die Prozedur ist hier nicht Bestandteil des Formblattes.
Die Deklaration von Prozeduren setzt sich immer aus dem Begriff procedure und dem Namen der Prozedur zusammen. Der Name darf dabei wieder keine Sonderzeichen enthalten und muss mit einem Buchstaben beginnen. Namen von Prozeduren sollten immer mit p_ ... beginnen.

7.3.2. Implementation von Prozeduren

Die Implementation der eigenen Prozeduren unterscheidet sich nicht von den uns bereits bekannten Implementierungen. Die Variablen, die dabei bereits in der Deklaration der Prozedur vorkommen dürfen dabei nicht noch einmal deklariert werden. Es können aber durchaus noch weitere Variablen hinzukommen, die dann wie üblich vor dem ersten begin deklariert werden müssen. Die Implementierung der Unterprogramme kann an beliebiger Stelle innerhalb des Implementationsteils der Unit erfolgen.

7.4. Verwendung eigener Prozeduren

Werden Unterprogramme in anderen Prozeduren aufgerufen, so müssen die entsprechend der Deklaration vorhandenen Variablen an das Unterprogramm übergeben werden. Dabei müssen die nachfolgenden Regeln beachtet werden:

Beim Aufruf einer Prozedur entfällt der Begriff procedure.

Übergabe call by value

BEISPIEL:

```
procedure p_addition (a, b : real);
```

Es müssen hierbei an das Unterprogramm zwei Zahlen im Realformat übergeben werden. Die Bezeichnung der Variablen muss dabei nicht unbedingt mit den Bezeichnungen in der Deklaration übereinstimmen. Es können aber auch einzelne Werte übergeben werden.

Beispiele für hier gültige Prozeduraufrufe:

```
p_addition (a, b);
p_addition (zahl1, zahl2);
p_addition (3, 4);
```

Übergabe call by reference

BEISPIEL:

```
procedure p_addition (var a, b : real);
```

In diesem Fall werden an das Unterprogramm wieder zwei Zahlen im Real-Format übergeben. Da das Unterprogramm ggf. geänderte Daten an das rufende Programm zurückgibt, können hier nicht einzelne Werte übergeben werden.

Beispiele für gültige Prozeduraufrufe

```
p_addition (a, b);
p_addition (zahl1, zahl2);
```

aber nicht:

```
p_addition (3, 4);
```

An dieser Stelle noch einmal einiges zu den beiden Übergabeformen call by value und call by reference. Bei der Übergabe call by value handelt es sich letztendlich um eine Initialisierung der im Unterprogramm benötigten Variablen. Diesen Variablen wird hier neuer Speicherplatz zugeordnet. Das heißt der benötigte Arbeitsspeicher wird größer.

Bei der Übergabe call by reference handelt es sich letztendlich um eine Übergabe von Speicherplatz. Der Wert der Variablen wird aus dem gleichen Speicher gelesen in dem das rufende Programm die Variable abgelegt hat. Daher wird kein neuer Speicherplatz benötigt, Änderungen an der Variablen (und somit an den Inhalt des Speicherplatzes) wirken sich daher aber auch auf die Variable im rufenden Programm aus.

Das Beispielprogramm stellt die unterschiedlichen Möglichkeiten zur Variablenübergabe dar.

PROGRAMME / 034 / Prozeduren_2.DPR

7.5. Die Deklaration und Implementation von Funktionen

7.5.1. Besonderheiten der Funktionsdeklaration

Funktionen haben im Gegensatz zu Prozeduren i.d.R. nur einen Rückgabewert. Dieser Rückgabewert muss nicht den gleichen Typ besitzen, wie die Variablen, die an die Funktion übergeben werden. Der Typ der Rückgabe wird dabei als Typ in der Funktionsdeklaration angegeben.

BEISPIEL:

```
function f_addition (var a, b : real) : real;
```

Die Angabe des Typs nach der Klammer bezieht sich dabei auf den Typ des zurückgegebenen Wertes. Die Übergabeformen call by value und call by reference behalten aber weiterhin ihre Gültigkeit. Mit einer Funktion will man im Allgemeinen einer Variablen ein ganz bestimmten Wert zuordnen, während mit einer Prozedur auch die Eigenschaften von Komponenten verändert werden sollen. Dies ist auch mit einer Funktion erreichbar.

7.5.2. Deklarieren und Implementieren einer Funktion

Die Deklaration und Implementierung einer Funktion erfolgt weitestgehend analog der Deklaration und Implementierung einer Prozedur. Wichtige Unterschiede sind:

- Bei einer Funktion muss der Typ des Rückgabewertes mit angegeben werden.
- Bei einer Funktion muss der Wert der Rückgabe mit implementiert werden.

zu 1. Deklaration

BEISPIEL:

```
function f_addition (var a, b : real) : real;
```

zu 2. Implementierung der Rückgabe

```
function f_addition (var a, b : real) : real;
var su : real;
begin
  su := a + b;
  f_addition := su;
end;
```

Zwei Beispiele zur Nutzung und Implementierung eigener Funktionen.

PROGRAMME / 035 / Funktionen_1.DPR
PROGRAMME / 036 / Funktionen_2.DPR

7.5.3. Verwendung eigener Funktionen

Eigene Funktionen werden genau wie Standardfunktionen verwendet. Das heißt, es müssen die Werte übergeben werden, von denen die Funktion berechnet werden soll. Der Rückgabewert muss unmittelbar einer Variablen zugeordnet werden.

BEISPIEL:

```
a := strtofloat (edit1.text);
b := strtofloat (edit2.text);
summe := f_addition (a, b);
```

7.6. Einbinden von Prozeduren und Funktionen aus anderen Units

Um bestimmte Funktionen und Prozeduren in mehreren Formularen oder Anwendungen einbinden zu können, ist es sinnvoll diese Funktionen und Prozeduren in eigenen Units zu verwalten.
Die Funktionen werden dabei genauso behandelt, wie oben beschrieben. Der Name der Unit, in der die Funktion oder Prozedur enthalten ist, ist im uses Teil der aufrufenden Funktion mit einzubinden.

Im Beispielprogramm erfolgt die Deklaration und Implementierung der Funktion in der Unit U_funktion. Es ist darauf zu achten, dass diese Unit in der Unit1 eingebunden ist.

PROGRAMME / 037 / Funktionen_3.DPR

Fragen und Aufgaben
7. Erstellen eigener Prozeduren

Grundlagen

1. Welche Möglichkeiten gibt es in Delphi weitere Units in einer anderen Unit einzubinden? Worauf ist dabei zu achten?
2. Welche wesentlichen Bestandteile enthält eine Unit?
3. Worin unterscheidet sich eine Unit, die einem Formblatt zugeordnet ist von einer Unit ohne Formblatt.
4. Wozu kann man Units ohne Formblatt nutzen?
5. Wie kann der Name einer Unit geändert werden?
6. Erklären Sie die Unterschiede der Übergabearten für Variablen call by value und call by reference.
7. Worauf müssen Sie achten, wenn Sie Funktionen in einer gesonderten Unit deklarieren und implementieren.
8. Worin bestehen die Unterschiede zwischen einer Funktion und einer Prozedur?

Programmieraufgaben

9. Erstellen Sie ein Delphi-Programm mit zwei Formblättern. Jedes Formblatt erhält eine Edit-Komponente. Beim Schreiben in der Edit-Komponente des Formblattes 1 soll dieser Text sofort in die Edit-Komponente von Formblatt 2 übertragen werden.

10. Erstellen Sie ein Delphi-Programm mit zwei Formblättern. Das Formblatt 2 soll über einen entsprechenden Button in Form1 sichtbar gemacht werden können und über einen entsprechenden Button in Form2 wieder unsichtbar.

11. Lösungen/007/023/Project1.dpr
    Erstellen Sie eine Delphi Anwendung mit zwei Formblättern. In Form1 soll die Berechnung e ines Rechtecks vorgenommen werden können, in Form2 die Berechnung eines Quaders.

12. Lösungen/007/024/Project1.dpr
    Erstellen Sie eine Delphi-Anwendung mit drei Formblättern. In Form 1 sollen die Umrechnungen von Einheiten der Länge in Meter vorgenommen werden, in Form2 die Umrechnungen von Einheiten der Fläche, in Form3 die Umrechnungen von Einheiten des Volumens (jeweils in m2 bzw. m3). Nutzen Sie dazu RadioGroups. Es soll dabei von jedem Programmteil (Formblatt) direkt in die anderen Formblätter gewechselt werden können.

13. Lösungen/007/025/Project2.dpr
    Erstellen Sie ein Programm zur Kreisberechnung. Der Nutzer soll bei der Eingabe einer negativen Zahl mit einer MessageBox auf seinen Fehler aufmerksam gemacht werden. Die Berechnung des Kreisumfanges und des Kreisflächeninhaltes soll in einer gesonderten Prozedur erfolgen. Dabei ist der Radius des Kreises zu übergeben, der Flächeninhalt und Umfang soll zurückgegeben werden.

14. Erstellen Sie ein Programm zum Lösen von Aufgaben mit den vier Grundrechenoperationen mit Ganzen Zahlen. Die Auswahl der Rechenoperation soll über eine RadioGroup erfolgen. Außerdem soll die Berechnung der Werte über einzelne Funktionen erfolgen. Beachten Sie dabei die Division durch 0.

15. Lösungen/007/026/Project1.dpr
    Erstellen Sie eine Delphi-Anwendung mit zwei Formblättern. Auf dem ersten Formular soll die Berechnung eines Rechtecks ausgeführt werden, auf dem zweiten die Berechnung eines Quaders. Für die Berechnungen der Werte sind entsprechende Funktionen vorzusehen. Diese Funktionen sind dabei in einer weiteren Unit abzulegen.

16. Erstellen Sie eine Delphi-Anwendung zur Berechnung der allgemeinen quadratischen Gleichung. Das Einlesen der Daten aus den Edit-Komponenten soll dabei in einer gesonderten Prozedur erfolgen.

17. Lösungen/007/027/Project1.dpr
    Erstellen Sie ein Delphi-Programm zur Umwandlung einer Dezimalzahl in eine Dualzahl mit Hilfe des Divisionsverfahrens. Das Einlesen und Ausgeben der Zahlen soll über ButtonClick erfolgen. Die Berechnung in einer gesonderten Prozedur, die in einer eigenen Unit verwaltet wird.

18. Erstellen Sie ein Programm für einen einfachen Taschenrechner. Die Eingabe der Daten soll dabei über Edit-Komponenten erfolgen. Die Berechnungen in gesonderten Funktionen. Standardfunktionen können dabei genutzt werden. Der Taschenrechner soll folgende Rechenoperationen ausführen können:
    a. die vier Grundrechenoperationen
    b. Quadrieren und Quadratwurzel
    c. Anzeigen der Zahl PI für Berechnungen
    d. Berechnung von Sinus und Kosinus für Winkeleingaben in Grad (Umrechnung über Funktion)

8. Grafikerstellung

8.1. Einfügen von Bildern

Es ist in Delphi möglich fertige Bilder und Grafiken auf dem Formblatt einzufügen. Es ist aber auch möglich Grafiken zur Laufzeit dynamisch zu erstellen. Damit wollen wir uns in den nächsten Kapiteln befassen.

Um ein bereits vorhandenes Bild auf der Oberfläche einbinden zu können, muss zunächst die Komponente Image (aus Komponentenleiste "Zusätzlich") auf der Oberfläche platziert werden. Über den Objektinspektor - Eigenschaftenseite - Picture wird dann die gewünschte Grafik ausgewählt.

Abb. 87: Ansicht des Bildeditors mit geladener Grafik

Manchmal wird dabei nur ein Teil der Grafik angezeigt. Setzen Sie dann die Eigenschaft "Proportional" im Objektinspektor auf True. Im Programmbeispiel sehen Sie eine Oberfläche mit eingebundener Grafik. Weitere Eigenschaften der Image-Komponente finden Sie in der Delphi-Hilfe. Im ersten Programmbeispiel finden Sie eine Darstellung mit eingebundener Grafik, das zweite Programmbeispiel zeigt eine "Diashow". Dabei wurde zusätzlich die Komponente TIMER benutzt.

PROGRAMME / 038 / Bilder_1.DPR
PROGRAMME / 039 / Diashow.DPR

8.2. Standardgrafikobjekte

In Delphi sind die folgenden Standardgrafikobjekte vorhanden:

- Kreis
- Ellipse
- Quadrat
- Rechteck
- Quadrat und Rechteck mit abgerundeten Kanten

Um diese Objekt auf der Oberfläche zu platzieren, wählt man zunächst die Komponente Shape (aus Komponentenleiste "Zusätzlich") aus und platziert diese auf der Oberfläche. Über den Objektinspektor kann nun das Aussehen dieses Objekts verändert werden.

Farbe und Art der Füllung:	über "Brush"
Art des Objekts:	über "Shape"
Art der Umrandung:	über "Pen"

Die Eigenschaften dieser Grafikobjekte können zur Laufzeit verändert werden. Wechseln Sie dazu in die Ereignisseite des Objektinspektors und wählen Sie das Ereignis aus, auf welches das Programm reagieren soll. Im Beispielprogramm sind einige dieser Ereignisse implementiert.

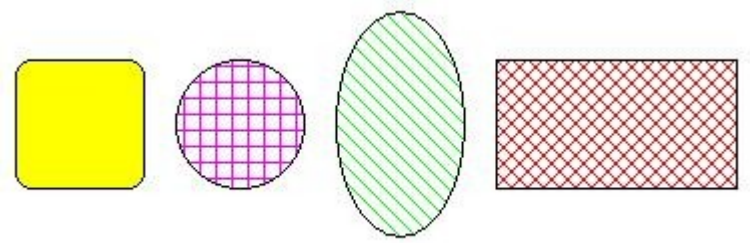


Abb. 88: Beispiele für Standardgrafikobjekte

Weitere Eigenschaften der Shape-Komponente finden Sie in der Delphi-Hilfe

Im Programmbeispiel werden mehrere Shape-Objekte dargestellt. Diese können durch bestimmte Ereignisse verändert werden.

PROGRAMME / 040 / Standardgrafikobjekte.DPR

8.3. Grafiken zur Laufzeit erstellen

8.3.1. Die Methode TCanvas

Die Methode TCANVAS stellt den Grafikbereich einer Komponente zur Verfügung. Will man also bestimmte Grafikobjekte selbst zeichnen muss vor jeder Anweisung CANVAS stehen.
CANVAS stellt die folgenden Grafikelemente zur Verfügung:

ELLIPSE	Zeichnen einer Ellipse
MOVETO, LINETO	Zeichnen einer Linie
RECTANGLE	Zeichnet ein Rechteck
ROUNDRECT	Zeichnet ein Rechteck mit abgerundeten Kanten
TEXTOUT	Gibt Position für eine Textausgabe und den Text an
PIXELS	Zeichnet einen Punkt

Weiterhin können Einstellungen für die Strichstärke und die Strichfarbe, für die Textformatierungen u.v.a. mehr vorgenommen werden.

8.3.2. Der Grafikbildschirm

Entgegen unserer üblichen Koordinatensysteme verläuft hier die y-Richtung von oben nach unten. Dies muss bei entsprechenden Darstellungen immer berücksichtigt werden.
Weiterhin ist zu beachten, dass die Koordinaten nur mit INTEGER-Werten angegeben werden können. Bei der Darstellung von Grafiken auf dem Bildschirm werden dabei die einzelnen Pixel angesteuert. Daher ist die maximale Anzahl in x- und y-Richtung auch von der Bildschirmauflösung abhängig.

Darstellung des Grafikbildschirms (Beispiel)

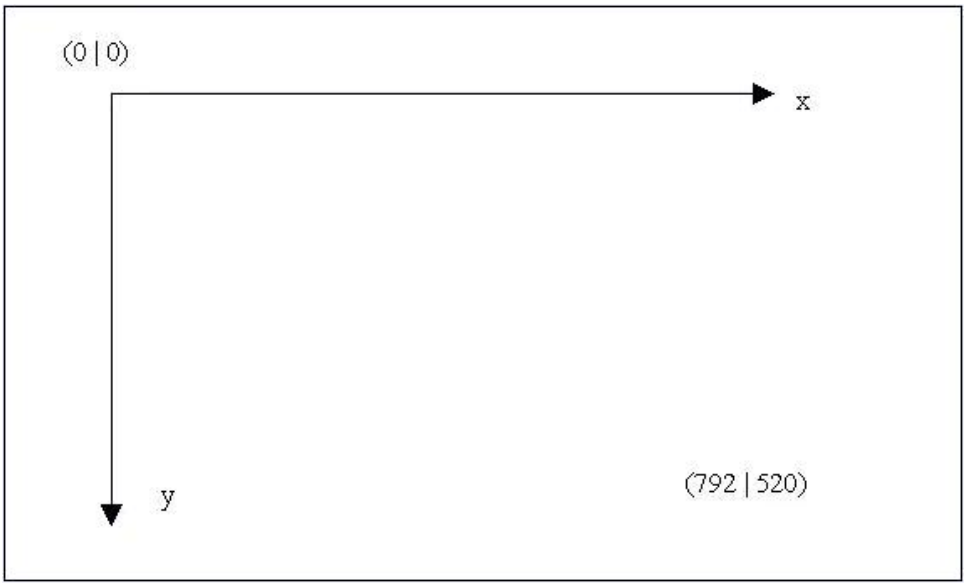


Abb. 89: Darstellung des Grafikbildschirms

8.3.3. Grafiken erstellen

Im Folgenden soll die Grafikerstellung mit Hilfe einiger einfacher Prozeduren näher erläutert werden.

Zeichnen einer Linie

Ähnlich, wie mit einem normalen Stift auf dem Papier muss dem Stift zunächst der Ansatzpunkt der Darstellung mitgeteilt werden. Dies geschieht mit dem Befehl MOVETO.
Von dieser angegebenen Position aus, kann dann eine Linie gezeichnet werden, von der dann nur noch der Endpunkt bezeichnet werden muss. Die Syntax zum Zeichnen einer Linie lautet daher wie folgt:

```
canvas.moveto (x1, y1);
canvas.lineto (x2, y2);
```

x1, y1, x2, y2 geben dabei die Koordinaten des Anfangs- und des Endpunktes an.

Will man einen Polygonzug zeichnen, dann ist es nicht notwendig, jedes Mal den Stift mit MOVETO zu positionieren. Der Stift befindet sich nach dem Zeichnen einer Linie an deren Endpunkt. Soll von dort aus weitergezeichnet werden, kann sofort wieder LINETO geschrieben werden.

```
canvas.moveto (x1, y1);
canvas.lineto (x2, y2);
canvas.lineto (x3, y3);
```

Im Folgenden werden wir die Farbe der Linie und deren Breite verändern. Diese Mitteilungen müssen Sie dem Programm in entsprechender Form geben.
Die Syntax für diese Anweisungen lautet:

```
canvas.pen.color := FARBE;
canvas.pen.width := BREITE;
```

Dabei werden die Farben wieder in der uns bekannten Form angegeben. Die Angabe der Breite erfolgt in Pixel, muss also als positive INTEGER-Zahl angegeben werden.

Sie merken sicher selbst, dass es ziemlich umständlich ist, vor jedem Grafikbefehl erst CANVAS und ggf. sogar noch eine Komponente schreiben zu müssen. Zwar kann man sich durch kopieren helfen, aber trotzdem wird der Quelltext bald ziemlich lang und unübersichtlich. Auch hier kann man sich helfen. Mit dem folgenden Quelltext wird das gleiche erreicht, wie oben (Button2), nur eben mit etwas weniger Schreibaufwand.

```
procedure TForm1.Button2Click(Sender: TObject);
var x, y : integer;
begin
  x := strtoint (edit1.Text);
  y := strtoint (edit2.Text);
  button3.Enabled := true;
  with pagecontrol1.canvas do
    begin
      Pen.Color := colorbox1.Selected;
      Pen.Width := strtoint (edit3.Text);
      Moveto (20, 20);
      Lineto (x, y);
    end;
end;
```

Mit diesem Befehl wird offenbar erreicht, dass für alle folgenden Befehle "PAGECONTROL1.CANVAS" davor gedacht wird. Es ist aber auch möglich andere Befehle wie z.B. IF-THEN-ELSE oder FOR-TO-DO Schleifen mit einzuarbeiten. Dieser Befehl with ... do kann auch für andere Befehle die für weitere Formatierungen eingesetzt werden sollen verwendet werden. Er kann auch geschachtelt werden.

```
with label1 do
  begin
    caption := 'TEXT';
    color   := clyellow;
    with font do
      begin
        Color := clblue;
        Name  := 'Symbol';
        Size  := 15;
      end;
  end;
```

Im folgenden Beispielprogramm wird eine Labelkomponente mit Hilfe einer WITH-Anweisung formatiert.

PROGRAMME / 041 / WITH_Anweisung.DPR

In den folgenden Beispielen wurde dieses Canvas einige Male weggelassen, um die Übersichtlichkeit der Darstellung zu erhalten.

Zeichnen von Rechteck und Quadrat

Von einem Rechteck muss zur Darstellung der obere linke und der untere rechte Punkt angegeben werden. Hierbei handelt es sich selbstverständlich wieder um INTEGER-Werte.

Syntax:

```
rectangle (x1, y1, x2, y2);
```

Da zum Zeichnen eines Quadrates keine spezielle Funktion vorhanden ist, muss diese Zeichnung wieder über entsprechende Rechtecke realisiert werden.

Zeichnen von Ellipse und Kreis

Auch das Zeichnen eines Kreises muss über die Darstellung einer Ellipse erfolgen. Bei der Zeichnung einer Ellipse werden dabei nicht die entsprechenden Radien angegeben, das Zeichnen erfolgt über die Kennzeichnung der Rechteckpunkte, welches diese Ellipse umgibt.

Syntax:

```
ellipse (x1, y1, x2, y2);
```

Punkte

Punkte werden durch die Angabe ihrer Koordinaten gezeichnet.

Syntax:

```
pixels [x, y];
```

ACHTUNG:

Die Angabe der Koordinaten erfolgt in eckigen Klammern.

Texte

Hierfür muss die Position in Integer-Werten und der auszugebende String angegeben werden.

Syntax:

```
textout (x, y, 'TEXT');
```

Die Textausgabe kann wieder auf verschiedene Weise formatiert werden. Der nachfolgende Programmauszug gibt einen roten fett gestalteten Text in der Größe 30 und der Schriftart Courier New aus.

```
canvas.Font.Color := clred;
canvas.Font.Height := 30;
canvas.Font.Style := [fsbold];
canvas.Font.Name := 'Courier New';
canvas.TextOut(20,20,'Hallo');
```

Für die Angaben kann wieder WITH .. DO verwendet werden.

Beispiele zur Erstellung von Grafiken zur Laufzeit.

PROGRAMME / 042 / GRAFIK1.DPR
PROGRAMME / 043 / Grafik2.DPR

Programmieraufgabe 25

Für das folgende Programm benötigen Sie auf Ihrer Oberfläche vier Edit-Komponenten und einen Button. Die Edit-Komponenten dienen dabei dem Einlesen des Anfangs- und des Endpunktes der Linie.

```
procedure TForm1.Button1Click(Sender: TObject);
var x1, y1, x2, y2 : integer;
begin
  x1 := strtoint (edit1.Text);
  y1 := strtoint (edit2.Text);
  x2 := strtoint (edit3.Text);
  y2 := strtoint (edit4.Text);
  canvas.MoveTo(x1, y1);
  canvas.LineTo(x2, y2);
  edit1.Text := edit3.Text;
  edit2.Text := edit4.Text;
end;
```

Mit den letzten beiden Anweisungen wird erreicht, dass für den Ansatzpunkt der folgenden Linie sofort die Koordinaten des Endpunktes der letzten Linie eingetragen werden. Diese können natürlich auch verändert werden.

PROGRAMME / 044 / Linien_Zeichnen.DPR

Programmieraufgabe 26

Im Folgenden soll ein Programm entwickelt werden, welches einen solchen geschlossenen Polygonzug zeichnet. Sie benötigen dazu zwei Edit-Komponenten und drei Button. Dabei werden wir über die Eigenschaft ENABLED einige Button abwechseln dem Nutzer nicht zur Verfügung stellen. Außerdem benötigen wir die globalen Variablen xa und ya, die den Anfang des Polygons kennzeichnen.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  xa := strtoint (edit1.Text);
  ya := strtoint (edit2.Text);
  button1.Enabled := false;
  button2.Enabled := true;
  canvas.MoveTo(xa, ya);
end;

procedure TForm1.Button2Click(Sender: TObject);
var x, y : integer;
begin
  x := strtoint (edit1.Text);
  y := strtoint (edit2.Text);
  button3.Enabled := true;
  canvas.Lineto(x, y);
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
  canvas.LineTo(xa, ya);
  button1.Enabled := true;
  button2.Enabled := false;
  button3.Enabled := false;
end;
```

Probieren Sie das Programm aus. Soll der Stift an einer anderen Stelle wieder ansetzen, muss natürlich wieder MOVETO verwendet werden.

PROGRAMME / 045 / Polygonzug1.DPR

Programmieraufgabe 27

Wir wollen das letzte Beispiel so ergänzen, dass die Auswahl einer Farbe und einer Stiftbreite für die Darstellung möglich wird. Für die Auswahl der Farbe nutzen wir dabei eine Colorbox. Die Breite soll über eine Edit-Komponente angegeben werden. Außerdem wollen wir erreichen, dass die Darstellung auf einer gesonderten Komponente ausgegeben wird, damit die Darstellung nicht immer zwischen den Eingabekomponenten ausgegeben wird. Sie benötigen daher auf Ihrer Oberfläche zusätzlich eine ColorBox, eine Edit-Komponente und eine Komponente PageControl. Diese sollte so groß wie möglich dargestellt werden. Beachten Sie im Quelltext insbesondere auch die Ausgabe mit Canvas.

Die Prozedur für Button2 ändert sich dann wie folgt:

```
procedure TForm1.Button2Click(Sender: TObject);
var x, y : integer;
begin
  x := strtoint (edit1.Text);
  y := strtoint (edit2.Text);
  button3.Enabled := true;
  pagecontrol1.canvas.Pen.Color := colorbox1.Selected;
  pagecontrol1.canvas.Pen.Width := strtoint (edit3.Text);
  pagecontrol1.canvas.Lineto(x, y);
end;
```

Ergänzen Sie die Prozeduren für Button1 und Button3 entsprechend. Ist für Button1 eine Angabe von Farbe und Breite notwendig? Im Quelltext steht jetzt vor dem Befehl CANVAS noch zusätzlich PAGECONTROL1. Diese Ergänzung ist notwendig, um dem Programm zu sagen, dass die Darstellung auf dieser Komponente erfolgen soll.

PROGRAMME / 046 / Polygonzug2.DPR

8.4. Einiges über Farben

Farben werden auch in Delphi Hexadezimal kodiert. Alle Farben werden dabei über die Mischung der Farben blau, gün und rot realisiert. Die Farben ergeben sich dabei folgendermaßen:

00	00	00	00
Intensität der Farbe	Blauanteil	Grünanteil	Rotanteil

Einige Farbkodierungen

Farbe	Kodierung	Name in Delphi	Beispiel
Schwarz	00 000000	clblack	
Blau	00 FF0000	clblue	
Grün	00 008000	clgreen	
Rot	00 0000FF	clred	
Weiß	00 FFFFFF	clwhite	
Maroon	00 000080	clmaroon	
Gelb	00 00FFFF	clyellow	

Mit Hilfe der entsprechenden Hexadezimalkodierung können demzufolge insgesamt 16^6 = 16.777.216 verschiedene Farben definiert werden. Die entsprechende Kodierung kann in Delphi auf verschiedene Art und Weise erfolgen.

Kodierung mit Hilfe des Farbnamens	z.B. clred, clyellow, clblue
Kodierung mit Hilfe einer Integerzahl	Zahlen von 0 ... 16.777.216
Kodierung mit Hilfe einer Hexadezimalzahl	Zahlen von 00 000000 ... 00 FFFFFF

Die nachfolgenden Anweisungen färben alle gleichartig das Formular gelb:

```
form1.color := clyellow;
form1.color := 65536;
form1.color := $0000FFFF;
```

Das Programmbeispiel stellt die unterschiedliche Zusammensetzung der Farben mit Hilfe eines Hexadezimalkodes dar.

PROGRAMME / 047 / Farben.DPR

Fragen und Aufgaben
8. Grafiken in Delphi

Grundlagen

1. Welcher grundlegenden Unterschiede bestehen zwischen Koordinatensystem in der Mathematik und den Bildschirmkoordinaten?
2. Geben Sie zwei Möglichkeiten zum Zeichnen eines Rechtecks auf dem Bildschirm an.

Programmieraufgaben

3. Erstellen Sie ein Formblatt mit mehreren eingebundenen Grafiken.

4. Erstellen Sie ein Programm zur Darstellung von Kreisen. Dabei sollen vom Nutzer der Mittelpunkt und der Radius eingegeben werden.

5. Erstellen Sie ein Programm zur Berechnung von Rechtecken. Das zu berechnende Rechteck soll dabei auch dargestellt werden. Wählen Sie dazu für eine Eingabeeinheit 10Pixel.

6. Lösungen/008/028/project1.dpr
   Erstellen Sie ein Programm zur Darstellung linearer Funktionen. Die Werte m und n sollen dabei vom Nutzer eingegeben werden können.

7. Erstellen Sie ein Programm zur Darstellung der quadratischen Funktion. Es sollen p und q vom Nutzer eingegeben werden. Weiterhin sind die Koordinaten des Scheitelpunktes und die Nullstellen der Funktion dem Nutzer anzuzeigen.

8. Lösungen/008/029/project1.dpr
   Erstellen Sie ein Grafikprogramm, welches auf Eingaben über die Tasten "Rechts", "Links", "Oben" und "Unten" reagiert. Ferner soll die Auswahl verschiedener Stiftfarben möglich sein.

9. Ergänzen Sie das Malprogramm mit der Möglichkeit zum Zeichnen eines geschlossenen Polygonzugs.

10. Lösungen/008/030/project2.dpr
    Erstellen Sie ein Grafikprogramm zur Darstellung der Körper: Würfel, Quader und Pyramide. Die Daten sind vom Nutzer einzugeben.

9. Strukturierte Datentypen

9.1. Vorbemerkungen

Nicht immer ist es günstig und möglich die einzugebenden Daten in einer einzelnen einfachen Variablen zu verwalten.

BEISPIEL 1:

Für eine Messwertreihe sollen die einzelnen Daten nacheinander aufgenommen, anschließend der Mittelwert der Daten berechnet und angezeigt werden. Hier ist es nicht mehr angebracht alle diese Eingabedaten über einzelne Variablen abzufragen, zumal auch die Anzahl der Messwerte nicht unbedingt vorher feststehen muss.

BEISPIEL 2:

Sie sollen die Verwaltung eines Sportvereins programmieren. Dazu sind von jedem Mitglied verschiedene Daten (z.B. Name, Geburtsdatum, Sportart) einzugeben und zu verwalten. Mit einfachen String-Variablen ist dies kaum möglich.

BEISPIEL 3:

Die Berechnung eines linearen Gleichungssystems erfordert eine klare Strukturierung der eingegebenen Daten für die einzelnen Koeffizienten. Soll dabei die Anzahl der Variablen und der Gleichungen auch noch vom Nutzer bestimmt werden, führt eine Programmierung mit einfachen Variablen unweigerlich in ein Chaos.

Um alle diese Probleme möglichst effizient lösen zu können, verwendet man in der Programmierung Arrays bzw. Records, mit denen wir uns im Folgenden näher befassen wollen.

9.2. Konstante (Statische) Arrays

9.2.1. Begriff

Statische Arrays (Felder) enthalten eine vom Programmierer bestimmte Anzahl zusammengehörender Daten desselben Typs.

Damit Sie eine gewisse Vorstellung von einem Array bekommen hier ein kleiner Vergleich.
In einem CD-Regal steht Ihnen eine begrenzte Anzahl von Einschüben für Ihre CDs zur Verfügung. An jede Stelle passt nur eine CD. Jetzt wäre es durchaus möglich diese Einschübe der Reihe nach durchzunummerieren. Jede CD hätte damit eine konkret festgelegte Nummer, die den Platz innerhalb des Regals kennzeichnet. Andererseits ist es Ihnen aber nicht möglich etwas anderes als eben CDs in diesem Regal abzustellen (z.B. Videokassetten). Damit haben wir also ein "CD-Array" mit dem Datentyp CD und der dazu begrenzten Anzahl Einschübe geschaffen.

Beispiel aus der Programmierung

Im Physikunterricht soll ein Experiment zum freien Fall mit Bestimmung der Fallbeschleunigung durchgeführt werden. Dazu werden für eine bestimmte Höhe jeweils 10 Zeitmessungen ausgeführt. Die Daten sind mit einem Delphi-Programm auszuwerten. Die einzelnen Messwerte können dabei günstig in einem Array verwaltet werden. Dabei wird jeder Messung die entsprechende Messungsnummer und der konkrete Zeitwert der Messung zugeordnet.

9.2.2. Deklaration eines Arrays

Wie die einfachen Variablen können auch Arrays lokal und global deklariert werden.

Array als lokale Variable

Die allgemeine Syntax zur Deklaration eines Arrays lautet wie folgt:

```
name : array [anfang .. ende] of typ;
```

Dabei bedeuten:

Name	Name der Arrayvariablen
[anfang .. ende]	Positionsnummern innerhalb des Arrays
of typ;	Variablentypen der Arrayinhalte

Wir erinnern uns:

Lokale Variablen "vergessen" nach Durchlauf der Prozedur ihren Inhalt. D.h. der zugehörige Speicherplatz wird wieder freigegeben und es kann somit nicht mehr auf diesen Inhalt zugegriffen werden. Daher ist es notwendig diese Arrays innerhalb der Prozedur zu initialisieren.
Es ist außerdem in jedem Fall dafür zu sorgen, dass bis zum Abschluss der Eingaben und der dazu zu treffenden Berechnungen die entsprechende Prozedur nicht verlassen wird.

Beispiel für die Deklaration eines lokalen Arrays:

```
procedure TForm1.Button1Click(Sender: TObject);
var feld1 : array [1..5] of integer;
begin
  ...
```

Wir wollen uns nun die Deklaration dieses Arrays etwas näher anschauen.

feld1 :	Ist der Bezeichner (Name) des Arrays. Es gelten die gleichen Regeln, wie für Einfache Variablentypen.
array	Ist der Befehl für die Erstellung eines Arrays
[1..5]	Gibt die Nummerierung der Daten innerhalb des Arrays an. Es werden in unserem Beispiel also Werte von Nr. 1 bis Nr. 5 aufgenommen. Gleichzeitig steht damit auch die maximale Anzahl der Arrayinhalte fest.
of integer;	Gibt den Datentyp der einzelnen Inhalte an. In dieses Array können also nur Daten vom Typ Integer aufgenommen werden.

Array als globale Variable

Der Vorteil der globalen Variablen liegt darin, dass die Daten

- Nach Durchlauf einer Prozedur erhalten bleiben
- Die Daten innerhalb des Arrays nicht initialisiert werden müssen

Aus diesen Gründen ist es meist vorteilhaft, Arrays global zu deklarieren.
Diese Deklaration erfolgt im Variablendeklarationsteil der Unit und sieht folgendermaßen aus:

```
var
  ...
  feld1 : array [1..5] of integer;
```

Die Deklaration selbst unterscheidet sich also nicht von der Deklaration eines lokalen Arrays, sie wird lediglich an einer anderen Stelle des Programms eingefügt.

9.2.3. Einfügen von Daten in ein Array

Mit der Deklaration eines Arrays wurden den einzelnen Positionen innerhalb des Arrays aber noch keine Werte zugewiesen. Wie dies geschieht wollen wir uns im Folgenden ansehen. Dabei sind dem Array folgende Werte mitzuteilen:

- Die Position innerhalb des Arrays an dem die Daten eingeschrieben werden sollen.
- Den Wert der Daten, die eingeschrieben werden sollen.

BEISPIEL:

An der Position 3 soll der Wert 7 eingeschrieben werden (vgl. Darstellung unten):

Syntax:

```
feld1 [3] := 7;
```

Die 3 in den eckigen Klammern gibt die Position an, an der der Wert nach der Ergibt-Anweisung eingeschrieben werden soll. Diese Position kann auch durch eine entsprechende Integer-Variable angegeben werden.

BEISPIEL:

```
...
pos := 3;
feld1 [pos] := 7;
...
```

Darstellung des Arrays Feld1:

Feld1 : array [1..5] of integer;

	1	2	3	4	5
Initialisiertes Array	0	0	0	0	0
nach Eingabe von Daten	2	5	7	1	3

Abb. 90: Darstellung eines Arrays

9.2.4. Lesen von Daten aus einem Array

Das Lesen der Daten erfolgt analog zu deren Eingabe. Es muss wieder die entsprechende Position angegeben werden, von der der Wert gelesen werden soll.

BEISPIEL:

Der Wert an der 3. Position soll ausgelesen werden:

Syntax:

```
wert := feld1 [3];
```

Es ist sicherlich klar, dass die Variable Wert in unserem Beispiel den gleichen Typ haben muss, wie die Daten innerhalb des Arrays. Natürlich ist es auch möglich die Position wieder über eine entsprechende Variable anzugeben.

BEISPIEL:

```
...
pos := 3;
wert := feld1 [pos];
...
```

Programmieraufgabe 28

Im folgenden Beispiel aus der Physik wird eine Messreihe ausgewertet. Dabei werden für eine vorgegebene Höhe 10 Zeiten für den Fall gemessen und die sich daraus ergebende Fallbeschleunigung berechnet. Zum Nachvollziehen des Programmbeispiels benötigen Sie auf dem Formblatt 2 Edit-Komponenten und einen Button.

```
{ 1}   unit Unit1;
{ 2}
{ 3}   interface
{ 4}   uses
{ 5}     Windows, Messages, SysUtils, Variants, Classes, Graphics,
{ 6}     Controls, Forms, Dialogs, StdCtrls;
{ 7}
{ 8}   type
{ 9}     TForm1 = class(TForm)
{10}       Edit1: TEdit;
{11}       Button1: TButton;
{12}       Edit2: TEdit;
{13}       procedure Button1Click(Sender: TObject);
{14}
{15}   private
{16}      { Private-Deklarationen }
{17}   public
{18}      { Public-Deklarationen }
{19}   end;
{20}
{21}   var
{22}     Form1   : TForm1;
{23}     Zeiten : array [1..10] of real;
{24}
{25}   implementation
{26}
{27}   {$R *.dfm}
{28}
{29}   procedure TForm1.Button1Click(Sender: TObject);
{30}   var i       : integer;
{31}       t, sum : real;
{32}       Mittel : real;
{33}       h, g    : real;
{34}   begin
{35}     {Einlesen der Daten}
{36}     for i := 1 to 10 do
{37}       begin
{38}         t := strtofloat(inputbox ('Geben Sie die Zeit
{39}              ein','',''));
{40}         Zeiten [i] := t;
{41}       end;
{42}     {Auslesen und Berechnung der Summe}
```

```
{43}      sum := 0;
{44}      for i := 1 to 10 do
{45}        sum := sum + zeiten [i];
{46}      {Berechnungen des Mittelwertes und der Fallbeschleunigung}
{47}      Mittel := sum / 10;
{48}      h := strtofloat (edit1.Text);
{49}      g := 2*h/(mittel*mittel);
{50}      edit2.Text := floattostr (g);
{51}    end;
{52}    end.
```

Erläuterungen zum Programmtext:

Zeile Nr.	Erläuterung
23	Deklaration des globalen Arrays
36	Mit Hilfe der FOR-Schleife werden nacheinander 10 Einzelwerte abgefragt
38	Abfrage erfolgt über eine Inputbox
40	Die Daten werden an die entsprechende Stelle in das Array übernommen
43	Initialisierung der lokalen Variable sum
44-45	Die Daten innerhalb des Arrays werden nacheinander gelesen und zur Variablen sum addiert
47-50	Berechnungen und Ausgabe

PROGRAMME / 048 / Fallbeschleunigung.DPR
PROGRAMME / 049 / Zensurenauswertung.DPR

9.3. Mehrdimensionale Arrays

9.3.1. Besonderheiten eines mehrdimensionalen Arrays

Die bisher betrachteten Arrays hatten die Dimension 1. Das heißt, Daten konnten nur nacheinander in ungeordneter Reihenfolge aufgenommen werden. Für einige Anwendungen ist es jedoch günstig zusammengehörige Daten zugeordnet aufzuschreiben. Ein Beispiel dafür ist das Lösen eines linearen Gleichungssystems. Hierbei schreibt man (mathematisch) die Gleichungen so untereinander, dass die gleichen Variablen auch untereinander stehen.

BEISPIEL:
$$2x + 3y = 5$$
$$4x - 2y = 4$$

Eine solche Ausrichtung ist mit unseren bisherigen Arrays nicht möglich. Um eine entsprechende Zuordnung trotzdem erreichen zu können benutzt man in der Programmierung mehrdimensionale Arrays. Dabei werden wir im Folgenden auf zweidimensionale Arrays näher eingehen. Die gesamten Erläuterungen dazu treffen aber auch auf höherdimensionale Arrays zu und können entsprechend angewendet werden.

Darstellung eines zweidimensionalen Arrays:

Feld2 : array [1..5,1..3] of integer;

	1	2	3	4	5
1	4	9	2	7	6
2	2	5	7	1	3
3	0	1	3	4	5

Abb. 91: Darstellung eines zweidimensionalen Arrays

9.3.2. Deklaration eines mehrdimensionalen Arrays

Wie ein einfaches konstantes Array kann auch ein mehrdimensionales Array als lokale und globale Variable deklariert werden. Auf den entsprechenden Unterschied soll hier nicht noch einmal eingegangen werden. Um ein mehrdimensionales Array zu deklarieren werden lediglich die weiteren Zählbereiche des Arrays angegeben.

BEISPIEL:

Deklaration eines einfachen Arrays

```
feld1 : array [1..5] of real;
```

Deklaration eines zweidimensionalen Arrays

```
feld2 : array [1..5,1..3] of real;
```

9.3.3. Einfügen von Daten in ein mehrdimensionales Array

Wenn man Daten in ein zweidimensionales Array schreiben möchte, muss man natürlich auch hier die Position innerhalb des Arrays angeben. Da wir hier aber zwei Positionierungen vornehmen müssen, ist es auch klar, dass dabei zwei Werte angegeben werden müssen.

BEISPIEL:

```
var
  Form1: TForm1;
  feld2 : array [1..5,1..2] of real;

implementation

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
var x, y : integer;
    wert : real;
begin
  x := strtoint (edit2.text);
  y := strtoint (edit3.text);
  wert := strtofloat (edit1.Text);
  feld2 [x,y] := Wert;
end;
```

9.3.4. Lesen von Daten aus einem mehrdimensionalen Array

Auch beim Lesen müssen wieder beide Koordinaten des Arrays angegeben werden.

BEISPIEL:

```
x := 4;
y := 2;
wert := feld2 [x,y];
```

9.3.5. Programmbeispiel: Lösen eines linearen Gleichungssystems

Günstig ist der Lösungsweg mit Hilfe der Cramerschen Regel. Dabei wird zuerst die Determinante des gegebenen GLS berechnet. Versuchen Sie die einzelnen Schritte nachzuvollziehen.

PROGRAMME / 050 / Gleichungssystem.DPR

9.4. Dynamische Arrays

9.4.1. Begriff

Bei den bisher betrachteten konstanten Arrays wurde die Anzahl der Daten, die vom Array aufgenommen werden können, vom Programmierer festgelegt. Im Folgenden wollen wir Arrays betrachten, bei denen die Anzahl der aufnehmbaren Daten nicht begrenzt ist, sondern erst zur Laufzeit vom Nutzer selbst bestimmt wird. Diese Arrays nennt man auch dynamische Arrays.

9.4.2. Dynamische Arrays deklarieren

Beispiel zur Deklaration eines dynamischen Arrays

```
feld3 : array of integer;
```

Erläuterung zur Deklaration:

- Die Deklaration eines dynamischen Arrays erfolgt also ohne Angabe des Bereiches
- Der Typ der Daten im Array muss wieder mit angegeben werden.
- Die Zählung der Daten in einem dynamischen Array beginnt immer mit 0.
- Dynamische Arrays können lokal und global deklariert werden.

9.4.3. Dynamische Arrays benutzen

Problemstellungen beim Arbeiten mit dynamischen Arrays:

- Festlegen der Länge eines dynamischen Arrays
- Bestimmen der Länge eines dynamischen Arrays
- Einfügen eines neuen Datensatzes
- Lesen eines Datensatzes
- Löschen eines Datensatzes

Festlegen der Länge eines dynamischen Arrays

Syntax und Beispiel:

```
setlength (feld3, 5);
```

Mit dieser Anweisung wird dem dynamischen Array mit dem Bezeichner feld3 die Länge 5 (gezählt von 0 bis 4) zugewiesen. Für die Angabe der Länge können hier wieder beliebige positive Integer-Variablen benutzt werden. Die Länge des Arrays kann dabei während der Programmabarbeitung beliebig oft verändert werden.

Bestimmen der Länge eines dynamischen Arrays

Syntax und Beispiel:

```
n := length (feld3);
```

Mit dem Befehl length bestimmen wir die Länge eines Arrays. Der bestimmte Wert ist dabei eine Ganzzahl (Integer). Dieser Wert wird einer entsprechenden Variablen zugeordnet oder direkt für Ausgaben benutzt.

Einfügen eines neuen Datensatzes

Das Einfügen eines Datensatzes in ein dynamisches Array erfolgt analog zum konstanten Array. Es ist dabei darauf zu achten, dass die Nummer, an der die Daten eingefügt werden sollen bereits vorhanden ist.

Syntax und Beispiel:

```
  feld3 [n-1] := wert;
```

Ggf. ist die Länge des dynamischen Arrays über die Funktion length zu bestimmen. Achten Sie darauf, dass die Zählung der Nummern des Arrays bei 0 beginnt. Daher auch im Beispiel die Angabe n-1.

Unter 9.3.1. haben wir einem dynamischen Array die Länge 5 zugewiesen. Das heißt, dass dieses Array 5 Positionen (von 0 ... 4) zur Aufnahme von Daten besitzt. Unter 9.3.2. wurde dann die Länge des dynamischen Arrays bestimmt. Dieser Befehl würde im Beispiel die Länge 5 und damit n := 5 zurückgeben. Soll nun an der letzten Stelle ein neuer Datensatz eingefügt werden, dann ist dies aber wieder die Position 4 also n-1.

Lesen eines Datensatzes

Auch das Lesen eines Datensatzes aus einem dynamischen Array erfolgt analog zum Lesen eines Datensatzes in einem konstanten Array. Es gelten die gleichen Bemerkungen, die bereits beim Einfügen eines Datensatzes angegeben wurden.

Syntax und Beispiel:

```
  wert := feld3 [n-1];
```

Löschen eines Datensatzes

Beim Löschen eines Datensatzes sind wieder zwei Fälle zu unterscheiden.

- Löschen eines Datensatzes am Ende des dynamischen Arrays
- Löschen eines Datensatzes in der Mitte oder am Anfang eines dynamischen Arrays

Löschen eines Datensatzes am Ende des dynamischen Arrays

Hier kann man dem Array am besten mit dem Aufruf von setlength einen neuen Wert für die Länge zuweisen.

Löschen eines Datensatzes in der Mitte oder am Anfang eines dynamischen Arrays

In diesem Fall besteht das Problem, dass die Daten am Ende erhalten bleiben sollen. Am einfachsten kopiert man daher den Wert vom Ende des Arrays an die zu löschende Stelle und löscht dann das Ende des Arrays.

BEISPIEL:

Hierbei soll der Datensatz an der Position 2 gelöscht werden.

```
feld3 [2]  := feld3 [length(feld3)-1];
setlength (feld3, length(feld3)-1);
```

PROGRAMME / 051 / Arrays_Verwenden.DPR

9.5. Stringvariablen als Array

9.5.1. Eigenschaften eines Strings

Strings (also Textvariablen) sind in Delphi (und auch in anderen Programmiersprachen) wie ein dynamisches Array aufgebaut. Das heißt, die Operationen auf ein Array lassen sich auch auf Strings übertragen.
Somit kann man einzelne Zeichen aus einem String auslesen, oder es lassen sich neue Zeichen in einen String schreiben.

9.5.2. String als Array benutzen

Beispiel zum Lesen eines Zeichens aus einem String:

```
{01} procedure TForm1.Button2Click(Sender: TObject);
{02} var text    : string;
{03}     pos     : integer;
{04}     zeichen : string;
{05} begin
{06}   text := edit1.text;
{07}   pos := strtoint (edit2.text);
{08}   zeichen := text [pos];
{09}   edit3.Text := zeichen;
{10} end;
```

Erläuterung zum Quelltext:

- Die Variable Text steht für den String, aus dem ein Zeichen gelesen werden soll
- Pos gibt die Position an, an der gelesen werden soll
- zeichen ist das im Text gelesene Zeichen
- in Zeile 6 und 7 werden der Text und die zu lesende Position bestimmt
- In Zeile 8 wird das Zeichen aus dem Text herausgelesen, dieser Syntax entspricht die Syntax eines Arrays
- In Zeile 9 wird das gelesene Zeichen ausgegeben

Beispiel zum Ersetzen eines Zeichens in einem String:

```
{01}  procedure TForm1.Button1Click(Sender: TObject);
{02}  var alt, neu : string;
{03}      zeichen  : char;
{04}      lesen    : string;
{05}      pos      : integer;
{06}  begin
{07}    alt := edit1.Text;
{08}    pos := strtoint (edit2.text);
{09}    neu := alt;
{10}    lesen := edit4.Text;
{11}    zeichen := lesen [1];
{12}    neu [pos] := zeichen;
{13}    edit3.Text := neu;
{14}  end;
```

Erläuterung zum Quelltext:

- Die Variablen alt und neu stehen für die Strings des alten und des neuen Textes
- Die Variable zeichen gibt den zu ersetzenden String an. Da man stets nur ein einzelnes Zeichen ersetzen kann, muss diese Variable den Typ CHAR besitzen
- Lesen steht für das Zeichen mit dem ersetzt werden soll
- Pos gibt wieder die zu ersetzende Position an
- Zeilen 7-10: Einlesen der Daten aus den Edit-Komponenten
- Zeile 9: Initialisierung der Variable neu
- Zeile 11: Das erste Zeichen aus der Variablen lesen wird bestimmt
- Zeile 12: Dieses Zeichen wird an der entsprechenden Position eingefügt und damit das alte Zeichen an dieser Stelle ersetzt

Erstellen Sie ein Programm, mit dem Sie innerhalb eines Strings Zeichen ersetzen können. Nutzen Sie dazu die beiden Programmsequenzen oben.

PROGRAMME / 052 / STRING_als_ARRAY.DPR

Programmieraufgabe 29

In der Zeit von Computer und E-Mail ist es besonders wichtig Daten vor unberechtigte Benutzer zu schützen. Aber auch in der Geschichte gibt es viele Beispiele für verschlüsselte Nachrichten. An dieser Stelle wollen wir eine der einfachsten Verschlüsselungstechniken implementieren. Die Verschlüsselung beruht auf der Verschiebung der Zeichen innerhalb des zu verschlüsselnden Textes. Diese Technik wird auch als Cäsar-Kodierung bezeichnet.
Jedes zu kodierende Zeichen wird um eine bestimmte Anzahl an Stellen nach rechts verschoben. So wird bei einer Verschiebung um 2 aus a ein c, aus d ein f, aus 2 die 4 usw.
Zur Bearbeitung der Aufgabenstellung benötigen Sie auf Ihrem Formblatt 3 Edit-Komponenten und zwei Button.

Die Oberfläche könnte wie folgt aussehen:

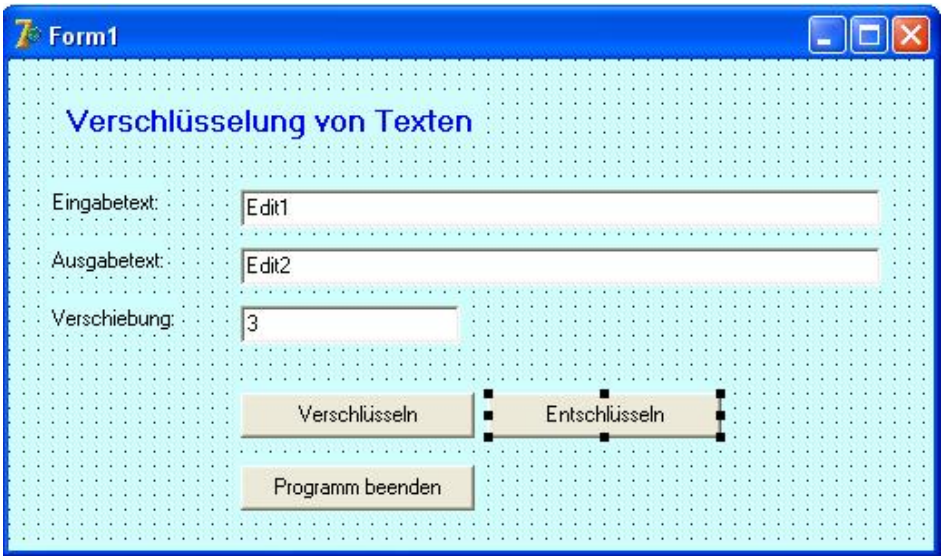


Abb. 92: Mögliche Oberflächengestaltung

Erzeugen Sie die Prozedur zu Button1 (Verschlüsseln) und ergänzen Sie den Quelltext.

```
procedure TForm1.Button1Click(Sender: TObject);
var eingabe, ausgabe  : string;
    laenge, i, versch : integer;
begin
  eingabe := edit1.Text;
  laenge  := length (eingabe);
  versch  := strtoint (edit3.Text);
  ausgabe := '';
  for i := 1 to laenge do
    begin
      inc (eingabe [i], versch);
      ausgabe := ausgabe + eingabe [i];
    end;
  edit2.Text := ausgabe;
end;
```

Erzeugen Sie die Prozedur zu Button2 (Entschlüsseln). Ergänzen Sie den Quelltext selbständig.

PROGRAMME / 053 / Caesar_Kodierung.DPR

9.6. Aufzählkonstanten

9.6.1. Begriff

Es gibt viele Mengen, die nur eine begrenzte Anzahl von Elementen beinhalten. (z.B. die Schüler des Informatikkurses, die Monate des Jahres oder die Farben einer Verkehrsampel). So beinhaltet die Menge M der Farben einer Verkehrsampel nur die Elemente {rot, gelb, grün}. Will man solche Mengen in Programmen nutzen, dann bietet es sich an, hiermit eigene Variablentypen mit genau diesen zugelassenen Werten zu deklarieren.

9.6.2. Aufzählungsvariablen deklarieren

Die Deklaration eigener Variablentypen erfolgt im Variablendeklarationsteil.

Beispiel zur Deklaration einer Aufzählungsvariablen

```
type
    TAmpel = (Rot, Gelb, Gruen);

var
    Form1: TForm1;
    farbe : TAmpel = rot;
```

Erläuterung zur Deklaration:

- Mit type wird dem Compiler mitgeteilt, dass nun eine Typdeklaration erfolgt.
- Eigene Typen sollten möglichst mit einem großen T beginnen.
- Unser deklarierter Typ heißt hier TAmpel.
- Der Typ TAmpel umfasst die drei Elemente rot, gelb, gruen (Umlaute und Sonderzeichen sind nicht erlaubt.

Im Variablendeklarationsteil wurde zusätzlich eine globale Variable mit dem Bezeichner Farbe deklariert. Diese Variable kann nur die im Typ angegebenen Werte annehmen.
Die Deklaration der Variablen muss nach der Deklaration des Typs erfolgen.
Der Variablen Farbe wurde der Anfangswert "rot" zugeordnet (Initialisierung).

9.6.3. Benutzung von Aufzählvariablen

Die nachfolgende Prozedur gibt ein Beispiel für die Benutzung einer Aufzählvariablen an.

```
  procedure TForm1.Button1Click(Sender: TObject);
  begin
    case farbe of
      rot   : form1.Color := clred;
      gelb  : form1.Color := clyellow;
      gruen : form1.Color := cllime;
    end;
    inc (farbe);
  end;
```

Erläuterung zum Quelltext:

Mit case erfolgt eine Auswahl über dem jeweils der Variablen Farbe zugeordneten Wert
Die Anweisung inc (farbe) gibt dem Compiler die Meldung, den nächsten Wert der Variablen aufzurufen.

PROGRAMME / 054 / Mengen.DPR

9.7. Variablentypen mit Arrays deklarieren

9.7.1. Vorbemerkungen

Bisher haben wir immer Arrays als eine einzelne Variable deklariert. In manchen Fällen benötigt man aber auch das Arrays der gleichen Größe und des gleichen Inhaltstyps für mehrere einzelne Variable. Dann bietet es sich an, für diese Arrays eigene Variablen zu deklarieren.

BEISPIEL:

Sie wollen Aufgaben aus dem Bereich der Analytischen Geometrie implementieren. Dazu benötigen Sie immer wieder Vektoren mit den entsprechenden Koordinaten x1, x2 und x3.
Diese Vektoren können im Vorfeld als eigener Typ deklariert werden. Es können dann die entsprechenden Variablen kürzer dargestellt werden.

9.7.2. Deklaration von Arraytypen

Arraytypen werden ähnlich deklariert wie die uns bereits bekannten Aufzählungstypen.

Beispiel zur Deklaration eines Arraytyps

```
type
  TVektor = array [1..3] of real;
```

Erläuterung zur Deklaration:

- Die Deklaration erfolgt wieder mit type
- Der Variablentyp hat den Namen TVektor. Der Inhalt dieses Variablentyps wird wieder mit = abgetrennt.
- Die Deklaration des Inhalts erfolgt analog zur Deklaration einer Arrayvariablen
- Die Deklaration muss wieder vor der ersten Benutzung des Typs erfolgen.

9.7.3. Benutzung der Arraytypen

Mit dem Arraytyp wurde wieder ein eigenständiger Variablentyp deklariert. Um diesen benutzen zu können, müssen nun auch noch die entsprechenden Variablen dieses Typs deklariert werden. Dies kann wieder global oder lokal geschehen.

Beispiel zur Deklaration einer lokalen Arrayvariablen

```
procedure TForm1.Button1Click(Sender: TObject);
var vek1 : TVektor;
begin
  ...
```

Beispiel zur Deklaration eines eigenen Variablentyps und dessen Verwendung. Im Beispiel werden zwei Vektoren addiert und es wird deren Skalarprodukt ermittelt.

PROGRAMME / 055 / Vektorrechnung.DPR

9.8. Variablentypen mit Record definieren

9.8.1. Begriff

In Arrays lassen sich immer nur Daten des gleichen Typs verwalten. Für viele Anwendungen ist es aber sinnvoll und notwendig Daten verschiedener Typen gleichzeitig und zusammengehörig zu bearbeiten und zu verwalten. Um dies zu erreichen kann man neue Variablentypen deklarieren, die Daten verschiedener Typen gleichzeitig verwalten. Diese Typen werden auch Recordtypen genannt.

BEISPIEL:

Um eine Adressverwaltung zu programmieren, benötigt man Daten über den Namen (String), Wohnort (String), Postleitzahl (Integer) und Geburtsdatum (TDatetime).

9.8.2. Deklaration eines Recordtyps

Wie oben bereits erwähnt werden in einem Recordtyp Variablen unterschiedlichen Typs miteinander kombiniert. Dazu müssen die einzelnen Inhalte des Records angegeben werden. Außerdem muss dieser Typ auch noch einen Namen erhalten. Im folgenden Beispiel ist ein Recordtyp zur Adressverwaltung dargestellt.

```
type
  TAdresse = record
    name       : string [20];
    vorname    : string [15];
    Wohnort    : string [15];
    PLZ        : Integer;
    Geburtsdat : tdatetime;
  end;
```

Erläuterung zur Deklaration:

Die Deklaration des Recordtyps beginnt mit dem Namen des neuen Variablentyps gefolgt von einem Gleichheitszeichen und dem Begriff `record`. Danach folgt eine Liste der in diesem Typ verwendeten Inhalte mit Angabe des verwendeten Variablentyps (es können auch Aufzählvariablen bzw. Arrayvariablen benutzt werden). Die Deklaration von Stringvariablen in dieser Liste beinhaltet immer die Länge der vorgesehenen Strings. Die Liste muss mit `end` beendet werden.

Die so deklarierten Variablen können z.B. wieder in einem Array verwaltet werden.

BEISPIEL:

```
var
  Form1: TForm1;
  adressen : array [1..5] of TAdresse;
```

9.8.3. Benutzung von Recordtypen

Darstellung des oben deklarierten Arrays adressen (Beispiel):

adressen : array [1..5] of TAdresse;

1	2	3	4	5
Müller	Mustermann	Meyer	Hempel	Leichtsinn
Lieschen	Max	Herbert	Susi	Eberhard
Haldensleben	Wedringen	Bebertal	Magdeburg	Berlin
12345	23456	34567	45678	56789
01.01.1980	02.02.1990	03.03.1995	04.04.1996	05.05.2000

Abb. 93: Array mit Recordvariable

Daten werden in dieses Array (fast) genauso eingeschrieben, gelesen und verändert, wie wir es bereits von einfachen Arrays kennen gelernt haben. Zusätzlich muss aber nun noch der entsprechende Inhaltsbereich des Recordtyps angesprochen werden. Dabei unterstützt uns die Delphi-Hilfe. Im folgenden Beispiel wird der Datensatz des ersten Arrayfeldes angelegt.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  adressen [1].name := 'Müller';
  adressen [1].vorname := 'Lieschen';
  adressen [1].Wohnort := 'Haldensleben';
  adressen [1].PLZ := 12345;
  adressen [1].Geburtsdat := strtodatetime ('01.01.1980');
end;
```

Nach der Angabe des Bezeichners des Arrays und der Position innerhalb des Arrays folgt nach einem Punkt der Inhalt des Recordtyps und danach werden mit einer normalen Ergibt-Anweisung die einzuschreibenden Inhalte angegeben.

Das Lesen der Daten erfolgt analog.

PROGRAMME / 056 / Adressverzeichnis.DPR

Fragen und Aufgaben
9. Strukturierte Datentypen

Grundlagen

1. Erläutern Sie die Unterschiede zwischen statischen und dynamischen Arrays.
2. Worauf ist zu achten, wenn Sie mehrdimensionale Arrays verwenden?
3. Nennen Sie Beispiele für die Verwendung eines Array-Typs.
4. Nennen Sie Beispiele für die Verwendung eines Record-Typs.
5. Ein Array-Typ kann auch Variablen eines Record-Typs enthalten. Nennen Sie ein Beispiel.

Programmieraufgaben

6. Erstellen Sie ein Delphi-Programm zur Auswertung einer Messwertreihe, bei dem für einen Körper mit feststehendem Volumen insgesamt fünfmal die Masse gemessen wurde. Berechnen Sie dabei den Durchschnittswert der Massen und bestimmen Sie daraus die Dichte des Körpers. Die Messwerte sind dabei in einem konstanten Array zu verwalten.

7. Erstellen Sie ein Delphi-Programm mit dem der Durchschnitt einer Klausur oder eines Tests für einen Kurs berechnet werden kann. Es sind dabei die Anzahlen der einzelnen Notenpunkte einzugeben. Der Durchschnitt soll berechnet und ausgegeben werden.

8. Erstellen Sie eine Delphi-Anwendung zur Lösung eines zweidimensionalen Gleichungssystems mit dem Gauß'schen Algorithmus. Die Eingabedaten sind dabei in einem zweidimensionalen Array zu verwalten.

9. Lösungen/009/031/dame.dpr
   Erstellen Sie ein Programm für das Damespiel mit zwei Spielern. Verwalten Sie dabei die Züge jedes Spielers in einem zweidimensionalen Array.
   Zusatzaufgabe:
   Das Programm soll selbständig auf falsche Züge aufmerksam machen.

10. Erstellen Sie ein Delphi-Programm für eine einfache Verschlüsselung von Texten. Dabei soll ein Eingabetext so umgewandelt werden, dass jeder Buchstabe durch seinen entsprechenden Nachfolger ersetzt wird (z.B. aus a wird b aus q wird r u.s.w.).
    Zusatzaufgabe:
    Erweitern Sie das Programm so, dass die Verschiebung der Zeichen vom Nutzer festgelegt werden kann.

11. Lösungen/009/032/project1.dpr
    Erstellen Sie ein Delphi-Programm zur Simulation eines Würfelspiels. Dabei sind eine vom Nutzer festzulegende Anzahl von Würfen durchzuführen. Die Daten sind in einem dynamischen Array zu verwalten. Über ein konstantes Array sind nach Abschluss der Würfe die Anzahlen der einzelnen Werte (1 .. 6) auszuzählen und auszugeben.

12. Lösungen/009/033/Project1.dpr
    Erstellen Sie ein Delphi-Programm zur Vektorrechnung. Für die Vektoren ist dabei ein eigener Datentyp zu implementieren. Es soll dabei die Addition, das Skalarprodukt und das Vektorprodukt zweier Vektoren berechnet werden.
    Zusatzaufgabe
    Es ist zu überprüfen, ob zwei eingegebene Vektoren linear unabhängig voneinander sind.

13. Lösungen/009/034/project1.dpr
    Erstellen Sie ein Delphi-Programm zur Verwaltung eines Sportvereins. Dabei sollen die folgenden Daten in einem dynamischen Array abgelegt werden:
        Name
        Vorname
        Mitgliedsnummer
        Geburtsdatum
        Abteilung (Sportart)
        Beitragssatz
        Funktion
    Innerhalb des Programms sollen die nachfolgenden Aufgaben erfüllt werden:
        Ein- und Ausgabe der Mitglieder
        Blättern innerhalb der Mitgliederliste
        Ändern von aktuell angezeigten Mitgliedern
        Löschen von Mitgliedern
    Die Ein- und Ausgabe von Daten soll über entsprechende Edit-Komponenten erfolgen.

14. An einem Triathlon-Wettbewerb nehmen insgesamt 40 Sportler teil. Erstellen Sie ein Delphi-Programm zur Vorbereitung und Auswertung dieses Wettbewerbs. Dazu sind die nachfolgenden Daten aufzunehmen:
        Name
        Vorname
        Startnummer
        Geburtsdatum
        Zeit Schwimmen
        Zeit Radfahren
        Zeit Laufen
        Gesamtzeit
    Innerhalb des Programms sollen die nachfolgenden Aufgaben erfüllt werden:
        Ein- und Ausgabe der Teilnehmer
        Blättern innerhalb der Teilnehmerliste
        Eingabe der Zeiten für die einzelnen Positionen soll nachträglich möglich sein
        Berechnung der Gesamtzeit und Ablage der Daten im Array
    Die Ablage der Teilnehmer soll dabei in einem konstanten Array erfolgen. Die Ein- und Ausgabe von Daten soll über entsprechende Edit-Komponenten erfolgen. Die besten drei Teilnehmer sollen dabei in 3 gesonderten Label angezeigt werden können.

10. Nutzung der Dialog-komponenten

10.1. Erstellen eines Menüs

10.1.1. MainMenu und PopupMenu

Um Menüs für ein Programm erstellen zu können, benötigt man zunächst die Komponenten MainMenu bzw. PopupMenu. Die entsprechenden Komponenten selbst sind zur Laufzeit nicht sichtbar. Sie beinhalten lediglich die Daten für die Menüerstellung. Zum Entwerfen eines Menüs klicken Sie doppelt auf die entsprechende Komponente. Es öffnet sich ein Fenster zum Erstellen des Menüs.

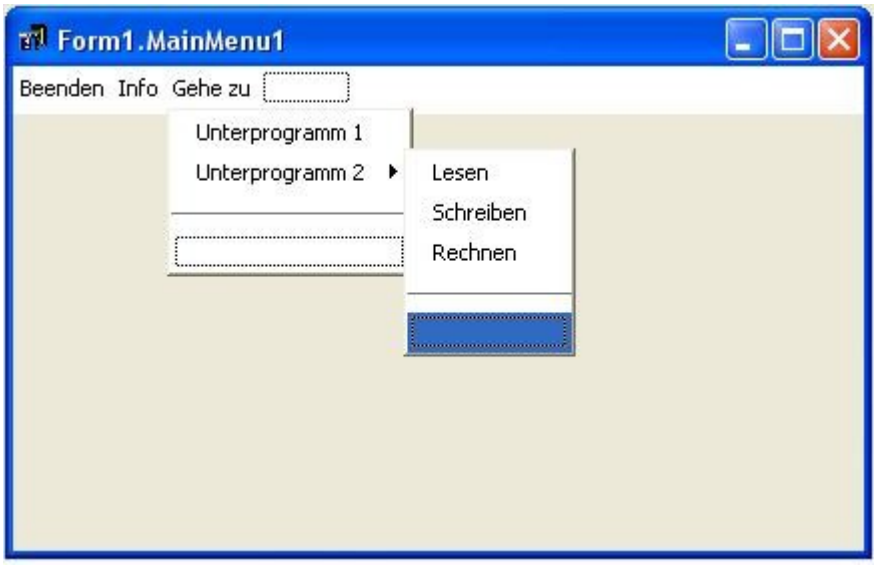


Abb. 94: Ansicht des Menü-Editors

Dieses Menü kann nun beliebig mit Texten versehen werden.

Die nachfolgenden Punkte sind dabei zu beachten:

- Soll ein Untermenü zu einem Menüpunkt erzeugt werden, dann markiert man zunächst den entsprechenden Menüpunkt, klickt dann auf die rechte Maustaste und wählt den Punkt Untermenü erstellen aus.
- Soll für einen Menüpunkt eine Tastenkombination ausgewählt werden, so wählt man im OI-ShortCut die gewünschte Tastenkombination aus.
- Soll im Menü eine waagerechte Trennlinie erscheinen, dann wird an der entsprechenden Stelle ein Bindestrich eingegeben.

10.1.2. Implementieren von Prozeduren

Das Erzeugen von Prozeduren zu einem Menü erfolgt denkbar einfach:

MainMenu: Klicken Sie einfach auf die zu implementierenden Menüpunkte im geschlossenen Menüdesigner.
PopupMenü: Klicken Sie doppelt auf die zu implementierenden Menüpunkte im geöffneten Menüdesigner.

Die Prozeduren werden dann auf gleiche Weise wie sonst implementiert.

10.1.3. Benutzen von Standardmenüs

Delphi bietet Ihnen eine Reihe von Standardmenüs an. Diese Menüs können beliebig eingefügt und bearbeitet werden.
Dazu klickt man mit der rechten Maustaste innerhalb des geöffneten Menüdesigners und wählt den Punkt "Aus Vorlage verwenden" aus. Das folgende Beispiel zeigt ein vollständiges Menü:

Abb. 95: Standardmenü

10.2. Laden und Speichern von Daten

10.2.1. Speichern von Daten

Zu einem Programm gehört im Allgemeinen auch die Möglichkeit Daten auf externen Datenträgern (Festplatte, CD-ROM, Diskette...) zu speichern und von dort auch wieder zu laden. Der spätere Nutzer möchte dazu die ihm aus vielen anderen Anwendungen bekannten Öffnen- und Speichern-Dialoge nutzen. Um dies zu erreichen nutzen wir die entsprechenden OpenDialog und SaveDialog Komponenten die uns Delphi anbietet. Damit erreichen wir ohne großen Programmieraufwand genau diese bekannten Windows-Dialoge. Zunächst wollen wir uns dazu den entsprechenden Save-Dialog anschauen und damit einen kurzen Text und Daten aus einem konstanten Array speichern.

Speichern eines Textes

Vgl. Programmieraufgabe 30.

Speichern von Daten aus einem Array

Das Speichern von Daten ist etwas anspruchsvoller als das Speichern von reinen Textdateien. Es müssen dabei alle Datensätze einzeln gespeichert werden. Eine entsprechende Prozedur sieht dann folgendermaßen aus:

```
procedure TForm1.Button_speichernClick(Sender: TObject);
 var datname : string;                    // Dateiname
     i       : integer;
     daten   : array of real;             // zu speicherndes Array
     datei   : file of real;              // zu speichernde Datei
 begin
   if savedialog1.execute then            // öffnet das Dialogfeld
     begin
       datname := SaveDialog1.FileName;   // Abfrage des Dateinamens
       assignfile (datei, datname);       // der Datei Name zuweisen
       rewrite (datei);                   // Datei erstellen und öffnen
       for i := 1 to length (daten) do
         write (datei, daten [i-1]);      // schreiben der Daten in Datei
       closefile (datei);                 // Datei wird geschlossen
     end;
 end;
```

Achten Sie wieder darauf, dass ein dynamisches Array mit der Zählvariablen 0 beginnt. Für ein konstantes Array kann die Anzahl der Arrayinhalte natürlich direkt angegeben werden. Die Einstellungen der Komponente SaveDialog werden genauso vorgenommen, wie bei einer Textdatei.

Programmieraufgabe 30

Platzieren Sie dazu die Komponente SaveDialog aus der Komponentenliste Dialoge auf der Oberfläche. Diese Komponente selbst ist zur Laufzeit wieder nicht sichtbar. Legen sie Zusätzlich eine Memokomponente auf die Oberfläche und geben Sie in diese einen mehrzeiligen Text ein. Weiterhin benötigen wir einen Button mit der Aufschrift "Speichern". Markieren Sie anschließend die Komponente SaveDialog und nehmen Sie die nachfolgenden Einstellungen im Objektinspektor vor.

Eigenschaft	Einstellung	Zweck
Filter	Es öffnet sich ein Dialogfeld, wie in der Abbildung unten, geben Sie die entsprechenden Daten ein.	Mit dem Filter geben Sie an, welche Dateierweiterung der Nutzer verwenden muss. Diese werden im späteren Speichern-Dialog unter Dateityp angezeigt.
FileName	Geben Sie hier *.txt ein.	Legt eine Vorgabe für die Dateierweiterung fest.

Abb. 96: Einstellen des Filters

Kommen wir nun zum Speichern unseres Textes. Erzeugen Sie dazu die OnClick Prozedur des Buttons. Geben Sie den nachfolgenden Quelltext ein:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  if Savedialog1.Execute then                    // Öffnet das Dialogfeld
    Memo1.Lines.SaveToFile(SaveDialog1.filename); // Speichern der Daten
end;
```

Der so gespeicherte Text kann jetzt mit einem einfachen Textverarbeitungsprogramm (z.B. Editor) geöffnet werden. Probieren Sie es aus.

10.2.2. Laden von Daten

Laden eines Textes

Vgl. Programmieraufgabe 31.

Laden von Daten in ein Array

Für das Laden von Daten sei die nachfolgende Prozedur angegeben. Die Bemerkungen zum Speichern gelten analog.

```
procedure TForm1.Button_ladenClick(Sender: TObject);
var datname : string;                          // Dateiname
    daten   : array of real;                   // Array für die Daten
    datei   : file of real;                    // zu öffnende Datei
    i       : integer;
    wert    : real;                            // Wert der gelesenen Daten
begin
  setlength (daten,0);                         // Array Länge 0 zuweisen
  if OpenDialog1.Execute then                  // öffnet das Dialogfeld
    begin
      datname := OpenDialog1.FileName;         // Abfrage des Dateinamens
      assignfile (datei, datname);             // der Datei Name zuweisen
      reset (datei);                           // Datei öffnen
      if ioresult <> 0 then                    // wahr, wenn Daten vorhanden
        rewrite (datei);
      for i := 1 to filesize (datei) do        // bis Dateilänge
        begin
          read (datei, wert);                  // Lesen der Daten
          setlength (daten, length (daten)+1);  // gelesene Daten in das
          daten [i-1] := wert;                  // Array übernehmen
        end;
      closefile (datei);                       // Datei schließen
    end;
 end;
```

PROGRAMME / 057 / Laden_und_Speichern_1.DPR

Programmieraufgabe 31

Natürlich wollen wir unsere gespeicherten Daten auch wieder von der Festplatte laden. Öffnen Sie dazu das Programm aus Programmieraufgabe 30. Legen Sie die Komponente OpenDialog auf die Oberfläche des Formblattes und nehmen die dazu notwendigen Einstellungen im OI vor. Weiterhin benötigen wir einen zweiten Button mit der Aufschrift "Öffnen". Erzeugen Sie wieder die OnClick Prozedur des Buttons und geben Sie den nachfolgenden Quelltext ein.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  if OpenDialog1.Execute then                    // öffnet das Dialogfeld
    Memo1.Lines.LoadFromFile(OpenDialog1.Filename);    // Laden der Daten
end;
```

Mit dieser Prozedur lassen sich auch anderen Dateien im Format *.txt (z.B. mit Editor erzeugte Dateien) öffnen. Probieren Sie es aus.

PROGRAMME / 058 / Laden_und_Speichern_2.DPR

10.3. Sonstige Dialogkomponenten

10.3.1. Fontdialog

Der Fontdialog ruft den uns bekannten Dialog zur Formatierung von Text auf. Mit ihm kann die Bildschirmanzeige der Daten vom Nutzer nach eigenen Wünschen angepasst werden.
Um zum Beispiel die Anzeige der Daten in einer Memokomponente zu gestalten geht man folgendermaßen vor:

Platzieren Sie die Komponente FontDialog auf der Oberfläche des Formulars. Platzieren Sie zusätzlich eine Komponente Memo auf dem Formular (es kann natürlich auch eine beliebige andere Komponente, die entsprechend formatiert werden kann benutzt werden). Geben Sie den nachfolgenden Quelltext ein:

```
  procedure TForm1.Button1Click(Sender: TObject);
  begin
    if fontdialog1.Execute then
      memo1.Font := fontdialog1.Font;
  end;
```

Es ist auch möglich nur bestimmte Eigenschaften (z.B. Schrifttyp oder Schriftfarbe) für die Formatierung zuzulassen. Dies könnte dann folgendermaßen aussehen.

```
  memo1.Font.Color := fontdialog1.Font.Color;
```

10.3.2. Colordialog

Mit dem Colordialog können Farben für Hintergrund und Schrift formatiert werden. Dazu benötigen Sie die Komponente ColorDialog. Der Colordialog funktioniert ähnlich dem Fontdialog, was die nachfolgenden Beispiele veranschaulichen sollen.

Beispiel 1: Ändern der Hintergrundfarbe einer Komponente

```
  if colordialog1.Execute then
    memo1.Color := colordialog1.Color;
```

Beispiel 2: Ändern der Schriftfarbe einer Komponente

```
  if colordialog1.Execute then
    memo1.Font.Color := colordialog1.Color;
```

10.3.3. Drucken von Daten

Natürlich möchte man in vielen Fällen auch die Möglichkeit haben, Ein- und Ausgabedaten zu drucken. Hierfür steht uns die Komponente PrintDialog zur Verfügung. Mit dieser Komponente werden entsprechende Daten an einen Drucker ausgegeben. Für den Printdialog wird wieder die entsprechende Komponente auf der Oberfläche benötigt. Außerdem muss die Unit "Printers" in der Uses-Liste der Unit aufgenommen werden. Im nachfolgenden Beispiel wird der Inhalt der Memokomponente an einen installierten Drucker ausgegeben.

```
{01} procedure TForm1.Button4Click(Sender: TObject);
{02} var pos    : integer;
{03}     n, i   : integer;
{04} begin
{05}   if printdialog1.Execute then
{06}     begin
{07}       pos := 100;
{08}       n   := memo1.Lines.Count;
{09}       with printer do
{10}         begin
{11}           BeginDoc;
{12}             for i := 0 to n do
{13}               canvas.TextOut(100,pos + i*50,
                                              memo1.Lines.Strings[i]);
{14}           EndDoc;
{15}         end;
{16}     end;
{17} end;
```

Erläuterungen zum Quelltext

02	Die Ausgabe erfolgt über die Zeichenoperationen von TCanvas. Daher muss auch die Position für die Textausgabe festgelegt werden. "pos" gibt die y-Ausrichtung an.
03	n gibt die Anzahl der Zeilen in der Memo-Komponente an
05	Hier wird das Druckerdialogfeld geöffnet
07	Der obere Rand der Seite wird durch Initialisierung von pos festgelegt.
08	Bestimmen der Anzahl der Zeilen in der Memo-Komponente.
09	Printer muss verwendet werden um die grafische Ausgabe an den Drucker und nicht an den Bildschirm zu übergeben.
11	Ab hier werden Daten an den Drucker gesendet.
13	Ausgabe der einzelnen Textzeilen über Canvas.
14	Ende der Datenübertragung

Probleme bei der Drucküübertragung

- Das Programm gibt auf diese Weise nur die erste Druckseite aus. Bei längeren Texten kann dies dazu führen, dass nicht der gesamte Druckbereich abgearbeitet wird.
- Soll der Druckauftrag über mehrere Blätter erfolgen muss mit NewPage diese neue Seite in Auftrag gegeben werden.
- Mit PageHeight kann die Höhe der Druckseite bestimmt werden (in Pixel).
- Mit PageWidth kann die Breite der Druckseite bestimmt werden (in Pixel).
- Diese Angaben benötigt man, um festzustellen, wann der Seitenumbruch erfolgen soll.

PROGRAMME / 059 / Weitere_Dialoge.DPR

Fragen und Aufgaben
10. Dialogkomponenten

Nutzen Sie für die Bearbeitung der nachfolgenden Aufgaben Ihre Programme aus vorherigen Aufgabenstellungen. Die zusätzlichen Prozeduren sollen dabei über ein Menü aufgerufen werden.

1. Erstellen Sie eine Delphi-Anwendung zur Lösung eines zweidimensionalen Gleichungssystems mit dem Gauß'schen Algorithmus. Die Eingabedaten sind dabei in einem zweidimensionalen Array zu verwalten.
   Ergänzen Sie das Programm wie folgt:
   Es soll die Möglichkeit zum Drucken des Gleichungssystems und der Lösung geschaffen werden.

2. Erstellen Sie ein Delphi-Programm für eine einfache Verschlüsselung von Texten. Dabei soll ein Eingabetext so umgewandelt werden, dass jeder Buchstabe durch seinen entsprechenden Nachfolger ersetzt wird (z.B. aus a wird b aus q wird r u.s.w.).
   Ergänzen Sie das Programm wie folgt:
   Der Original- und der verschlüsselte Text sind zu speichern und zu laden
   Der verschlüsselte Text soll gedruckt werden.

3. Erstellen Sie ein Delphi-Programm zur Verwaltung eines Sportvereins. Dabei sollen die folgenden Daten in einem dynamischen Array abgelegt werden:
   - Name
   - Vorname
   - Mitgliedsnummer
   - Geburtsdatum
   - Abteilung (Sportart)
   - Beitragssatz
   - Funktion

   Innerhalb des Programms sollen die nachfolgenden Aufgaben erfüllt werden:
   - Ein- und Ausgabe der Mitglieder
   - Blättern innerhalb der Mitgliederliste
   - Ändern von aktuell angezeigten Mitgliedern
   - Löschen von Mitgliedern

   Ergänzen Sie das Programm wie folgt:
   Die Daten der Mitglieder sind zu speichern und zu laden
   Drucken von Mitgliederlisten

4. An einem Triathlon-Wettbewerb nehmen insgesamt 40 Sportler teil. Erstellen Sie ein Delphi-Programm zur Vorbereitung und Auswertung dieses Wettbewerbs. Dazu sind die nachfolgenden Daten aufzunehmen:
   - Name
   - Vorname
   - Startnummer
   - Geburtsdatum
   - Zeit Schwimmen
   - Zeit Radfahren
   - Zeit Laufen
   - Gesamtzeit

   Innerhalb des Programms sollen die nachfolgenden Aufgaben erfüllt werden:
   - Ein- und Ausgabe der Teilnehmer
   - Blättern innerhalb der Teilnehmerliste
   - Eingabe der Zeiten für die einzelnen Positionen soll nachträglich möglich sein
   - Berechnung der Gesamtzeit und Ablage der Daten im Array

   Die Ablage der Teilnehmer soll dabei in einem konstanten Array erfolgen.
   Die Ein- und Ausgabe von Daten soll über entsprechende Edit-Komponenten erfolgen. Die besten drei Teilnehmer sollen dabei in 3 gesonderten Label angezeigt werden können.
   Ergänzen Sie das Programm wie folgt:
   - Drucken von Startlisten.
   - Drucken von Urkunden für die ersten drei Plätze
   - Speichern und Laden der Daten.

5. Entwickeln Sie ein Delphi-Programm zur Verwaltung einer Bibliothek. Es sind die folgenden Daten über einen Record-Typ in einem dynamischen Array zu verwalten:
   Titel, Autor, Buchnummer, Jahr. Es soll weiterhin die Möglichkeit bestehen die Bücher nach Autor bzw. nach Nummer zu sortieren.
   Ergänzen Sie das Programm wie folgt:
   Drucken der Bücherlisten
   Speichern und Laden der Listen

6. Entwickeln Sie ein Delphi-Programm zur Auswertung eines Laufwettbewerbs (Weitsprungwettbewerbs). Die Daten sollen dabei in einem Array mit einem Recordtyp verwaltet werden. Der Recordtyp soll dabei die folgenden Daten erfassen: Name, Vorname, Startnummer, Leistung. An den Wettbewerben nehmen jeweils 20 Sportler teil. Anzuzeigen ist jeweils eine Liste der Ergebnisse sortiert nach Platz. Nutzen Sie dafür eine Komponente StringGrid.
   Ergänzen Sie das Programm wie folgt:
   Drucken von Startlisten.
   Drucken von Urkunden für die ersten drei Plätze
   Seichern und Laden der Daten.

11. Fehlerbehandlung

11.1. Fehler mit IF...THEN abfangen

In vorhergehenden Kapiteln haben Sie Fehler bereits oft mit IF..THEN..ELSE abgefangen. Erinnert sei hier noch einmal an die Division durch Null, bei der durch eine vorherige Abfrage des Divisors eine entsprechende Fehlermeldung ausgeschlossen wird. Das Abfangen von Fehlermeldungen ist mit dieser Anweisung aber nur sehr beschränkt ausführbar. Sie beschränken sich im Wesentlichen Programmieraufgaben, die nicht vom Nutzer beeinflusst werden. Beispiele zur Anwendung sind:

Abfangen der Division durch 0
Fehlerhafte Eingaben und Berechnungen mit arcsin und arccos u.ä.
Bereichsüberschreitungen bei der Eingabe von Variablen

11.2. Exceptionbehandlung

11.2.1. Begriff

In den bisherigen Programmen sind oft Fehlermeldungen des Compilers während der Ausführung aufgetreten. Z.B. dann, wenn der Text in einer Edit-Komponente keine Zahl darstellt, obwohl diese verlangt wurde. Eine solche Fehlermeldung sieht dann oft folgendermaßen aus:

Abb. 97: Beispiel für eine Fehlermeldung

Diese Fehlermeldungen nennt man auch EXCEPTION. Sie treten immer dann auf, wenn während der Programmabarbeitung ein Fehler gefunden wird.
Diese Exceptions können sehr unterschiedlicher Art sein. Im Folgenden wollen wir uns etwas näher mit diesen Fehlern beschäftigen und dabei lernen, wie diese Fehler während der Programmausführung abgefangen werden können.

11.2.2. Beispiele für Exceptionmeldungen

Exceptionmeldung	möglicher Grund
EAccessViolation	ungültige Speicherzugriffe (z.B. bei der Arbeit mit Listen)
EConvertError	Tritt z.B. bei der Konvertierung von Strings in Zahlen auf.
EDivByZero	Division einer Ganzzahl durch Null.
EInvalidPointer	Tritt bei ungültigen Zeigeroperationen auf.
EMathError	Mathematische Fehler (z.B. Wurzel aus negativer Zahl)
EZeroDivide	Division einer Gleitkommazahl durch Null.

Es gibt noch weitaus mehr Fehlermeldungen. Die hier aufgelisteten sind die für uns zunächst wichtigsten Fehlermeldungen. Mehr erfahren Sie in der Delphi-Hilfe.

11.2.3. Die Anweisung try ... except

Um z.B. Eingabefehler mit eigenen Fehlermeldungen abzufangen benutzt man die Anweisungen try ... except. Im folgenden Beispiel soll aus einer Edit-Komponente eine Integer-Zahl gelesen werden. Im Falle einer falschen Eingabe soll der Nutzer über eine MessageBox auf seinen Fehler aufmerksam gemacht werden.

```
procedure TForm1.Button1Click (Sender : TObject);
var a : integer;
begin
  try                              // hier beginnt die Excptionbehandlung
    a := strtoint (edit1.Text);
  except                           // wird ausgeführt, wenn Fehler auftritt
    on EConvertError do            // bei entsprechendem Fehler
      application.MessageBox('Falsche Eingabe', 'FEHLERMELDUNG', MB_OK );
  end;                             // beendet try
  ...
end;
```

Die allgemeine Syntax dieser Fehlerbehandlungen lautet:

```
try
  Anweisung(en)
except
  on Fehlermeldung 1 do
    begin
      ...
    end;
  on Fehlermeldung 2 do
    begin
      ...
    end;
  ...
  on Fehlermeldung n do
    begin
      ...
    end;
else
  begin
    ...
  end;
end;
```

Es ist also durchaus möglich mehrere Fehler gleichzeitig abzufangen. Weiterhin kann man im ELSE-Zweig alle noch nicht berücksichtigten Fehler unterbinden.

Wichtiger Hinweis:

Sollte die Exception-Meldung beim Programmablauf doch noch auftreten, dann müssen Sie die Debugger-Einstellungen verändern. Gehen Sie dazu folgendermaßen vor:

- Wählen Sie im Menü TOOLS und anschließend Debugger-Optionen
- Wählen Sie das Registerblatt Sprach-Exceptions
- Entfernen Sie das Häkchen in "Bei Delphi-Exceptions stoppen" und klicken Sie auf OK.

PROGRAMME / 060 / Exceptionbehandlung.DPR bzw. PROJECT1.EXE

Fragen und Aufgaben
11. Fehlerbehandlung

Überarbeiten Sie die Programme aus Kapitel 10, indem Sie mögliche Fehler mit try-except abfangen.

12. Klassen und Objekte

12.1. Klassen und Objekte

12.1.1. Begriffe

Bei der Programmiersprache Delphi handelt es sich um eine so genannte Objektorientierte Programmiersprache (OOP). Grundlage dieser Programmierung bilden dabei Objekte und Klassen. Viele dieser Objekte und Klassen haben Sie während der Programmierung bereits benutzt, denn alle Komponenten (also das Formblatt, Button, Label usw.) sind Objekte die jeweils zu einer Klasse gehören.

Was aber sind Objekte? Und was versteht man unter Klassen?

Vergleichen wir diese Begriffe zunächst einmal mit einem Beispiel aus dem nichtinformatischen Bereich. Jeder von Ihnen kann mit dem Begriff "HUND" etwas anfangen. Dabei ist dieser Begriff aber zunächst sehr allgemein gehalten. Der eine denkt an einen Schäferhund, der andere an einen Dackel. Trotzdem haben alle Hunde Gemeinsamkeiten, die sie zur Klasse HUND dazugehören lassen. Zu diesen Gemeinsamkeiten gehören:

- Eigenschaften
- Ereignisse
- Methoden

BEISPIEL:

Unser HUND hat z.B. die Eigenschaften: Fellfarbe, Größe, Alter, Rasse. Er reagiert auf die Ereignisse "Platz", "Sitz" und hat die Methoden: Schlafen, Fressen, Bellen. Da der Begriff HUND aber immer noch allgemein gehalten ist, wurden diesen E, E, M noch keine konkreten Werte zugeordnet. Im Folgenden werden diesen E, E, M konkrete Werte zugeordnet:

Eigenschaften	**Wert**
Fellfarbe	schwarz
Größe	50 cm
Alter	3 Jahre
Rasse	Mischling
Ereignis	**Reaktion des Hundes**
Platz	Hund legt sich hin
Sitz	Hund läuft weg
Methode	**das macht der Hund**
Schlafen	Hund schnarcht
Fressen	Hund mag nur Trockenfutter
Bellen	Hund bellt böse

Objekte und Klassen in der Programmierung

In der objektorientierten Programmierung sind z.B. alle Komponenten die Sie über die Komponentenliste auswählen können zunächst Klassen.
Die Komponenten die bereits auf dem Formblatt liegen sind ebenso wie das Formblatt selbst Objekte.
Diese Objekte haben viele voreingestellte Eigenschaften, die Sie über den Objektinspektor verändern können.

Beispiel:

Betrachten wir dazu eine Memokomponente auf dem Formblatt. Dieser Komponente können die verschiedensten Eigenschaften mit Hilfe des Objektinspektors zugeordnet werden. Außerdem können mit Hilfe der Ereignisseite des Objektinspektors verschiedene Ereignisse implementiert werden (z.B. OnClick, OnChange).

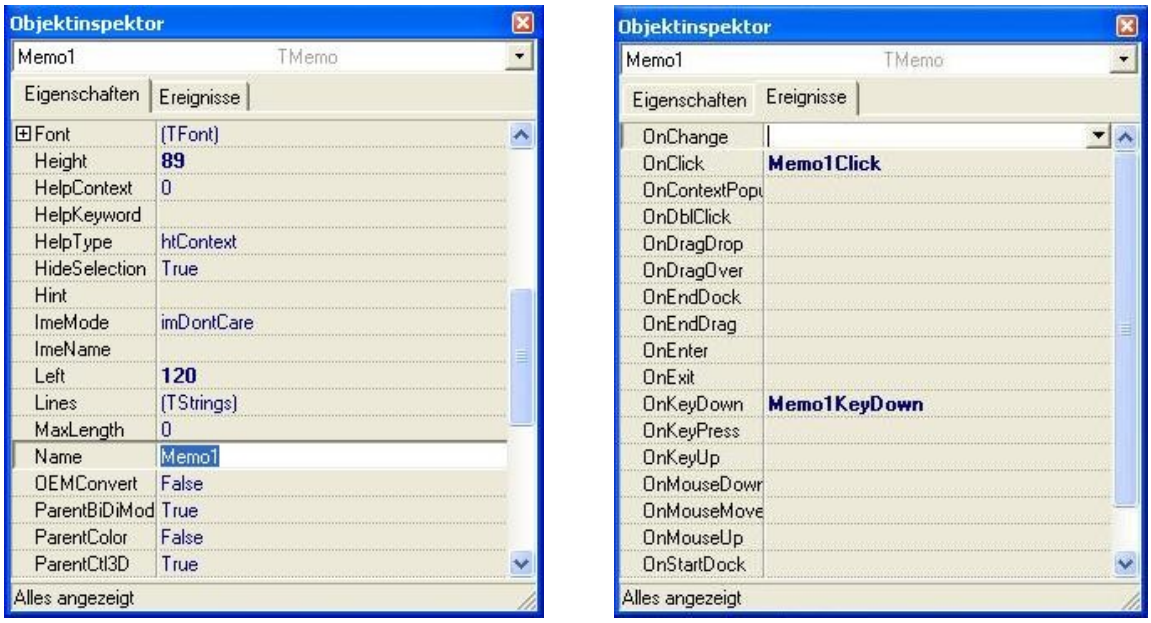


Abb. 98, 99: Objektinspektor mit Eigenschaften und Ereignissen

Die Methoden der Memokomponente sind fest vorgegeben. Z.B. kann im Programm die Methode CLEAR (memo1.clear) aufgerufen werden. Damit wird die Memokomponente angewiesen den gesamten Inhalt (Text) zu löschen. Allen Klassen und Objekten sind notwendig solche Methoden zugeordnet. Die Angabe von Eigenschaften und Ereignissen ist dabei optional.

Es ist möglich mehrere Memokomponenten in einem Projekt gleichzeitig zu verwenden. Jede dieser Komponenten ist ein anderes Objekt, da sie sich zumindest im Namen unterscheiden. Da es sich aber in jedem Fall um Memokomponenten handelt, gehören sie zur gleichen Klasse TMemo. Vergleichen Sie dies ruhig noch einmal mit den Hunden:
Dackel Paul ist ein einzelnes Objekt, Schäferhund Bello ebenfalls. Sie gehören aber beide zur gleichen Klasse HUND.

Ein Objekt besteht aus Methoden und (sehr oft auch) aus Eigenschaften und Ereignissen. Eigenschaften repräsentieren die Daten, die im Objekt enthalten sind. Die Methoden sind Aktionen, die das Objekt ausführen kann. Ereignisse sind Bedingungen, auf die das Objekt reagiert. Jedes Objekt gehört einer Klasse an.

12.1.2. Vererbung

Klassen in der OOP

In der OOP können Klassen wieder Unterklassen aufweisen. Jede Unterklasse erbt dabei die Eigenschaften, Ereignisse und Methoden der übergeordneten Klasse. Auch dies kann wieder mit der Tierwelt verglichen werden. Die Klasse HUND gehört zu den RAUBTIEREN, diese wieder zu den SÄUGETIEREN und diese zu den WIRBELTIEREN. Jede Untergeordnete Klasse erbt die Merkmale der übergeordneten Klasse.

Jede Memokomponente gehört der Klasse TMemo an, diese Klasse ist eine Unterklasse von TCustomMemo. Letztendlich gehören alle Komponenten zur Klasse TObject.
Alle Komponenten haben die Eigenschaften, Ereignisse und Methoden der Klasse TObject geerbt.

Abb. 100: Hierarchie einer Memokomponente

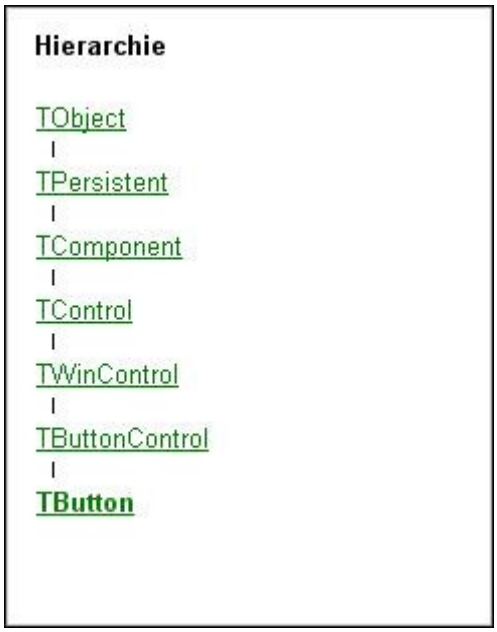


Abb. 101: Hierarchie einer Buttonkomponente

In der Objektorientierten Programmierung ist es möglich neben den bereits vorhandenen Komponenten (Klassen) eigene Klassen zu erstellen und in ein Projekt einzubinden. Es ist auch möglich neue Komponenten so zu erstellen, dass diese in allen neuen Projekten zur Verfügung stehen.

12.2. Grafikobjekte erzeugen

Einen Kreis kann man über die Klasse ELLIPSE in Delphi zeichnen. Dabei müssen aber immer entsprechende Berechnungen angestellt werden, um das umgebende Rechteck zu zeichnen. Wir wollen hier eine Klasse zur Darstellung eines Kreises konstruieren, mit dem durch die Eingabe von Mittelpunkt und Radius ein solcher Kreis dargestellt wird. Außerdem soll es möglich sein über bestimmte Methoden Flächeninhalt und Umfang des Kreises auszugeben.
Zur Erzeugung einer solchen Klasse müssen drei Schritte ausgeführt werden.

- Deklarieren der neuen Klasse mit Eigenschaften und Methoden (Ereignisse werden nicht benötigt)
- Quelltexte der Methoden schreiben
- Einbinden der Klasse als Bestandteil des Formblattes

Sind diese Schritte ausgeführt kann mit der Klasse "gearbeitet" werden.

Programmieraufgabe 32

Die Deklaration der Eigenschaften und Methoden kann z.B. so aussehen:

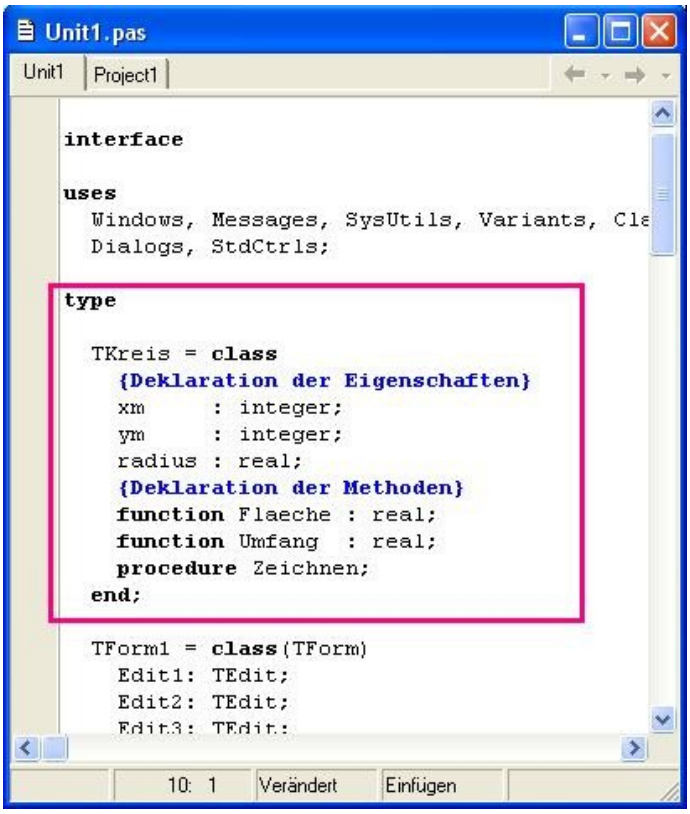


Abb. 102: Deklaration einer Komponente Kreis

In der Darstellung erkennen Sie auch, an welcher Stelle dies zu geschehen hat. Hierbei müssen zwei Funktionen (ähnlich den Standardfunktionen) selbst implementiert werden. Diese Funktionen sind dabei aber nur der Klasse TKreis zugeordnet.

Implementierung der Methoden

Im Anschluss daran sollten die Methoden (also die beiden Funktionen und die Prozedur) geschrieben werden. Diese Prozeduren und Funktionen gehören in den IMPLEMANTATION-Teil der Unit.

```
{Es folgen die Methodendeklarationen für den Kreis}

 function TKreis.Flaeche;
 begin
   result := pi*radius*radius;
 end;

 function TKreis.Umfang;
 begin
   result := 2*pi*radius;
 end;

 procedure TKreis.Zeichnen;
 var r : integer;
 begin
   r := round(radius * 10);
   form1.Canvas.Ellipse(xm-r,ym-r,xm+r,ym+r);
 end;
```

Einbinden der Klasse in das Formblatt

Das Einbinden der Klasse erfolgt in zwei Schritten:

1. Deklaration einer globalen Variablen vom Typ TKreis.
2. Aufnehmen dieser Variablen in das Formblatt.

1. Deklaration einer globalen Variablen vom Typ TKreis

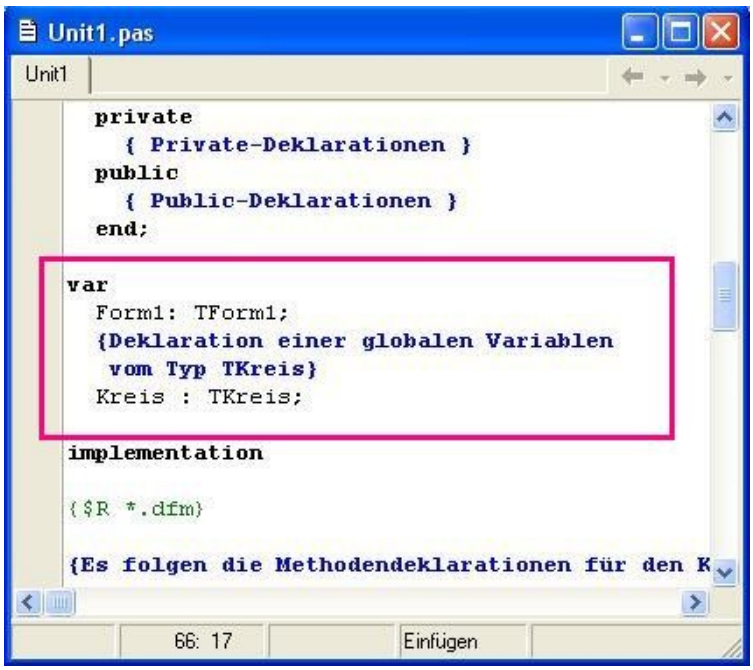


Abb. 103: Deklaration einer globalen Variable vom Typ TKreis

2. Aufnehmen der Variablen in das Formblatt

Erzeugen Sie dazu die Methode FormCreate des Formblattes indem Sie doppelt auf das Formblatt klicken. Geben Sie den nachfolgenden Quelltext ein:

```
{Das neue Objekt muss nun noch in das Formblatt aufgenommen werden.}

 procedure TForm1.FormCreate(Sender: TObject);
 begin
   Kreis := TKreis.Create;
 end;
```

Verwenden der neuen Klasse

Legen Sie dazu auf die Oberfläche des Formblattes 3 Edit, 2 Label und einen Button. Die Edit-Komponenten dienen der Eingabe des Mittelpunktes und des Radius, die Label der Ausgabe des Umfangs und des Flächeninhalts der Kreises.

```
procedure TForm1.Button1Click(Sender: TObject);
 begin
   repaint;
   kreis.xm := strtoint (edit1.Text);
   kreis.ym := strtoint (edit2.Text);
   kreis.radius := strtofloat (edit3.Text);
   Kreis.Zeichnen;
   label3.Caption := floattostr (kreis.Flaeche);
   label4.Caption := floattostr (kreis.Umfang);
 end;
```

PROGRAMME / 061 / Eigene_Komponenten.DPR

Fragen und Aufgaben
12. Klassen und Objekte

1. Was verstehen Sie unter Klassen und Objekten in der objektorientierten Programmierung?
2. Suchen Sie in der Delphi-Hilfe die Objektableitung für ein Objekt vom Typ TLabel.

3. Lösungen/012/035/project1.dpr
   Erstellen Sie in einem Delphi-Programm ein Objekt TQuadrat, mit dem Sie das Quadrat zeichnen, sowie Umfang und Flächeninhalt ausgeben können.

13. Suchen und Sortieren

13.1. Rekursionen

Den Begriff Rekursion kennen Sie bereits aus der Mathematik. Im Zusammenhang mit Zahlenfolgen haben Sie die rekursiven Zuordnungsvorschriften kennen gelernt. Die rekursive Bildungsvorschrift verlangte, dass alle Folgeglieder bis zu einem bestimmten Endwert berechnet wurden. Um z.B. das 100. Folgeglied berechnen zu können mussten Sie zunächst die ersten 99 Glieder der Zahlenfolge berechnen.

Unter Rekursionen (bzw. rekursive Prozeduren) in der Informatik versteht man Prozeduren, die sich bei der Programmabarbeitung mehrmals selbst aufrufen. Wichtig dabei ist, dass hier eine Abbruchbedingung vorgegeben wird, die Rekursion also terminierend ist.

Programmieraufgabe 33

Rekursionen können ähnlich wie Schleifen benutzt werden. Sie werden immer dann angewendet, wenn der Inhalt einer Prozedur entsprechend der Aufgabenstellung mehrmals durchlaufen werden muss. Ein einfaches Beispiel ist die Berechnung der Fakultätsfunktion.

Programmbeispiel

Zur Berechnung der Fakultät muss bis zu einem Endwert mehrfach mit einer nachfolgenden Zahl multipliziert werden. Die Fakultät wird dabei in einer gesonderten Prozedur berechnet, die sich rekursiv immer wieder aufruft.
Sie benötigen dazu auf der Oberfläche 2 Edit-Komponenten und einen Button.
Achten Sie bei der Implementierung auf die Deklaration der Prozedur p_fakultaet.

```
Type

...
procedure Button1Click(Sender: TObject);    // ButtonClick Programmbeginn
procedure p_fakultaet (var lauf, fak : integer);  // Prozedur Fakultät
   ...

var
  Form1: TForm1;
  n : integer;                        // n gibt Ende an

implementation

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
{lauf gibt Nr. des Durchlaufs an, fak ist die aktuell Fakultät}
var lauf, fak : integer;
begin
  n    := strtoint (edit1.Text);      // Einlesen des Endwertes n
  lauf := 1;                          // Initialisieren des Durchlaufs
  fak  := 1;                          // Initialisieren der Fakultät
  p_fakultaet (lauf, fak);            // Proc p_fakultaet aufrufen
end;

procedure TForm1.p_fakultaet;         // Unterprogramm p_fakultaet beginnt
begin
  fak := fak * lauf;                  // Multiplikation
  lauf := lauf + 1;                   // Durchlauf um 1 erhöhen
  if lauf > n then                    // Ende erreicht?
    edit2.Text := inttostr (fak)      // JA - Ausgabe
```

```
  else
    p_fakultaet (lauf, fak);          // NEIN - p_fakultaet aufrufen
end;
```

PROGRAMME / 062 / Fakultaet.DPR

13.2. Einfache Sortieralgorithmen

Daten zu sortieren gehört zu den wichtigsten Aufgaben in der Programmierung.

Beispiele

Auswertung eines Wettkampfes, Daten müssen nach Leistung sortiert werden
Verwaltung einer Adressdatei, Daten müssen alphabetisch sortiert werden

Zum Sortieren von Daten gibt es verschiedene Algorithmen. Wir werden einige davon hier kennen lernen.

13.2.1. Selection Sort

Der SelectionSort-Algorithmus arbeitet nach dem nachfolgenden Prinzip:

1. Es wird das kleinste Element innerhalb einer Liste gesucht
2. Dieses kleinste Element wird dann mit dem Anfang der Liste vertauscht
3. Aus der verbleibenden Restliste wird nun wieder das kleinste Element gesucht
4. Dieses wird mit dem Anfang der Restliste vertauscht
5. ...

Beispiel

Beginn:	4	7	1	3	0
Suchen des Kleinsten:	4	7	1	3	0
Vertauschen:	0	7	1	3	4
Suchen des Kleinsten:	0	7	1	3	4
Vertauschen:	0	1	7	3	4
Suchen des Kleinsten:	0	1	7	3	4
Vertauschen:	0	1	3	7	4
Suchen des Kleinsten:	0	1	3	7	4
Vertauschen:	0	1	3	4	7
Sortierte Liste:	0	1	3	4	7

Das allgemeine Struktogramm zu diesem Algorithmus sieht folgendermaßen aus:

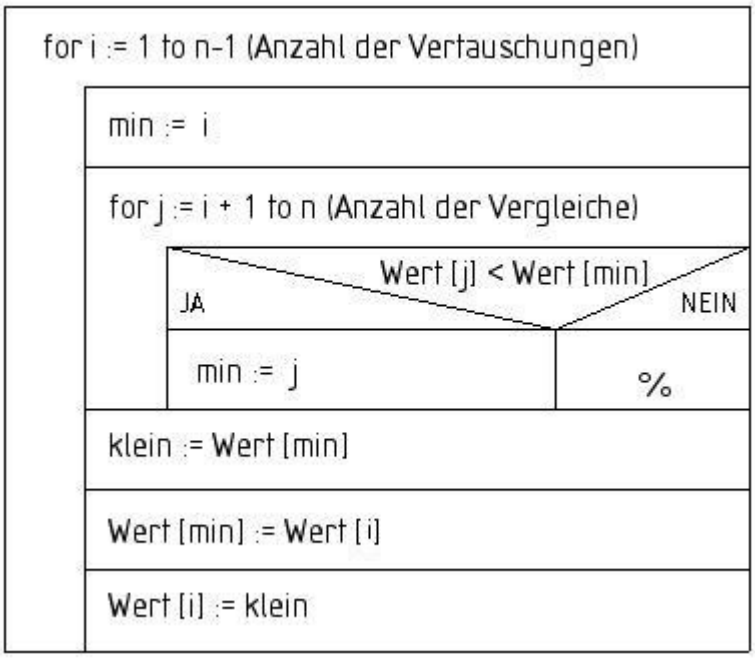


Abb. 104: Struktogramm SelectionSort

Programmieraufgabe 34

Im Folgenden sollen Sie den Selection-Sort Algorithmus an einem Beispiel implementieren. Dazu wird vom Programm ein Array mit 10 Integer Zahlen zufällig erzeugt. Dieses Array ist dabei als globale Variable zu deklarieren. Die im Array enthaltenen Daten werden dann der Größe nach sortiert. Sie benötigen auf der Oberfläche 2 Button, eine Edit- und eine Memo-Komponente.

Deklaration des globalen Arrays:

```
var
  Form1: TForm1;
  zahlen : array [1..10] of integer;
```

Erzeugen der Zufallszahlen und Anzeige in Edit1

Für die Auflistung der Zahlen in einem STRING wird ein Unterprogramm p_anzeige verwendet.

```
procedure TForm1.p_anzeige (var ausgabe : string);
var i : integer;
begin
  ausgabe := '';
  for i := 1 to 10 do
    ausgabe := ausgabe + '   ' + inttostr (zahlen [i]);
end;
```

Mit folgender Prozedur zu Button1Click wird das Array erzeugt und die Daten nach Aufruf von p_anzeige in Edit1 angezeigt.

```
procedure TForm1.Button1Click(Sender: TObject);
var i       : integer;
    ausgabe : string;
begin
  randomize;
  for i := 1 to 10 do
    zahlen [i] := random (500);
  p_anzeige (ausgabe);
  edit1.Text := ausgabe;
end;
```

Sortieren des Arrays

Zur Anzeige der einzelnen Schritte wird die Prozedur p_anzeige mehrfach aufgerufen.

```
procedure TForm1.Button2Click(Sender: TObject);
var i, j       : integer;
    min, klein : integer;
    ausgabe    : string;
begin
  memo1.Text := '';
  for i := 1 to 9 do
    begin
      min := i;
      for j := i+1 to 10 do
        begin
          if zahlen [j] < zahlen [min] then
            min := j;
        end;
      klein := zahlen [min];
      zahlen [min] := zahlen [i];
      zahlen [i] := klein;
      p_anzeige (ausgabe);
      memo1.Lines.Add('Durchlauf ' + inttostr (i) + ': ' + ausgabe);
    end;
end;
```

PROGRAMME / 063 / SelectionSort.DPR

13.2.2. Insertion Sort

Der InsertionSort-Algorithmus arbeitet nach dem nachfolgenden Prinzip:

1. Vergleiche die ersten beiden Elemente und vertausche diese gegebenenfalls
2. Beginne jetzt mit dem 3. Element und bewege es so lange nach links, bis es am richtigen Platz steht
3. Beginne jetzt mit dem 4. Element ...
4. ...

Beispiel

Beginn:	4	7	1	3	0
2. und 1. vergleichen:	4	7	1	3	0
kein Austausch:	4	7	1	3	0
3. und 2. vergleichen:	4	7	1	3	0
Vertauschen:	4	1	7	3	0
2. und 1. vergleichen:	4	1	7	3	0
Vertauschen:	1	4	7	3	0
4. und 3. vergleichen:	1	4	7	3	0
Vertauschen:	1	4	3	7	0
3. und 2. vergleichen:	1	4	3	7	0
Vertauschen:	1	3	4	7	0
Das ganze jetzt etwas kürzer, es werden nur noch die Vertauschungen gezeigt.					
Vertauschen 5. und 4.	1	3	4	0	7
Vertauschen 4. und 3.	1	3	0	4	7
Vertauschen 3. und 2.	1	0	3	4	7
Vertauschen 2. und 1.	0	1	3	4	7

Struktogramm

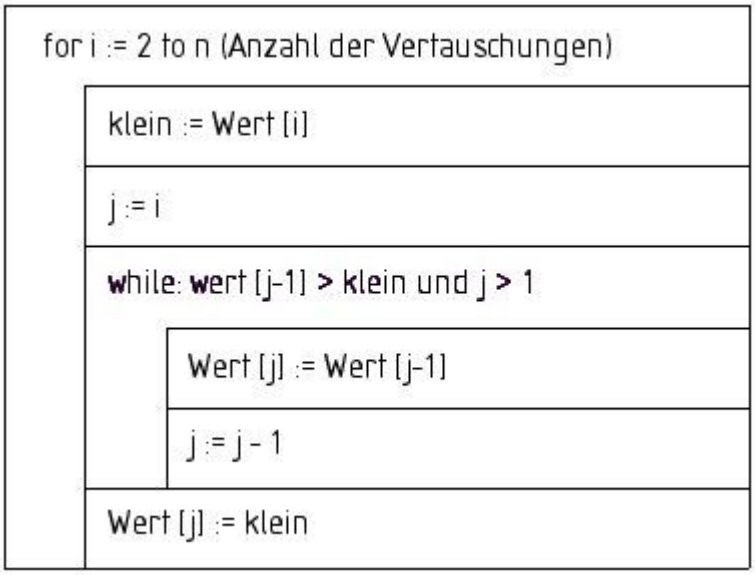


Abb. 105: Struktogramm InsertionSort

Implementieren Sie den Insertion-Sort Algorithmus selbständig. Nutzen Sie dazu wieder ein Array mit 10 Zufallszahlen.

PROGRAMME / 064 / InsertionSort.DPR

13.2.3. Bubble Sort

Der BubbleSort-Algorithmus arbeitet nach dem nachfolgenden Prinzip:

1. Vergleiche die ersten beiden Elemente und vertausche diese gegebenenfalls
2. Vergleiche das zweite und dritte Element und vertausche diese ggf.
3. ...
4. Das größte Element befindet sich nun an der richtigen Stelle
5. Vergleiche die ersten beiden Elemente der Restliste und vertausche diese gegebenenfalls
6. Vergleiche das zweite und dritte Element der Restliste und vertausche diese ggf.
7. ...

Beispiel

Beginn:	4	7	1	3	0
1. und 2. vergleichen:	4	7	1	3	0
kein Austausch:	4	7	1	3	0
2. und 3. vergleichen:	4	7	1	3	0
Vertauschen:	4	1	7	3	0
3. und 4. vergleichen:	4	1	7	3	0
Vertauschen:	4	1	3	7	0
4. und 5. vergleichen:	4	1	3	7	0
Vertauschen:	4	1	3	0	7
Das ganze jetzt etwas kürzer, es werden nur noch die Vertauschungen gezeigt.					
Vertauschen 1. und 2.:	1	4	3	0	7
Vertauschen 2. und 3.:	1	3	4	0	7
Vertauschen 3. und 4.:	1	3	0	4	7
Kein Austausch 1. und 2.	1	3	0	4	7
Vertauschen 2. und 3.:	1	0	3	4	7
Vertauschen 1. und 2.:	0	1	3	4	7

Struktogramm

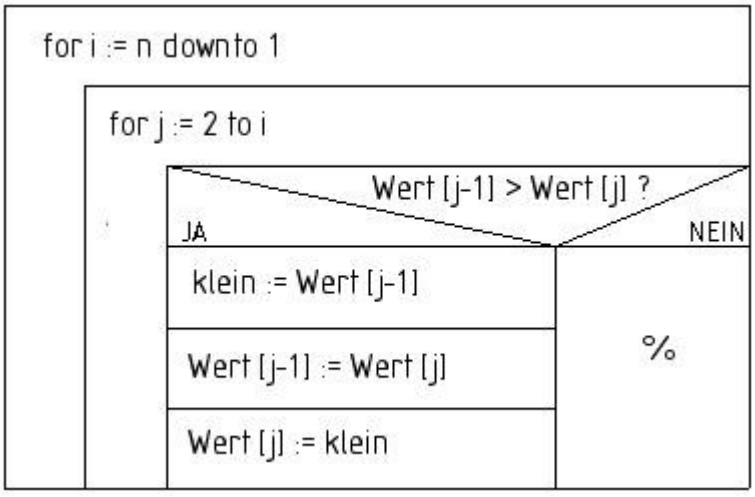


Abb. 106: Struktogramm BubbleSort

Implementieren Sie auch den Bubble-Sort Algorithmus selbständig.

PROGRAMME / 065 / BubbleSort.DPR

13.3. QuickSort Algorithmus

Der QickSort-Algorithmus arbeitet nach dem nachfolgenden Prinzip:

1. Falls die Folge F länger als ein Element ist
2. Wähle ein festes Element k in der Folge F (z.B. das mittlere oder das letzte). Dieses Element nennt man auch das Pivotelement.
3. Teile die Folge F in die Teilfolgen F1 und F2 ein (ohne das Element k zu benutzen).
4. Ändere die Folge F1 so ab, dass F1 nur Elemente enthält, die kleiner oder gleich k sind.
5. Ändere die Folge F2 so ab, dass F2 nur Elemente enthält, die größer als k sind.
6. Setze F1 als F und beginne von vorn (rekursiv).
7. Setze F2 als F und beginne von vorn (rekursiv).

Beispiel

Beginn:	3	7	1	4	5	9	0
Pivotelement 3:	**3**	7	1	4	5	9	0
F1 und F2 nach erstem Austausch	1	0	**3**	7	4	5	9
Folge F1 bearbeiten	1	0	**3**	7	4	5	9
neues Pivotelement 1:	**1**	0	**3**	7	4	5	9
F11 und F12 nach zweitem Austausch	0	**1**	**3**	7	4	5	9
F111 hat die Länge 0, und muss daher nicht bearbeitet werden. Daher folgt die Bearbeitung der Folge F2.	**0**	**1**	**3**	7	4	5	9
Pivotelement 7:	**0**	**1**	**3**	**7**	4	5	9
F21 und F22 nach drittem Austausch	**0**	**1**	**3**	4	5	**7**	9
Folge F21 bearbeiten:	**0**	**1**	**3**	4	5	**7**	9
Pivotelement 4:	**0**	**1**	**3**	**4**	5	**7**	9
Folgen nach Vertauschung	**0**	**1**	**3**	**4**	5	**7**	9
Alle verbleibenden Folgen haben nur noch die Länge 1 und müssen daher nicht mehr weiter bearbeitet werden.	**0**	**1**	**3**	**4**	**5**	**7**	**9**

Entwickeln Sie ein Struktogramm für den QuickSort-Algorithmus.
Eine Implementierung des Algorithmus finden Sie im Folgenden:

PROGRAMME / 066 / SORTIEREN4.DPR

13.4. Suchen von Daten in einem Array

Oftmals ist es auch notwendig in einer Liste nach bestimmten Daten zu suchen. Hierfür gibt es wieder mehrere unterschiedliche Verfahren.

13.4.1. Sequentielles Suchen

Beim Sequentiellen Suchen wird eine Liste vom Beginn an untersucht. Jedes einzelne Feld wird dabei mit der zu suchenden Zeichenkette oder dem zu suchenden Wert verglichen, bis eine Übereinstimmung gefunden wurde.
Der allgemeine Algorithmus lautet:

1. Beginne mit dem ersten Element.
2. Durchlaufe die Liste vom Beginn bis zum Ende.
3. Tritt eine Gemeinsamkeit auf, dann gib das gefundene Element aus, ansonsten gehe weiter.

Erstellen Sie das entsprechende Struktogramm und implementieren Sie die Suche in einem Array.

13.4.2. Binäres Suchen

Voraussetzung für die binäre Suche ist, dass die Liste in geordneter Reihenfolge vorliegt. Daher ist es in der Regel notwendig, die Liste mit einem bekannten Sortierverfahren zu sortieren. Die binäre Suche erfolgt dann rekursiv nach der folgenden Verfahrensweise:

1. Teile die Liste rekursiv in zwei Hälften.
2. Prüfe in welcher Hälfte die gesuchte Zeichenkette oder der gesuchte Wert liegen müsste.
3. Teile die gefundene Hälfte wieder in zwei Hälften
4. ...

Erstellen Sie das entsprechende Struktogramm.
Das zweite Programm stellt die Suche in Texten dar.

PROGRAMME / 067 / SUCHEN.DPR
PROGRAMME / 068 / Suchen_im_Text.DPR

13.5. Komplexität

Warum gibt es für die Sortierung von Daten so viele unterschiedliche Algorithmen?
In welchen Fällen sollten die einzelnen Algorithmen angewendet werden?
Diese beiden Fragen führen zu der Fragestellung: Wie gut sind die Algorithmen?
Zwar werden in allen vier hier behandelten Algorithmen die Daten genau sortiert, da in den Beispielen aber verhältnismäßig kleine Datensätze verwendet wurden, ist das eigentliche Problem bei der Programmabarbeitung aber nicht erkennbar.
Unternehmen Sie einmal den nachfolgenden (Gedanken)Versuch:
Gegeben ist ein Array mit 100 Folgegliedern, welches der Größe nach sortiert werden soll. Mit welchem der vier Algorithmen sind sie am schnellsten fertig?
Natürlich hängt die Aufgabe auch davon ab, inwieweit die Arrays bereits vorsortiert sind. Im Allgemeinen werden Sie aber recht unterschiedliche Ergebnisse erhalten. Die Aufgabenstellung lässt sich dabei am schnellsten mit dem QuickSort Algorithmus ausführen.

Die Komplexität (meist als Zeitkomplexität betrachtet) gibt an, wie viele Schritte bis zur vollständigen Abarbeitung eines Algorithmus benötigt werden. Diese sind insbesondere davon abhängig, wie oft bestimmte Schleifen durchlaufen werden, wie viele Vergleiche vorgenommen werden und wie oft Datensätze dabei bewegt werden. Wird ein Algorithmus auf seine Komplexität untersucht, dann werden dabei die Dauer für verschieden große Anzahlen von Eingangsdatensätzen miteinander verglichen.

Das folgende Programm zeigt Ihnen die verschiedenen hier behandelten Sortieralgorithmen im Überblick. Außerdem wird ein Komplexitätsvergleich vorgenommen.
Der Quelltext zu diesem Programm ist hier nur für besonders interessierte Schüler gedacht.

PROGRAMME / 069 / SORTIEREN.DPR

Fragen und Aufgaben
13. Suchen und Sortieren

Grundlagen

1. Sortieren Sie die nachfolgenden Zahlenreihen mit Hilfe der Ihnen bekannten Sortierverfahren.
   a. 4; 19; 2; 16; 5
   b. 0; 7; -2; 8; 0
   c. 16; 9; 8; 5; 1

Programmieraufgaben

2. Lösungen/013/036/Project1.dpr
   Es ist die Summe der ersten n natürlichen Zahlen mit Hilfe einer rekursiven Prozedur zu ermitteln.

3. Lösungen/013/037/Project1.dpr
   Entwickeln Sie eine Delphi-Prozedur zur Berechnung der Kreiszahl p mit Hilfe der Formel:
   pi/2 = 2/1 * 2/3 * 4/3 * 4/5 * 6/5 * 6/7 * 8/7 * ... * (n-2)/(n-3) * (n-2)/(n-1) * n/(n-1) * n/(n+1)
   Die geforderte Genauigkeit soll dabei über eine Edit-Komponente eingegeben werden. Die Programmabarbeitung ist rekursiv zu gestalten.

4. Lösungen/013/038/Project1.dpr
   Ermitteln Sie die Nullstellen einer ganzrationalen Funktion 3. Grades mit Hilfe des Bisektionsverfahrens. Es ist dabei die gewünschte Genauigkeit und ein Anfangsintervall vom Nutzer vorzugeben. Dabei soll das Programm zunächst überprüfen, ob in diesem Intervall überhaupt eine Nullstelle existiert. Die Abarbeitung des Programms soll rekursiv erfolgen.

5. Es soll auf rekursiven Weg eine Prozedur entwickelt werden, mit der Kreise mit kleiner werdendem Radius dargestellt werden können. Dabei ist das Intervall der Darstellung vom Nutzer vorzugeben. Die Kreise sind mit gleichem Mittelpunkt darzustellen.

6. Entwickeln Sie ein Delphi-Programm zur Verwaltung einer Bibliothek. Es sind die folgenden Daten über einen Record-Typ in einem dynamischen Array zu verwalten:
   Titel, Autor, Buchnummer, Jahr. Es soll weiterhin die Möglichkeit bestehen die Bücher nach Autor bzw. nach Nummer zu sortieren.

7. Entwickeln Sie ein Delphi-Programm zur Auswertung eines Laufwettbewerbs (Weitsprungwettbewerbs). Die Daten sollen dabei in einem Array mit einem Recordtyp verwaltet werden. Der Recordtyp soll dabei die folgenden Daten erfassen: Name, Vorname, Startnummer, Leistung. An den Wettbewerben nehmen jeweils 20 Sportler teil. Anzuzeigen ist jeweils eine Liste der Ergebnisse sortiert nach Platz. Nutzen Sie dafür eine Komponente StringGrid.

SYNTAXDIAGRAMME

Syntaxdarstellungen 1

Eigenschaften von Komponenten

Änderung der Farbe einer Komponente

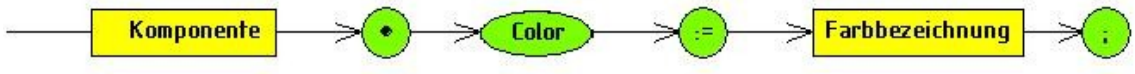


Abb. 107: Ändern der Farbe einer Komponente

Für die Farbbezeichnungen vgl. Kapitel 8.4.

Änderung der Beschriftung einer Komponente

Abb. 108: Ändern der Beschriftung einer Komponente

Beachten Sie:

Die Beschriftung einer Edit-Komponente wird dabei über die Eigenschaft TEXT ausgeführt. Sollen Zahlen in der Komponente ausgegeben werden, müssen diese zunächst in einen Text umgewandelt werden.

Änderung der Schriftstile

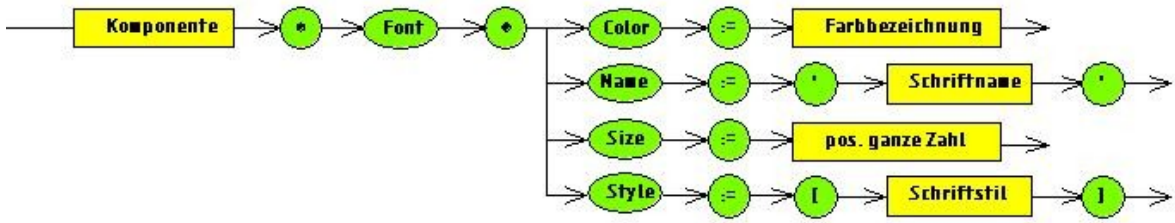


Abb. 109: Ändern der Schriftstile einer Komponente

Beachten Sie:

Nutzen Sie Schriftarten nur vorsichtig. Bei ausgefallenen Schriftarten kann es sein, dass diese auf dem PC des Nutzers nicht vorhanden sind und damit auch nicht dargestellt werden können.
Für die Schriftstile stehen folgende Möglichkeiten zur Verfügung.

fsstrikeout	(durchgestrichener Text)
fsbold	(Text wird fett dargestellt)
fsitalic	(Text wird kursiv dargestellt)
fsunderline	(Text wird unterstrichen)

Die Stile können durch Komma getrennt auch miteinander kombiniert werden.

Syntaxdarstellungen 2

Umwandlung von Zahlen und Zeichen

Umwandlung einer Gleitkommazahl in einen String

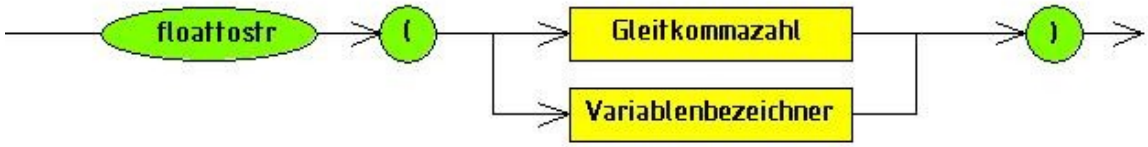


Abb. 110: Umwandlung einer Gleitkommazahl in einen String

Umwandlung einer Ganzzahl in einen String

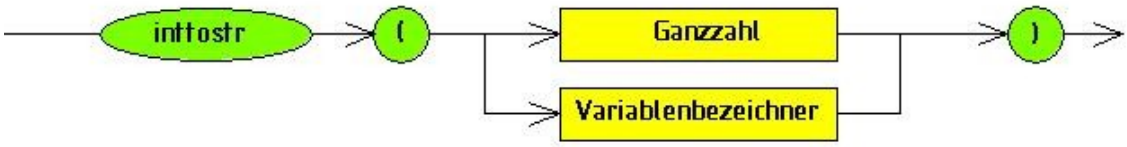


Abb. 111: Umwandlung einer Ganzzahl in einen String

Umwandlung eines Strings in eine Gleitkommazahl

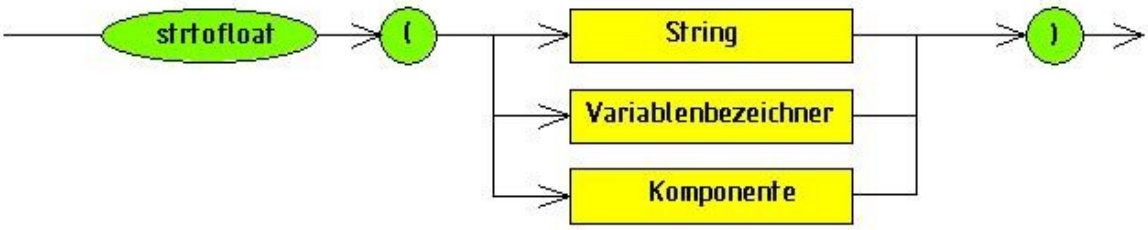


Abb. 112: Umwandlung eines Strings in eine Gleitkommazahl

Beachten Sie:

Beim Lesen aus einer Komponente muss auf die Eigenschaft Caption oder Text zugegriffen werden.

Umwandlung eines Strings in eine Ganzzahl

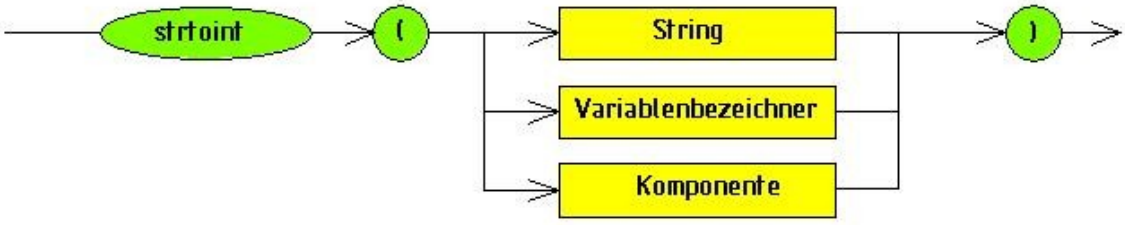


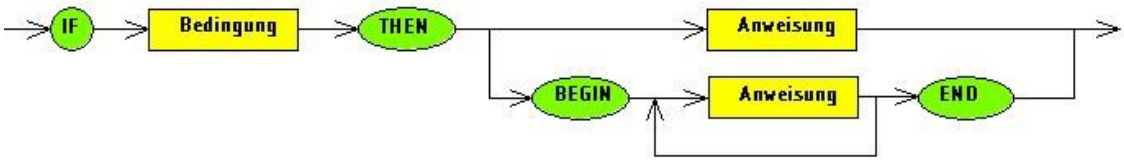
Abb. 113: Umwandlung eines Strings in eine Ganzzahl

Beachten Sie:

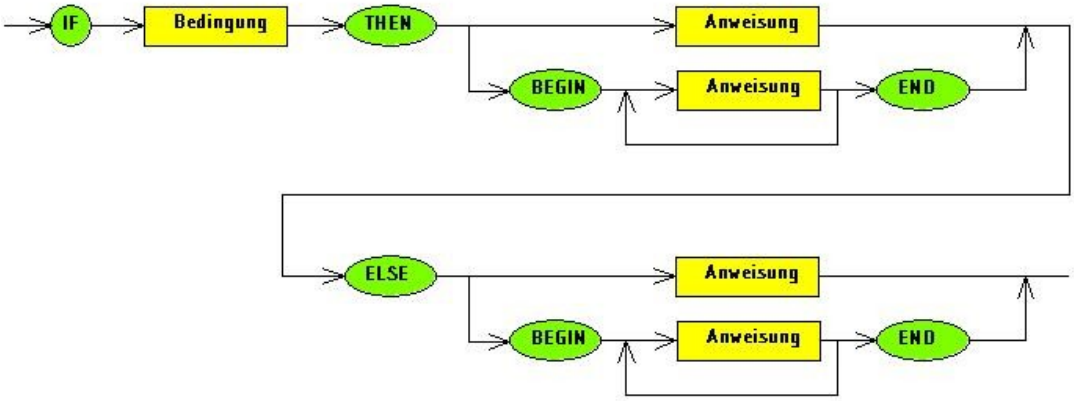
Beim Lesen aus einer Komponente muss auf die Eigenschaft Caption oder Text zugegriffen werden.

Syntaxdarstellungen 3

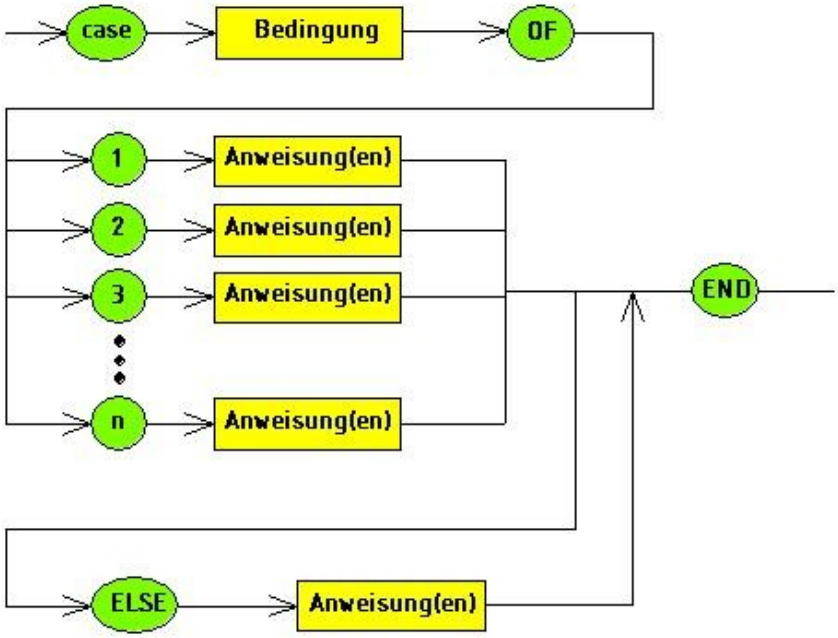
Einfache Alternative



Vollstädige Alternative



Fallauswahl



Beachten Sie:

Gehören zu einem Auswahlpunkt mehrere Anweisungen, so müsen diese in BEGIN und END gefasst werden.
Der ELSE - Zweig kann entfallen.

LÖSUNGEN

Struktogramm zur Berechnung der quadratischen Gleichung

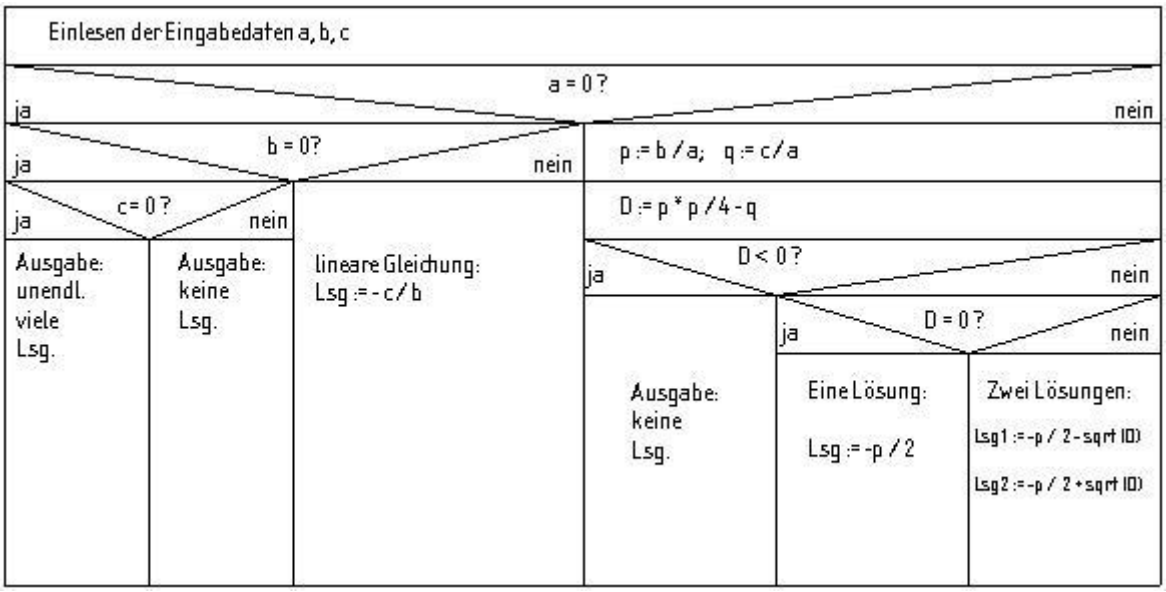


Abb. 117: Struktogramm quadratische Gleichung

Quelltext quadratische Gleichung

```
procedure TForm1.Button1Click(Sender: TObject);
{Löen einer quadratischen Gleichung ax² + bx + c = 0}
var a, b, c : real;
    p, q, d : real;
    x1, x2  : real;
begin
 P_Lsg1.Caption := '';
 P_Lsg2.Caption := '';
 P_Mitteilung.Caption := '';
 a := strtofloat (E_Eing_a.Text);
 b := strtofloat (E_Eing_b.Text);
 c := strtofloat (E_Eing_c.Text);
 if a=0 then                              // a = 0
  begin
   if b=0 then                            // a = 0, b = 0
    begin
     if c=0 then                          // a = 0, b = 0, c = 0
      P_Mitteilung.Caption := 'unendlich viele Nullstellen'
     else                                 // a = 0, b = 0, c <> 0
       P_Mitteilung.Caption := 'keine Nullstellen';
    end
   else
    begin                                 // a = 0, b <> 0 ==> lineare Glg.
     P_Mitteilung.Caption := 'eine Nullstelle';
     x1 := -c/b;
     P_Lsg1.Caption := floattostr (x1);
    end
  end
 else                                     // a <> 0
```

```
  begin
   p := b/a;
   q := c/a;
   d := p*p/4-q;                      // Berechnung der Diskriminante d
   if d<0 then                        // d < 0
    P_Mitteilung.caption := 'keine Nullstelle'
   else
    begin
     if d = 0 then                    // d = 0
      begin
       P_Mitteilung.Caption := 'eine Nullstelle';
       x1 := -p/2;
       P_Lsg1.Caption := floattostr (x1);
      end
     else                             // d > 0
      begin
       P_Mitteilung.Caption := 'zwei Nullstellen';
       x1 := -p/2 - sqrt(d);
       x2 := -p/2 + sqrt(d);
       P_Lsg1.Caption := floattostr (x1);
       P_Lsg2.Caption := floattostr (x2);
      end;
    end;
  end;
end;
```

Quelltext kgV und ggT

```
  procedure TForm1.Button1Click(Sender: TObject);
  var a, b                 : integer;
      a1, b1, a2, b2, rest : integer;
      ggt, kgv             : integer;
  begin
    a := strtoint (edit1.Text);
    b := strtoint (edit2.Text);
    a1 := a;
    b1 := b;
    repeat
      rest := a1 mod b1;
      ggt := b1;
      a1 := b1;
      b1 := rest;
    until (rest =0);
    a2 := a div ggt;
    b2 := b div ggt;
    kgv := a2*b2*ggt;
    edit3.Text := inttostr (ggt);
    edit4.Text := inttostr (kgv);
  end;
```