1. Programmieren mit Delphi – Grundlagen

1.1. Was ist Delphi?

Delphi ist ein Entwicklungssystem zum Erstellen von Windowsprogrammen. Dazu werden ein leistungsfähiger Pascal-Compiler, visuelle Komponenten und die Möglichkeit des Erstellens von Datenbankprogrammen in einem System vereinigt.

Mit Delphi kann *jeder* einfach, sicher und schnell Windowsprogramme entwickeln.

Der Vorteil von Windowsprogrammen liegt in ihrer einheitlichen Bedienung. Die meisten Windowsprogramme besitzen eine Menüleiste und ein Hauptfenster und lassen sich größtenteils mit der Maus bedienen. Programme werden in Fenstern ausgeführt, die oft nur einen Teil des gesamten Bildschirmes beanspruchen. Dieser Fenstertechnik verdankt Windows seinen Namen. Über Fenster und Dialoge, die sogenannten Benutzerschnittstellen, kommuniziert der Anwender mit dem Programm.

1.2. Algorithmen und Programme

Die elektronische Datenverarbeitung (EDV) ermöglicht die Erleichterung der menschlichen Arbeit durch den Einsatz von Maschinen bzw. Computern. Damit der Computer verschiedene Arbeitsschritte automatisch ausführen kann, müssen diese vorher genau beschrieben werden. Ein Algorithmus ist eine Folge von Anweisungen, die genau diese Arbeitsschritte beschreiben.

Jedes Problem, dessen Lösung durch einen Algorithmus beschrieben werden kann, ist im Prinzip durch einen Computer lösbar.

Ein Programm ist eine Folge von Anweisungen (Algorithmus), die in einer Programmiersprache wie z.B. Pascal formuliert sind.

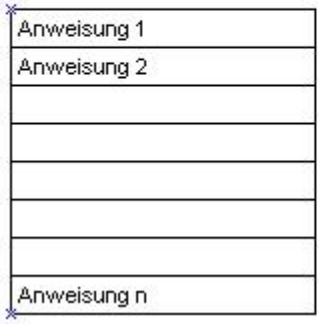
In einem Programm stehen somit nur Anweisungen, die der Computer versteht und umsetzen kann.

1.3. Vom Algorithmus zum Programm

Im folgenden wird eine Möglichkeit zur grafischen Darstellung der Grundstrukturen (Nassi-Schneiderman-Diagramme) von Algorithmen und deren Implementation mithilfe der Programmierumgebung Delphi dargestellt. Alle Sprachelemente von Delphi sind weinrot gekennzeichnet.

Die Sequenz

Die Sequenz beschreibt eine Folge von Anweisungen die nacheinander (sequentiell) abgearbeitet werden.

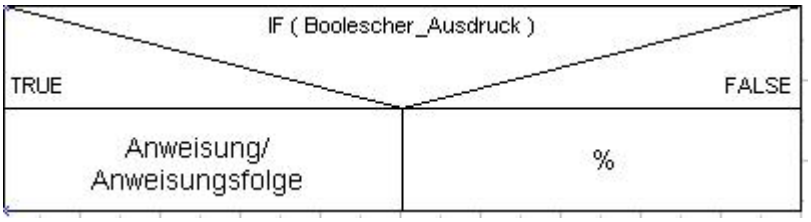


Die einseitige Auswahl

Die einseitige Auswahl ist eine Verzweigungsstruktur. Wenn eine bestimmte Bedingung erfüllt ist, dann wird eine Anweisung oder eine Anweisungsfolge abgearbeitet, die ansonsten ignoriert wird. Handelt es sich nur um eine Anweisung, ist eine Blockbildung mit begin und end nicht erforderlich.

```
IF Bedingung Then Anweisung;
```

```
IF Bedingung Then
begin
  Anweisungsfolge;
end;
```

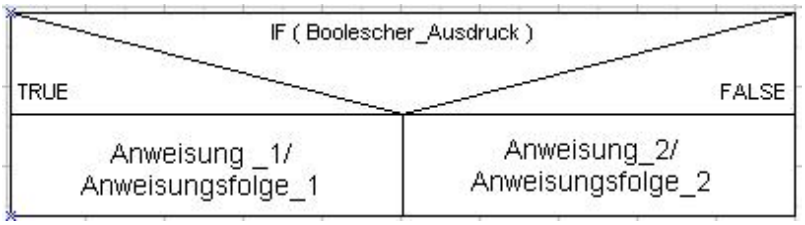


Die zweiseitige Auswahl

Die zweiseitige Auswahl ist eine Verzweigungsstruktur. Wenn eine bestimmte Bedingung erfüllt ist, dann wird eine Anweisung oder eine Anweisungsfolge abgearbeitet, anderenfalls wird eine alternative Anweisung oder Anweisungsfolge ausgeführt. Handelt es sich nur um eine Anweisung, ist eine Blockbildung mit begin und end nicht erforderlich. Zu beachten ist, dass vor else kein Semikolon gesetzt werden darf.

```
IF Bedingung Then Anweisung_1 else Anweisung_2;
```

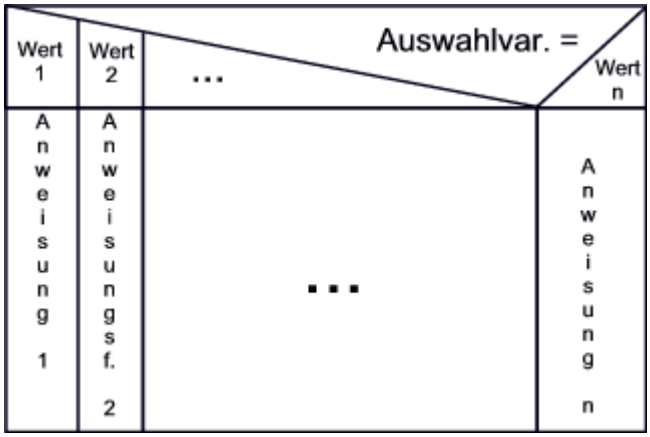
```
IF Bedingung Then
begin
  Anweisungsfolge_1;
end
else
begin
  Anweisungsfolge_2;
end;
```



Die Mehrfachauswahl

Die Mehrfachauswahl ist eine Verzweigungsstruktur. In Abhängigkeit der Belegung einer Auswahlvariablen, wird eine Fallauswahl getroffen und die zugehörige Anweisung/ Anweisungsfolge abgearbeitet.

```
Case Auswahlvariable of
wert 1 : Anweisung_1;
wert 2 : begin Anweisungsfolge_2; end;
...
wert n : Anweisung_n;
end;
```

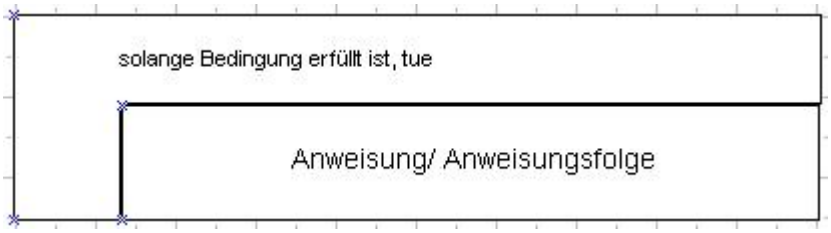


Die anfangsgeprüfte Schleife

Die anfangsgeprüfte (abweisende) Schleife ist eine Wiederholungsstruktur. Die im Schleifenkörper eingebundene Anweisung/ Anweisungsfolge wird solange wiederholt, wie die im Schleifenkopf festgelegte Bedingung gilt.

```
while Bedingung do Anweisung;
```

```
while Bedingung do
begin
  Anweisungsfolge;
end;
```



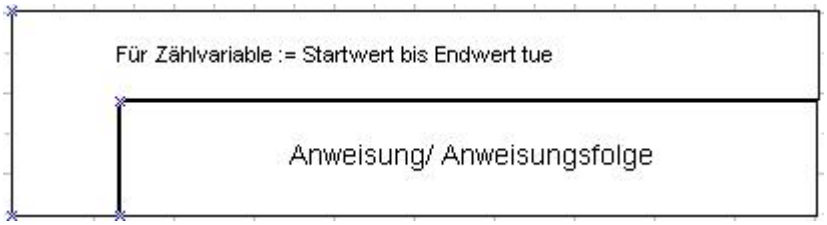
Die Zählschleife

Die Zählschleife ist eine spezielle anfangsgeprüfte Schleife mit der Eigenschaft, dass die Anzahl der Schleifendurchläufe explizit angegeben wird. Ersetzt man das Wort do durch downto, so wird "rückwärts" gezählt. Dabei ist zu beachten dass Startwert und Endwert in der entsprechenden Relation (kleiner bzw. größer als) zueinander stehen.

```
for Zählvariablen := Startwert to Endwert do Anweisung;
```

```
for Zählvariablen := Startwert to Endwert do
begin
  Anweisungsfolge;
```

```
end;
```

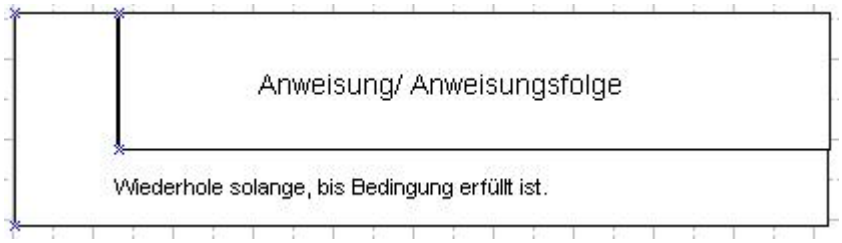


Die endgeprüfte Schleife

Die endgeprüfte (nicht abweisende) Schleife ist eine Wiederholungsstruktur. Die im Schleifenkörper eingebundene Anweisung/ Anweisungsfolge wird solange wiederholt, bis die im Schleifenfuß festgelegte Bedingung erfüllt ist

```
repeat  Anweisung; until Bedingung;
```

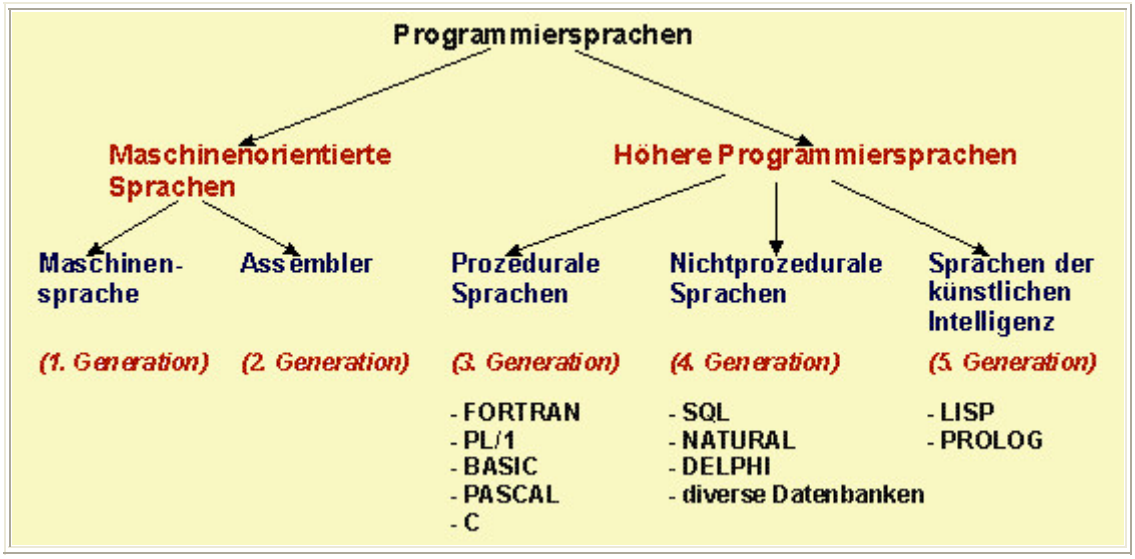
```
repeat
  Anweisungsfolge;
until Bedingung;
```



1.4. Entwicklung und Einteilung der Programmiersprachen

Da man noch nicht in natürlicher Sprache mit einem Rechner kommunizieren kann, wurden im Laufe der Jahre verschiedene Programmiersprachen entwickelt.
Der Unterschied zwischen gesprochenen Sprachen und Programmiersprachen liegt darin, dass die Worte einer Programmiersprache nur eine Bedeutung zulassen, während der Sinngehalt mancher Worte der Umgangssprache erst aus dem Kontext heraus deutlich werden kann. Ein Rechner benötigt aber stets eindeutig formulierte Anweisungen zur Bearbeitung.

Wie viele Programmiersprachen und Programmiersysteme es heute weltweit gibt, lässt sich nicht beantworten. Es können einige hundert sein, da viele Sprachen nur für spezielle Aufgaben und Einsatzgebiete konzipiert wurden. Die bekanntesten Programmiersprachen lassen sich in Auszügen in folgende Hauptgruppen unterteilen:



1. Generation: Maschinensprachen

Die ersten EDV-Anlagen (Ende der 40er Jahre) ließen sich nur maschinennah programmieren. Der Programmcode musste bitweise in den Speicher des Rechners geschrieben werden. Der Vorteil der maschinennahen Programmierung liegt bis heute darin, dass diese Art von Programm direkt von einem Computer ausgeführt werden kann. Allerdings sind sehr genaue Rechnerkenntnisse erforderlich, da alle Anweisungen in Form von elementaren Befehlen sehr kleinschrittig beschrieben werden müssen. Problematisch gestaltet sich die Fehlersuche, wenn ein Programm überhaupt nicht läuft oder falsche Ergebnisse liefert.

Beispiel:	`11001011` `00110101`	`11100011` `10111101`

2. Generation: Assemblersprachen

Die Assemblersprachen, deren Befehlsvorrat speziell für jeden Rechnertyp zugeschnitten ist, verwenden anstelle des Binärcodes leichter verständliche Symbole, Mnemonics genannt. Ein Assemblerprogramm ist auf einem Computer nicht mehr direkt ablauffähig, sondern muss erst in ein entsprechendes Maschinenprogramm übersetzt werden. Ein Programm, das dies automatisch durchführt, bezeichnet man als Assembler, den Übersetzungsvorgang als assemblieren. Der Nachteil von Assemblerprogrammen besteht darin, dass sie auf eine ganz bestimmte Hardware zugeschnitten sind und sich nur schwer auf andere Computertypen übertragen lassen. Bei größeren Problemlösungen werden die Programme sehr umfangreich und damit wartungsunfreundlich. Daher werden Assemblersprachen hauptsächlich nur noch da, wo Programme und Programmsysteme schnell reagieren müssen, und für Teile des Betriebssystems eingesetzt.

Beispiel:	`ADD FELD_2 FELD_3` `MOV BX, OFFSET FELD_3`

3. Generation: Prozedurale Programmiersprachen

Diese Sprachengeneration, der die überwiegende Mehrheit der heute gebräuchlichen Programmiersprachen angehört, ist unabhängig von einem Computersystem. Lediglich der Übersetzer (Interpreter oder Compiler) muss an das jeweilige System angepasst sein und den entsprechenden Maschinencode erzeugen. Prozedurale Sprachen besitzen einen speziellen, der menschlichen Sprache angenäherten Befehlssatz, um Probleme aus einem bestimmten Anwendungsbereich zu lösen. Sie lehnen sich somit an die Denkweise des Programmierers an. Auch ohne fundamentierte Programmierkenntnisse lassen sich diese Programme leicht nachvollziehen. Die Bezeichnung "prozedural" kennzeichnet den modularen Aufbau der entsprechenden Programme in Prozeduren oder Funktionen.

Beispiel:

```
Write('Fahrstrecke='); Readln(kilometer);
Write('Benzin='); Readln(liter);
verbrauch := liter/kilometer * 100;
Writeln('Sie verbrauchten auf 100 km ',verbrauch);
if verbrauch > 7 then writeln "Verbrauch zu hoch!";
```

4. Generation: Nichtprozedurale Programmiersprachen

Bei nichtprozeduralen Programmiersprachen wird nicht mehr festgelegt, **wie** ein Problem gelöst wird, sondern der Programmierer beschreibt lediglich, **was** das Programm leisten soll. Danach werden diese Angaben von dem Programmiersystem in ein Programm umgesetzt. Der Vorteil dieser Sprachen besteht darin, dass für diese Art der Programmierung keine umfangreiche Programmierausbildung notwendig ist. Nichtprozedurale Programmiersprachen werden z.B. für Datenbankabfragen oder Tabellenkalkulationen eingesetzt.
In **Delphi** verwendet man z.B. die visuellen Komponenten, um eine Benutzerschnittstelle zu erstellen.

Beispiel:

```
select KUNDE from TABLE_1 where ALTER > 18
create ERWACHSENE
```

5. Generation: Programmiersprachen der künstlichen Intelligenz

Die Programmierung der künstlichen Intelligenz (KI) dient der fortgeschrittenen Programmierung. Es wird versucht, die natürliche Intelligenz des Menschen (z.B. seine Lernfähigkeit) durch entsprechend konstruierte Computer nachzuvollziehen. Hierbei fließt beispielsweise auch die natürliche Sprache in die Programmierung ein. KI-Programme werden überwiegend zu Forschungszwecken eingesetzt und beschreiben Schlussfolgerungen aus Forschungsergebnissen. Erfolgreich werden derartige Systeme zur Spracherkennung eingesetzt.

Beispiel: `Berechnung auswerten.`

Einordnung von Delphi:

Als komplexes Programmiersystem lässt sich Delphi in zwei Generationen einordnen:

- **die von Delphi verwendete Programmiersprache Object Pascal**

ordnet man in der 3. Generation ein.
- **die visuellen und SQL-Komponenten gehören der 4. Generation an.**

Aufgrund dieser Einordnung wird Delphi auch als eine **hybride Programmiersprache** bezeichnet.
Die Kombination der Funktionalität zweier Generationen von Programmiersprachen mit visuellen Programmiertechniken führen zu einer hohen Bedienerfreundlichkeit bei der Programmerstellung gepaart mit einer enormen Mächtigkeit der erzeugbaren Programme.

1.5. Visuelles Programmieren mit Delphi

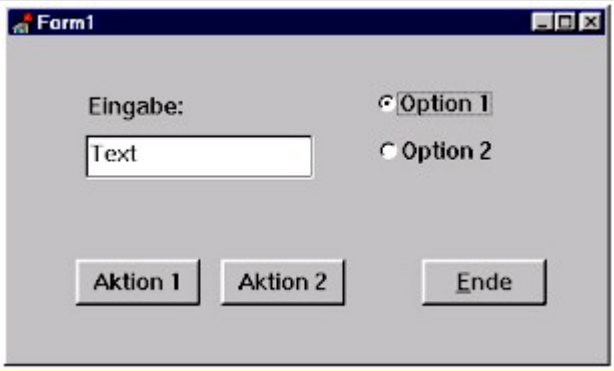
Interpreter und Compiler

Der Prozessor eines Computers kann nur Maschinenbefehle lesen (bestehend aus Binärcode 0/1). Programme, die nicht in Maschinensprache geschrieben sind, müssen erst in diese übersetzt werden.
Die Aufgabe des Übersetzens übernehmen eigens dafür entwickelte Programme, Interpreter oder Compiler genannt.

Interpreter	Interpreter übersetzen (interpretieren) die Programme zeilenweise. Das Programm kann deshalb zur Laufzeit geändert werden. Die Befehle werden Zeile für Zeile in Maschinensprache übersetzt und vom Prozessor ausgeführt. Bei jedem Neustart des Programms muss dieses auch wieder neu interpretiert werden. Aus diesem Grund können keine Optimierungen vorgenommen werden, und die Programme laufen langsamer ab. *Beispiele für Interpreter-Sprachen:* Q-BASIC, JAVA, LOGO
Compiler	Ein Compiler übersetzt einen Programmtext vollständig in Maschinensprache und legt diesen in einer eigenständigen Programm-Datei ab. Während der Compilierung optimiert der Compiler die Programmgröße und -geschwindigkeit. Beim Neustart wird vom Prozessor direkt die Programmdatei abgearbeitet. Dadurch laufen compilierte Programme 10 bis 20 mal schneller ab als zu interpretierende Programme. *Beispiele für Compiler-Sprachen:* PASCAL, DELPHI, C++

Visuelles Programmieren

Delphi erleichtert durch seine visuellen Komponenten wie Menüs, Schaltflächen und Oberflächenkomponenten das Erstellen einer Benutzerschnittstelle in Windows. Dadurch wird die Komplexität der Windowsprogrammierung, die auf Fenstern und Botschaften beruht, wesentlich vereinfacht.
Hinter den visuellen Komponenten verbergen sich nicht nur grafische Darstellungen. Vielmehr stellt jede Komponente dem Programm eine oder mehrere Funktionen zur Verfügung.



Das Programmieren unter Windows baut auf zwei wichtigen Konzepten auf, den **Fenstern** und den **Botschaften**.

Die Abbildung zeigt ein typisches **Dialogfenster**, welches mit visuellen Komponenten in Sekundenschnelle und ganz ohne "Insiderkenntnisse" erstellt werden kann.
Der Anwender kommuniziert über dieses Fenster mit dem jeweiligen Programm. Im Eingabefeld kann ein beliebiger Text oder Zahlenwert eingegeben werden, der dann über das Anklicken eines Aktionsschalters vom Programm verarbeitet wird.
Dabei stellt das Ereignis *"Schalter geklickt"* eine **Botschaft** an das Programm dar, woraufhin dieses einen entsprechenden Algorithmus (Ereignisbehandlungsroutine) ausführt.

2. Delphi

2.1. Die Delphi-Entwicklungsumgebung

Nach dem Start von Delphi werden verschiedene Fenster geöffnet, die für die visuelle Programmierarbeit notwendig sind. Diese Arbeitsfläche wird als Integrierte Entwicklungsumgebung, kurz IDE (engl. Integrated Development Enviroment) bezeichnet. Alle Fenster der IDE können frei und einzeln auf dem Bildschirm positioniert oder geschlossen werden. Durch das Schließen des Hauptfensters wird Delphi beendet.

Die nachstehende Tabelle gibt einen einführenden Überblick zu Erscheinungsbild und Funktion der wichtigsten Bestandteile der Delphi-Entwicklungsumgebung:

Das **Formular** stellt das zentrale Entwicklungsobjekt eines Delphi-Programms dar. Auf ihm werden die gewünschten Komponenten wie Schaltflächen, Menüs und Eingabefelder per Mausklick platziert und größenmäßig angepasst. Das Erscheinungsbild des Formulars entspricht dem Aussehen des Windows-Fensters, in dem das fertige Programm später ablaufen wird.

Die **Komponentenpalette** ist in verschiedene Register (Standard, Zusätzlich usw.) unterteilt, und diese erlauben die Auswahl der benötigten Windows-Komponenten.

Mithilfe des **Objektinspektors** werden Darstellungsweise und Verhalten der Objekte (Komponenten) in einem Formular sowie das Aussehen des Formulars selbst festgelegt. Das Erscheinungsbild wird über die Seite "Eigenschaften", das Verhalten beim Eintritt eines Ereignisses über die Seite "Ereignisse" eingestellt.

Der **Quelltexteditor** wird zum Schreiben des Programmcodes in der Programmiersprache Pascal genutzt. Dabei generiert Delphi für neu in das Formular eingefügte Komponenten automatisch den Programmcode, der dann im Editor angezeigt wird. Dem Programmierer obliegt daher "nur noch" die Einarbeitung der Algorithmen zur Ereignisbehandlung.

Die Komponenten der Registerkarte Standard der Komponentenpalette machen die Standard-Steuerelemente von Windows für Ihre Delphi-Anwendungen verfügbar:

MainMenu	Menüleisten mit Menüs für Ihre Formulare. Um auf Ereignisse für Einträge im Hauptmenü zuzugreifen, fügen Sie dem Formular eine MainMenu-Komponente hinzu und klicken sie doppelt an, um den Menü-Designer zu öffnen.
PopupMenu	Popup-Menüs, die angezeigt werden, wenn der Benutzer mit der rechten Maustaste klickt. Um auf Ereignisse für Einträge im Popup-Menü zuzugreifen, fügen Sie dem Formular eine PopupMenu-Komponente hinzu und klicken sie doppelt an, um den Menü-Designer zu öffnen.
Label	Text, den der Benutzer weder markieren noch bearbeiten kann, z.B. Titel oder Beschriftungen für Steuerelemente. Siehe Beschriftungen.
Edit	Ein Eingabebereich, in dem der Benutzer einzeiligen Text eingeben und bearbeiten kann. Diese Komponente ist eines von mehreren Text-Steuerelementen.
Memo	Ein Eingabebereich, in dem der Benutzer mehrzeiligen Text eingeben und bearbeiten kann. Siehe Memo.
Button	Eine Schaltfläche, mit der der Benutzer eine Aktion auslösen kann. Siehe Schaltflächen.
CheckBox	Eine Option, die der Benutzer zwischen Ja/Nein oder Wahr/Falsch umschalten kann. Kontrollfelder bilden eine Gruppe von Optionen, die sich nicht gegenseitig ausschließen. Der Benutzer kann mehrere Kontrollfelder einer Gruppe aktivieren. Siehe Kontrollfelder.
RadioButton	Eine Option, die der Benutzer zwischen Ja/Nein oder Wahr/Falsch umschalten kann. Optionsfelder bilden eine Gruppe von Optionen, die sich gegenseitig ausschließen. Der Benutzer kann immer nur ein Optionsfeld einer Gruppe markieren. Siehe Optionsfelder.
ListBox	Eine aufklappbare Liste mit Auswahlmöglichkeiten. Siehe Listenfelder und Eigenschaften aller Listen-Steuerelemente.
ComboBox	Ein Listenfeld mit Auswahlmöglichkeiten, das mit einem Eingabefeld kombiniert ist. Der Benutzer kann im Eingabefeld Daten eingeben oder einen Eintrag aus der Liste wählen. Siehe Kombinationsfelder.
ScrollBar	Eine Möglichkeit, den angezeigten Bereich einer Liste oder eines Formulars zu bewegen oder einen Bereich von Werten schrittweise zu durchlaufen. Siehe Bildlaufleisten.
GroupBox	Ein Container für eine Gruppe verwandter Optionen in einem Formular. Siehe Gruppenfelder.
RadioGroup	Ein Gruppenfeld, das Optionsfelder enthält. Siehe Optionsfeldgruppen.
Panel	Eine Tafel, die andere Steuerelemente enthält. Mit Tafeln können Sie Mauspaletten und Statusleisten erzeugen. Siehe Tafeln.

2.2. Das Prinzip der ereignisgesteuerten Programmierung

Die Programmierung mit Delphi wird auch als ereignisgesteuerte Programmierung bezeichnet. Ein Ereignis ist eine Aktion, die den Programmablauf beeinflusst.

Was sind Ereignisse?

Alle Aktionen (Tastatureingaben, Mausbewegungen) eines Benutzers, wie zum Beispiel:

- Einfaches oder doppeltes Klicken auf eine Befehlsschaltfläche,
- Verschieben, Öffnen oder Schließen eines Fensters mit der Maus,
- Positionieren des Cursors in ein Eingabefeld mit der Tabulatortaste.

Interne Programmabläufe, wie zum Beispiel:

- Berechnungen durchführen,
- Öffnen und Schließen eines Fensters (vom Programm gesteuert),
- Ermitteln von aktueller Uhrzeit und Datum.

Reaktionen auf Ereignisse

Mit Delphi werden Programme entwickelt, die durch die grafische Benutzeroberfläche von Windows mit Steuerelementen wie beispielsweise Dialogfenstern, Schaltflächen und Eingabefeldern gesteuert werden.

Durch ein **Ereignis** (z.B. Klicken auf eine Schaltfläche) wird eine **Reaktion** (z.B. Öffnen eines Fensters) hervorgerufen. Diese Reaktion wird in Form von Object-Pascal-Code in einer Prozedur erstellt, die als **Ereignisbehandlungsroutine** bezeichnet wird.

Als Ereignisbehandlungsroutinen werden diejenigen Anweisungen bezeichnet, die ausgeführt werden, sobald ein Ereignis eintritt.

2.3. Schrittfolge zur Programmerstellung mit Delphi

Die nachfolgende Tabelle soll aufzeigen, wie sich der "klassische" Software-live-cycle unter den Gegebenheiten einer visuellen Programmierumgebung erweitert bzw. modifiziert. Dabei darf man sich den Durchlauf der einzelnen Schritte keinesfalls als einmalige lineare Abfolge vorstellen, vielmehr entstehen durch Rücksprünge und das mehrmalige Durchlaufen von Teilen der Schrittfolge zyklische Strukturen. Werden z.B. bei der Resultatsprüfung unter Punkt 5) Fehler festgestellt, muss wieder bei Punkt 2A) begonnen werden, indem man logische Fehler im Algorithmus beseitigt.

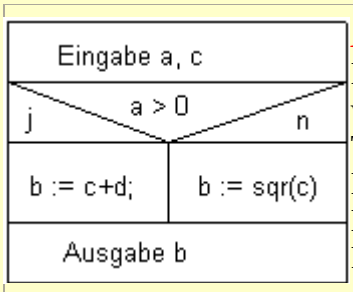
Und schließlich kann auch die gesamte Schrittfolge als zyklische Struktur begriffen werden: kaum ist die Version 1.0 eines Programms ausgeliefert, werden bisher unerkannte "Bugs" entdeckt und Wünsche zur Erweiterung der Funktionalität des Programms laut - also neue Problemformulierung, Problemanalyse, ... und schon existiert die Version 1.1 usw.

1) Problemformulierung

"Es ist ein Programm zu erstellen, das folgendes leistet: ... "
So oder so ähnlich beginnt die Formulierung dessen, was das Programm im ganzen realisieren soll. Man wird mehr oder weniger klare Vorstellung zu notwendigen Eingabewerten, zum Erscheinungsbild des Programms, zur Art der Ergebnisausgabe, zum potentiellen Nutzerprofil etc. vorfinden.

Die nachfolgenden beiden Schritte verlaufen im Prinzip parallel - die Programmierhandlung springt zwischen algorithmischer und visueller Seite hin und her, weil der algorithmische Aufbau des Programms bestimmte Komponenten einer Nutzeroberfläche bedingt bzw. die gewünschte Oberflächenstruktur Einfluss auf die Modularisierung der Lösungsalgorithmen haben kann.

2A) Algorithmischer Problemlösungsprozess - Algorithmische Seite -



I) Problemanalyse

Das Gesamtproblem wird in überschaubare Teilprobleme zerlegt - Modularisierung und Modellierung der Problemsituation.

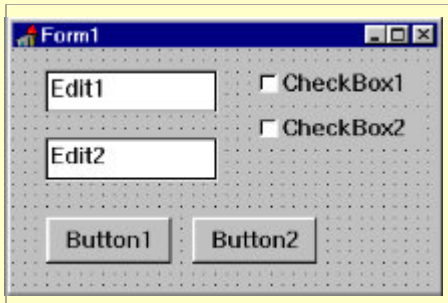
II) Entwurf von Lösungsalgorithmen für die Teilprobleme

Die Lösungswege dazu werden zunächst in abstrakter Weise beschrieben. Je nach Komplexität kann dies in verbaler Form geschehen (Eindeutigkeit!) oder es kommen grafische Beschreibungsformen, z.B. Struktogramme oder Programmablaufpläne zum Einsatz.
Es wird noch nicht mit konkreter Programmiersprache gearbeitet.

III) Synthese der Teillösungen

Nunmehr werden die entwickelten Teillösungen zur Gesamtlösung verknüpft und notwendige E/A-Funktionen zwischen den einzelnen Modulen

2V) Benutzeroberfläche erstellen und Eigenschaften der Objekte festlegen - Visuelle Seite -



Mit zunehmendem Voranschreiten im algorithmischen Problemlösungsprozess wächst auch die Vorstellung über Beschaffenheit und Funktionalität der Benutzeroberfläche.

Erstellt wird diese durch Anordnung aller notwendigen Komponenten (Textfelder, Optionsfelder, Eingabefelder, Schalter usw.) auf dem Formular.

Die Auswahl der Komponenten erfolgt über die Komponentenpalette. Dabei werden mit Hilfe des Objektinspektors die Eigenschaften der Komponenten festgelegt, z.B. Größe, Farbe,

festgelegt.	Bezeichnung der Schaltflächen, Text- und Eingabefelder. **Spätestens an dieser Stelle sollte das Projekt erstmals gespeichert werden!**

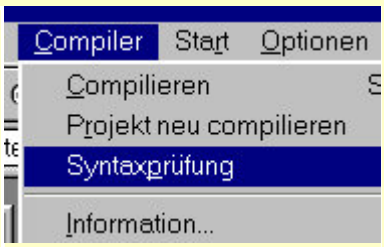
4) Ereignisbehandlung codieren
- "Hochzeit" von Algorithmus und Programmoberfläche -

Zunächst werden über den Objektinspektor diejenigen Ereignisse ausgewählt, die für den späteren Programmablauf von Bedeutung sein werden (z.B. Klicken eines Schalters).
In einer zugehörigen Ereignisbehandlungsroutine wird dann im Quelltexteditor in der Programmiersprache Pascal beschrieben, wie die jeweiligen Komponenten auf das betreffende Ereignis reagieren sollen.

Dabei werden also die unter 3A) gefundenen Lösungsalgorithmen in Programmiersprache "übersetzt" und die Ergebnisausgabe wiederum über Komponenten realisiert.

5) Test-Korrektur-Durchläufe

I) Syntaxprüfung:	Vorm ersten Start sollte unbedingt die syntaktische Korrektheit der eingegebenen Pascal-Anweisungen getestet und ggf. korrigiert werden. Anschließend wird das Projekt erneut gespeichert!
II) Resultatsprüfung:	Das syntaktisch korrekte Programm wird nun compiliert und gestartet. Um die Richtigkeit seiner Resultate hinreichend abzusichern, muss es mit verschiedenen Beispieleingaben getestet werden. Treten während des Tests Fehler auf, sind die oben genannten Schritte zu wiederholen.



6) Sicherung der Dateien, Dokumentation und Nutzung des Programms

In einem eigens dafür angelegten Verzeichnis (z.B. im Projekte-Ordner) werden abschließend alle für eine spätere Weiterbearbeitung des Programmprojektes nötigen Dateien gesichert. Die beim Compilieren entstandene Programmdatei (.exe) kann nunmehr unabhängig von Delphi unter Windows genutzt werden.

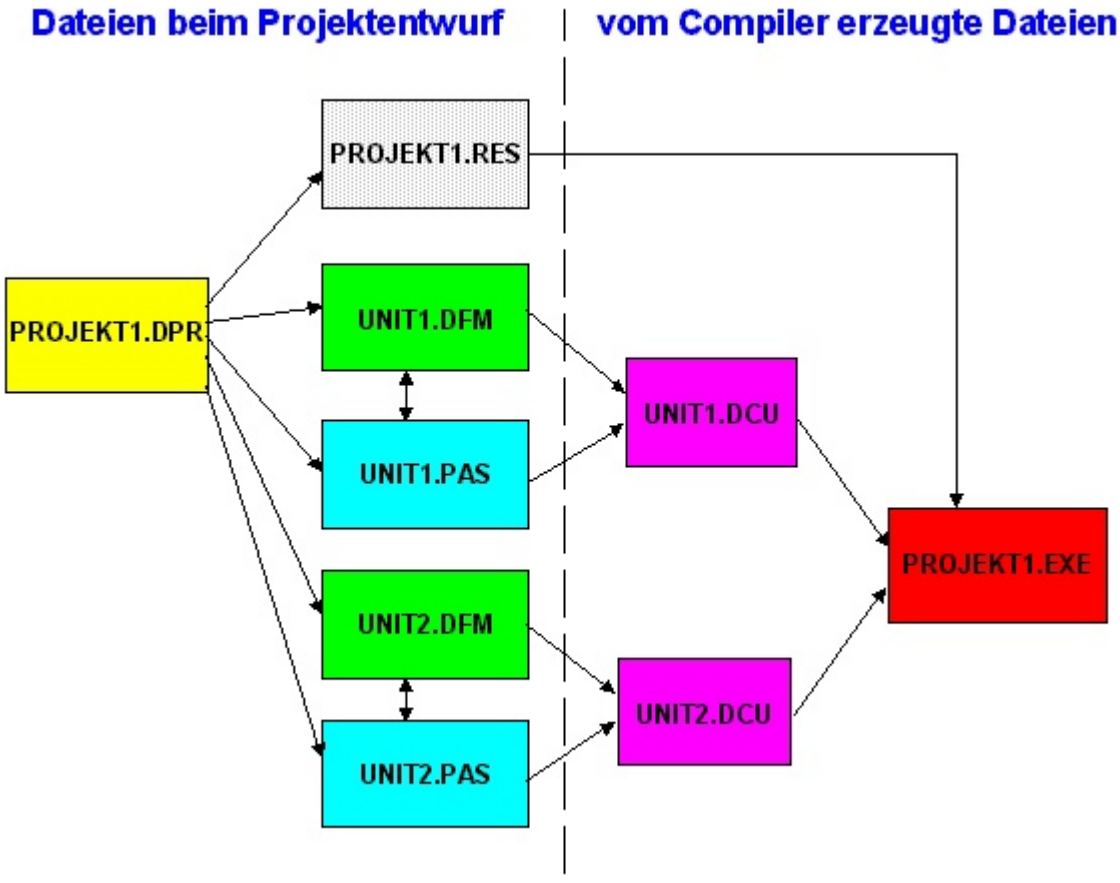
2.4. Projektverwaltung unter Delphi

Ein Delphi-Projekt ist eine Sammlung aller Dateien, die zusammen eine Delphi-Anwendung auf dem Entwicklungssystem ergeben. Einige dieser Dateien werden im Entwurfsmodus erzeugt, andere werden bei der Compilierung des Projekt-Quelltextes angelegt.

***Merke:** Jedes Projekt sollte unbedingt in einem separaten Verzeichnis gespeichert werden.*

2.4.1. Beispiel für die Dateistruktur eines Projektes

Ein Delphi-Projekt namens Projekt1 bestehe aus zwei Formularen:



2.4.2. Dateien, die in der Entwicklungsphase erzeugt werden:

Dateinamen-erweiterung	Definition	Zweck / Bemerkungen
.DPR	***Projektdatei***	➢ Pascal-Quelltext für die Hauptprogrammdatei des Projektes, ➢ enthält den Standardnamen des Projektes, ➢ verzeichnet alle Formular- und Unit-Dateien im Projekt und enthält den Initialisierungscode ➢ **wird von Delphi verwaltet und sollte nicht manuell verändert werden!**
.DFM	***Grafische Formulardatei***	➢ Quelltext, der die Entwurfseigenschaften eines Formulars des Projekts enthält, ➢ für jedes projektzugehörige Formular wird beim ersten Speichern des Projekts eine .DFM Datei zugleich mit der entsprechenden .PAS Datei angelegt. ➢ **Datei selbst wird von Delphi verwaltet während die Entwurfseigenschaften des Formulars vom Programmierer über den Objektinspektor eingestellt werden!**
.PAS	***Unit-Quelltext*** (in Pascal)	➢ wichtigste Bausteine für den Programmablauf! ➢ Für jedes Formular wird automatisch eine zugehörige Unit erzeugt, die alle Deklarationen und Prozeduren (Methoden) für die vom Formular auslösbaren Ereignisse enthält. ➢ **nur in diese Dateien wird vom Programmierer der eigentliche Programmquelltext geschrieben!**
.RES	***Compiler-Ressourcendatei***	➢ Binärdatei für vom Projekt zu verwendende äußere Ressourcen ➢ **wird von Delphi verwaltet**

Zur Datensicherung bzw. zur Speicherung von Systemeinstellungen erzeugt Delphi noch weitere Dateiarten, die hier nicht aufgeführt sind.

Um ein Projekt jedoch zwecks späterer Weiterbearbeitung zu sichern, genügt es (für den Einsteiger), alle zum Projekt gehörenden Dateien mit den oben aufgeführten Erweiterungen zu sichern.

2.4.3. Vom Compiler erzeugte Projektdateien:

Dateinamen-erweiterung	Definition	Zweck / Bemerkungen
.DCU	***Unit-Objekt-Code***	➢ während der Compilierung wird automatisch aus jeder .PAS-Datei und der zugehörigen .DFM-Datei eine entsprechende .DCU Datei erzeugt, die alle Eigenschaften und Methoden eines Formulars im Binärcode enthält.
.EXE	***Compilierte ausführbare Programmdatei***	➢ vertriebsfähige Programmdatei, die alle für das Programm nötigen .DCU-Dateien enthält.

Soll also das fertige Delphi-Programm auf andere Rechner übertragen werden, genügt es, die entsprechende .EXE Datei dorthin zu kopieren. Dabei müssen diese Rechner natürlich unter Windows laufen und die nötigen Delphi-Ressourcen besitzen.

2.4.4. Empfohlene Vorgehensweise im Unterricht:

Das nachfolgende Szenario geht davon aus, dass der Unterricht in einem vernetzten Computerkabinett durchgeführt wird, wobei jedem Schüler ein Serverlaufwerk zum Lesen (bei uns P:) und ein Laufwerk zum Schreiben (bei uns U:) zur Verfügung steht. Auf diesem Laufwerk werden zur Entwicklungszeit alle zum Delphi-Projekt gehörenden Dateien geführt.

2.5. Grundlegendes zur Syntax

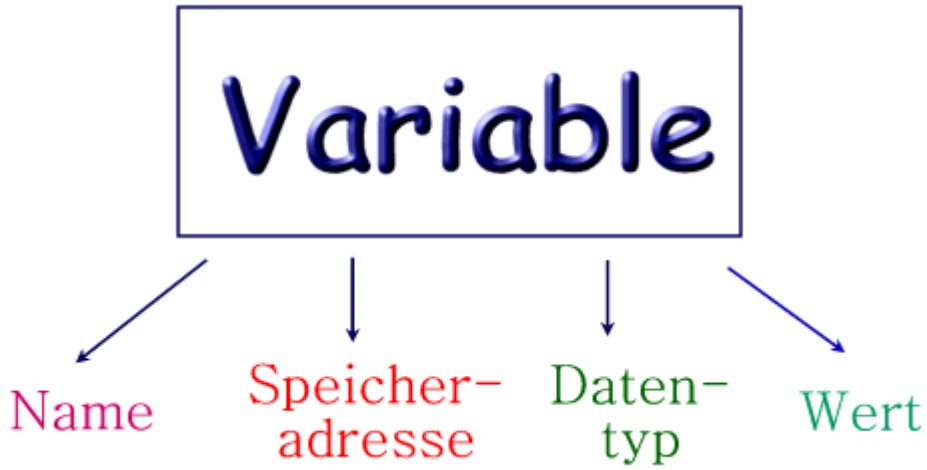
Wie viele andere Programmiersprachen werden mehrere Befehle durch Semikolon (;) getrennt.
Besonders im Zusammenhang mit "else" gibt es hier aber Unterschiede.
Die Delphi-Sprache unterscheidet nicht zwischen Groß- und Kleinschreibung. Während "meine_variable" und "Meine_Variable" in C/C++ unterschiedlich sind, sind sie für die Delphi-Sprache gleich. In dieser Hinsicht braucht der Entwickler also nicht so viel Disziplin beim Programmieren. Leerzeichen (natürlich nicht innerhalb eines Bezeichners) und Leerzeilen können verwendet werden, um die Übersichtlichkeit im Code zu erhöhen.

Das heißt natürlich nicht, dass er machen kann, was er will. Die Delphi-Sprache ist nämlich für ihre Typstrenge bekannt. Im Gegensatz zu Visual Basic muss eine Variable vor ihrer Verwendung in einem bestimmten Bereich deklariert werden. Ist ihr somit einmal ein Typ zugeordnet, kann sie keine Werte eines anderen Typs aufnehmen, nur Werte des gleichen Typs oder eines

Untertyps. Auch Zuweisungen zwischen Variablen unterschiedlichen Typs lassen sich häufig nur über Konvertierfunktionen (wie z.B. IntToStr) bewerkstelligen.

2.6. Variablen

In der Mathematik werden Variablen im Allgemeinen als Platzhalter für mathematische Objekte (Zahlen, Größen, Funktionen etc.) benutzt. Für den Informatiker stellt sich der Begriff "Variable" komplexer dar. Im Wesentlichen wird eine Variable durch vier Merkmale charakterisiert (siehe Abbildung).



Hießen die Variablen in Mathe meist x oder y, sollte ein Programmierer solche Variablen nicht verwenden. Sie erschweren die Lesbarkeit des Quellcodes sehr. Diese Erfahrung wird jeder schon einmal gemacht haben, der ein Programm mehrere Monate liegen gelassen hat, um es dann erneut anzugehen.
Wichtige Regel also: Variablen selbsterklärende Namen geben.
Groß- und Kleinschreibung ist nicht von Bedeutung, jedoch dürfen nur Buchstaben (keine Umlaute!), Zahlen und der Unterstrich verwendet werden. Der Variablenname muss mit einem Buchstaben beginnen.

2.7. Datentypen

Bevor eine Variable verwendet werden kann, sollte man sich darüber im Klaren sein, welche Werte sie aufnehmen sollen.
Variablen sind also Platzhalter oder "Container" für einen Wert, der Platz im Arbeitsspeicher belegt; der Datentyp ist dagegen der Bauplan, nach dem die Variable erstellt wird.
Folgendes sind die grundlegenden Datentypen in Delphi:

Typ	**Wertebereich**	**Beispiel**	
		Deklaration	Zuweisung
Integer (ganze Zahlen)	-2147483648..2147483647	**var** zahl: integer;	zahl:=14;
Real (Gleitkommazahlen)	5.0×10^{324} .. 1.7×10^{308}	**var** zahl: real;	zahl:=3.4;
String (Zeichenketten)	ca. 2^{31} Zeichen	**var** text: **string**;	text:='Hallo Welt!';
Char (Zeichen)	1 Zeichen	**var** zeichen: char;	zeichen:='a';
Boolean (Wahrheitswert)	true, false	**var** richtig: boolean;	richtig:=true;

Dies sind die Standardtypen, die generell verwendet werden können. Vor allem für Zahlen gibt es jedoch noch weitere Typen. Wenn z. B. sicher ist, dass nur ganze Zahlen von 1 bis 50 gespeichert werden sollen, so kann statt Integer auch der Typ Byte verwendet werden, der nur die Zahlen von 0 bis 255 aufnehmen kann. Das Gleiche gilt für reelle Zahlen. Die weiteren Typen sind unter "Reelle Typen" bzw. "Integer-Typen" in der Hilfe aufgeführt.
Bei Strings gibt es einige grundlegende Unterschiede, die hier aber nicht behandelt werden.

2.8. Deklaration

Man kann eine Variable nicht einfach verwenden. Vorher muss sie dem Compiler bekannt gemacht werden, damit er beim Kompilieren entsprechende Typprüfungen durchführen kann. Diese Bekanntmachung nennt man Deklaration. Vor einer Deklaration wird das reservierte Wort **var** geschrieben. Als erstes kommt der Name der Variablen, hinter einem Doppelpunkt der Typ. Mehrere Variablennamen vom gleichen Typ können in einer Zeile, durch Kommas getrennt, stehen.

Beispiel:

```
var zahl1, zahl2, zahl3: integer;
    ergebnis: real;
    text, eingabe: string;
```

An folgenden Stellen im Code ist das möglich, je nach Gültigkeitsbereich:

2.8.1. Globale Variablen

Bei manchen Programmierern sind sie verpöhnt, trotzdem sind sie möglich: globale Variablen. Ihr Wert ist in der gesamten Unit verfügbar und in allen Units, die diese einbinden. Die Deklaration erfolgt am Anfang der Unit:

```
unit Unit1;

interface

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

type
  TForm1 = class(TForm)
  private
    { Private-Deklarationen }
  public
    { Public-Deklarationen }
  end;

var
  Form1: TForm1;
  einezahl: integer;     //Diese Variable gilt in der ganzen Unit
und in allen Units,
                         //die diese Unit einbinden

implementation

{$R *.DFM}

var eine_andere_zahl: real; //Diese Variable gilt nur in dieser
Unit
```

2.8.2. Lokale Variablen

Das Gegenstück zu globalen Variablen sind die lokalen. Hierbei wird eine Variable zu Beginn einer Prozedur oder Funktion deklariert. Sie kann somit nur innerhalb dieses Abschnitts verwendet werden. Wird die Prozedur/Funktion verlassen, dann wird der Speicher für die Variablen wieder freigegeben, d. h. auf die Werte kann nicht mehr zugegriffen werden.
So könnte eine lokale Variablendeklaration aussehen:

```
procedure IchMacheIrgendwas;
var text: string;
begin
  ... //Irgendwas Sinnvolles
end;
```

2.9. Initialisierung

Bevor auf eine Variable zugegriffen wird, sollte sie initialisiert werden, es sollte ihr also ein Anfangswert zugewiesen werden. Der Delphi-Compiler weist Integer-Werten zwar den Wert 0 und Strings einen leeren String ('') zu; alle anderen Variablen enthalten jedoch meistens irgendwelche Zufallswerte.

2.10. Zuweisungen

Zuweisung von Werten an eine Variable erfolgt in Pascal durch die Symbolfolge := ("ergibt sich aus"). Im Gegensatz zur Mathematik ist deshalb auch Folgendes möglich:

```
x := x+1;
```

Das Laufzeitsystem berechnet hier die Summe von x und 1 und legt das Ergebnis dann wieder in x ab. x ergibt sich aus dem bisherigen x plus 1. Kurz: x wird um 1 erhöht. Für diesen Fall gibt es auch eine eigene Prozedur: inc(x).

2.11. Typumwandlung

Im Gegensatz zu manch anderen Sprachen ist Delphi bei Typen sehr streng. Es ist also nicht möglich einer Integer-Variable eine Gleitkommazahl-Variable zuzuweisen. Dafür stehen eine große Auswahl an Konvertierungsfunktionen zur Verfügung:

von	nach	Funktion	Beispiel

Integer	Real	kein Problem, einfache Zuweisung	**var** zahl1: integer;
			zahl2: real;
			begin
			zahl2 := zahl1;

Real	Integer	Möglichkeiten:
		- Nachkommastellen abschneiden (trunc)
		- kaufm. Runden (round)
		- aufrunden (ceil, Unit Math)
		- abrunden (floor, Unit Math)
			var zahl1: real;
			zahl2: integer;
			begin
			zahl2 := trunc(zahl1);
			zahl2 := round(zahl1);

Integer	String	IntToStr	**var** textzahl: **string**;
			zahl: integer;
			begin
			textzahl := IntToStr(zahl);

Real	String	FloatToStr
		FloatToStrF
		(siehe Tipps & Tricks)
			var textzahl: **string**;
			zahl: real;
			begin
			textzahl := FloatToStr(zahl);

String	Integer	StrToInt
		StrToIntDef
			var textzahl: **string**;
			zahl: Integer;
			begin
			zahl := StrToInt(textzahl);

String	Real	StrToFloat	**var** textzahl: **string**;
			zahl: real;
			begin
			zahl := StrToFloat(textzahl);

String	Char	Zugriff über Index	**var** text: **string**;

		(1 ist erstes Zeichen)	zeichen: char;
			begin
			zeichen := text[1];

Char	String	kein Problem, einfache Zuweisung	**var** zeichen: char;
			text: **string**;
			begin
			text := zeichen;

2.12. Besonderheiten bei Strings

Stringtexte werden immer in einfachen Anführungszeichen (#-Taste) geschrieben. Um ASCII-/ANSI-Zeichen darzustellen verwendet man die Funktion chr; chr(169) stellt z. B. das Copyright-Symbol dar.
Hat man mehrere String-Variablen, die man aneinanderhängen will, verwendet man das +-Zeichen:

```
text1 := 'Hallo';
text2 := 'Welt';
text3 := text1 + ' ' + text2 + '!!';
//Ausgabe: Hallo Welt!!
```

Stringvergleiche kann man wie bei Zahlen mit = (gleich), < (kleiner), > (größer) oder <> (ungleich) durchführen. Alternativ gibt es auch Pascal-Funktionen dafür wie StrComp, StrLComp, StrIComp usw.
Stringmanipulationen werden mit den Funktionen **delete** (Löschen einer bestimmten Anzahl von Zeichen in einem String), **copy** (Kopieren bestimmter Zeichen eines Strings in einen anderen) und **pos** (Ermitteln der Position eines bestimmten Zeichens innerhalb eines Strings) durchgeführt.
Weitere Informationen zu diesen Funktionen finden sich in der VCL- bzw. Object Pascal-Hilfe.

2.13. Unterbereichstypen

Einen weiteren Datentyp, der eigentlich gar kein eigener Datentyp ist, gibt es noch, nämlich den Unterbereichstyp. Hierüber kann man Variablen z. B. zwar den Typ Integer zuordnen, aber nicht den kompletten Definitionsbereich, sondern nur ein Unterbereich davon:

```
var kleineZahl: 0..200;
```

Der Variablen kleineZahl lassen sich jetzt nur ganze Zahlen zwischen 0 und 200 zuweisen.
Ähnlich lassen sich auch Strings begrenzen:

```
var kleinerString: string[10];
```

Diese Variable kann nur zehn Zeichen aufnehmen, es handelt sich nun um einen ShortString, der auch Nachteile gegenüber einem "normalen" AnsiString hat.

2.14. Konstanten

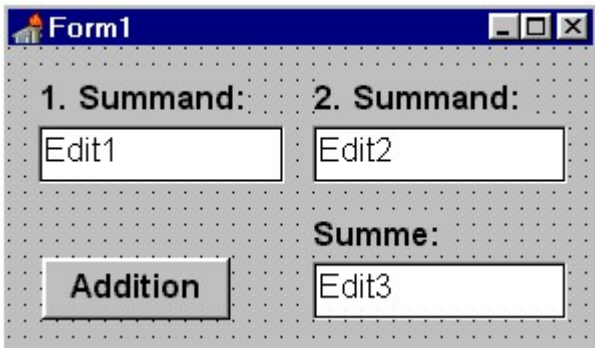
Konstanten sind letztendlich Variablen, die innerhalb des Programms jedoch nur ausgelesen, nicht überschrieben werden dürfen.

Deklariert werden sie an den gleichen Stellen wie Variablen, allerdings mit dem Schlüsselwort **const**, einem Gleichheitszeichen und ohne Angabe eines Datentyps:

```
const version = '1.23';
```

2.15. Struktur einer Unit unter Delphi

Die nachfolgend aufgelistete Unit bezieht sich auf das abgebildete einfache Formular - Addition zweier Zahlen.



Bemerkenswert (und beruhigend!) ist die Tatsache, dass alle in der Unit **schwarz geschriebenen Zeilen** während der visuellen Formularerstellung **automatisch von Delphi erzeugt** werden.
Nur die wenigen **rot gekennzeichneten Zeilen** der Prozedur zur Realisierung der Addition sind wirklich **vom Programmierer zu tippen**.
Dies untersetzt die eingangs getroffene Feststellung, dass auch Programmier-Einsteiger ohne nennenswerte Vorkenntnisse unter Delphi schnell zu "greifbaren" Ergebnissen - sprich ablauffähigen Programmen - gelangen und einfache Algorithmen implementieren können. Es erscheint daher sinnvoll, erst nach einigen Projektübungen zum praktischen Teil der Programmierung die Lernenden sukzessive mit der Struktur einer Unit und der Funktion ihrer Bestandteile vertraut zu machen.

Strukturelemente der Beispiel-Unit	Benennung / Bedeutung
`unit Utest1;`	**Kopfzeile der Unit:** enthält deren Dateinamen
`interface`	**Interface-Teil**: bestimmt, was in der Unit von außen zugänglich ist

`uses`
`SysUtils, WinTypes, WinProcs, Messages,`
`Classes, Graphics, Controls, Forms,`
`Dialogs, StdCtrls;`
	Uses-Teil: Benennung von Units (Prozedurbibliotheken), die von der aktuellen Unit verwendet werden

`type`
`TForm1 = class(TForm)`
`  Edit1: TEdit;`
`  Edit2: TEdit;`
`  Edit3: TEdit;`
`  Label1: TLabel;`
`  Label2: TLabel;`
`  Label3: TLabel;`
`  Button1: TButton;`
`  procedure Button1Click(Sender:`
`TObject);`
`end;`
	Typdeklaration des Formularobjektes:
	
	enthält alle Komponenten, die auf dem Formular angeordnet sind sowie die zum Formular gehörenden Prozeduren

`var` `  Form1: TForm1;`	**Deklaration globaler Variablen** hier: Variable (Objekt) Form1 vom Typ TForm1 (Objekttyp)
`implementation`	**Implementationsteil**: enthält Programmteil der Prozeduren
`{$R *.DFM}`	**Formulardatei wird an die Unit gebunden**
`procedure TForm1.Button1Click(Sender: TObject);`	**Kopfzeile der Prozedur**
`var a, b, c: real;`	**Deklaration lokaler Variablen:** diese gelten nur in der jeweiligen Prozedur

`begin`
`  a := StrToFloat(Edit1.Text);`
`  b := StrToFloat(Edit2.Text);`
`  c := a+b;`
`  Edit3.Text := FloatToStr(c);`
`end;`
	Anweisungsteil der Prozedur
	
	Algorithmus zur Ereignisbehandlung
	hier: Zugriff auf lokale Variablen

`end.`	**Schlusszeile der Unit** - Ende.

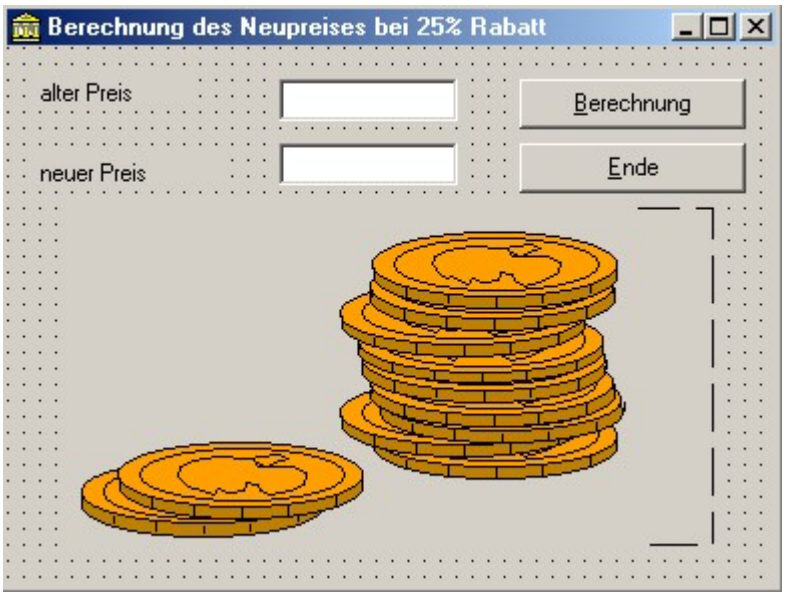
---

3. Übungen

3.1. Übungen zu Sequenzen

Übung: Erstellen Sie das obige Projekt (siehe 2.15.) und speichern Sie alle zugehörigen Dateien in einem Ordner mit dem Namen *Addition*! Sie benötigen für das Projekt Label, Edit-Felder und Button!
Erweitern Sie das Formular um einen Reset-Button!
Erweitern Sie das Programm auch um die Multiplikation!

Übung: Erstellen Sie ein Projekt mit dem Namen *Neupreis*, bei dem ein alter Preis eingegeben wird, und bei Klick auf die Schaltfläche *Berechnung* der neue Preis ausgegeben wird.
(Tipp: Wenn man im Objektinspektor bei der Caption-Eigenschaft nicht nur *Berechnung*, sondern *&Berechnung* eingibt, dann wird der Buchstabe, vor dem das & steht, unterstrichen und kann im laufenden Programm auch mit der Tastenkombination <ALT + unterstrichener Buchstabe> benutzt werden.)



Der Preis soll mit zwei Nachkommastellen angegeben werden.
Um das image einzufügen, muss in der Komponentenpalette das Register *Zusätzlich* geöffnet und das image-Symbol ausgewählt werden. Im Objektinspektor wird nun die Eigenschaft picture bearbeitet, indem man hier das richtige Bild aus dem richtigen Ordner sucht. Das Bild Geld.bmp befindet sich im Ordner P:\TEXTE\KLASSE10\DELPHI\DOWNLOADS. Das Bild soll zu Beginn nicht sichtbar sein. Dazu muss die visible-Eigenschaft auf *false* gestellt werden.

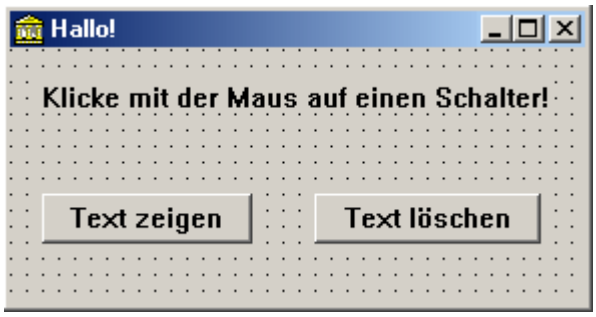
Die Click-Procedur für den Berechnung-Button sieht dann folgendermaßen aus:

```
procedure TForm1.Button1Click(Sender: TObject);
var a,b:real;
begin
  a:=strtofloat(edit1.text);
  b:=a*0.75;
  edit2.text:=floattostrF(b,ffFixed,5,2);
  image1.visible:=true
end;
```

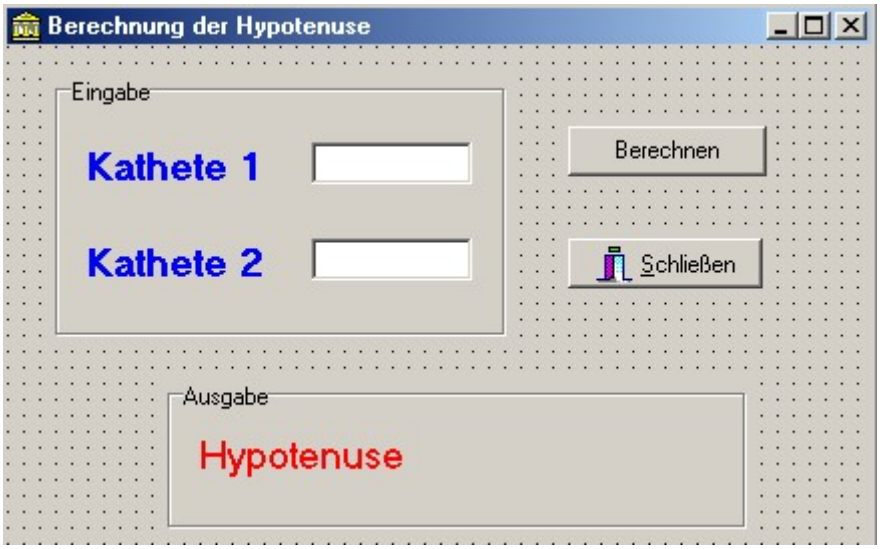
Die Click-Procedur für den Ende-Button sieht dann folgendermaßen aus:

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

Übung: Erstellen Sie ein Projekt mit dem Namen *Hallo* mit folgendem Aussehen: Der Text in dem Label soll bei Klick auf die Button jeweils gezeigt bzw. gelöscht werden. Die Click-Procedur für die Button sieht so aus:




```
procedure
TFormHallo.ButtonZeigeClick(Sender:
TObject);
begin
  LabelText.Caption := 'Klicke mit der Maus auf einen Schalter!';
end;

procedure TFormHallo.ButtonLoescheClick(Sender: TObject);
begin
  LabelText.Caption := '';
end;
```



Übung: Erstellen Sie ein Projekt mit dem Namen *Pythagoras*, bei dem die Länge der beiden Katheten gegeben sein soll und die Länge der Hypotenuse zu berechnen ist. (Für $c := \sqrt{a^2 + b^2}$ schreibt man `c:=sqrt(a*a+b*b))` Arbeiten Sie mit Group-Boxen und Labeln. Der Schließen-Button ist ein Bitbtn aus dem Register *Zusätzlich* aus der Komponentenpalette. Die Kind-Eigenschaft muss auf bkclose gestellt werden, dann schließt ein Klick auf die Schaltfläche das Fenster ohne weitere Eingabe von Quelltext.

Die Click-Procedur für den Berechnen-Button sieht dann folgendermaßen aus:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  a:=strtofloat(edit1.text);
  b:=strtofloat(edit2.text);
  c:=sqrt(a*a+b*b);
..label4.caption:='= '+floattostrF(c,ffFixed,7,4);
end;
```

Übung: Das nächste Programmierobjekt soll das Zusammenspiel von Objekteigenschaften, Ereignissen und Methoden verdeutlichen, indem an einem Edit-Feld bzw. einem Button über Button-Click-Ereignisse bestimmte Eigenschaften zur Laufzeit per Wertzuweisung geändert werden.

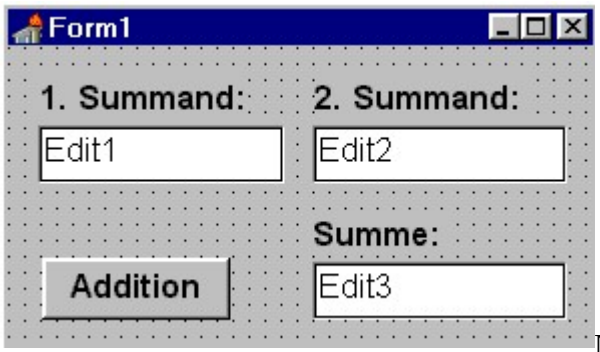
Zur Zeiteinsparung finden Sie das untenstehende Formular als Vorgabe im Ordner P:\TEXTE\KLASSE10\DELPHI\DOWNLOADS\EDITFELD. Kopieren Sie den gesamten Ordner in Ihren Ordner auf Laufwerk U:\, starten Sie die Datei *editfeld.exe* und probieren Sie!

Deklariert werden sie an den gleichen Stellen wie Variablen, allerdings mit dem Schlüsselwort **const**, einem Gleichheitszeichen und ohne Angabe eines Datentyps:

```
const version = '1.23';
```

2.15. Struktur einer Unit unter Delphi

Die nachfolgend aufgelistete Unit bezieht sich auf das abgebildete einfache Formular - Addition zweier Zahlen.



Bemerkenswert (und beruhigend!) ist die Tatsache, dass alle in der Unit **schwarz geschriebenen Zeilen** während der visuellen Formularerstellung **automatisch von Delphi erzeugt** werden.
Nur die wenigen **rot gekennzeichneten Zeilen** der Prozedur zur Realisierung der Addition sind wirklich **vom Programmierer zu tippen**.
Dies untersetzt die eingangs getroffene Feststellung, dass auch Programmier-Einsteiger ohne nennenswerte Vorkenntnisse unter Delphi schnell zu "greifbaren" Ergebnissen - sprich ablauffähigen Programmen - gelangen und einfache Algorithmen implementieren können. Es erscheint daher sinnvoll, erst nach einigen Projektübungen zum praktischen Teil der Programmierung die Lernenden sukzessive mit der Struktur einer Unit und der Funktion ihrer Bestandteile vertraut zu machen.

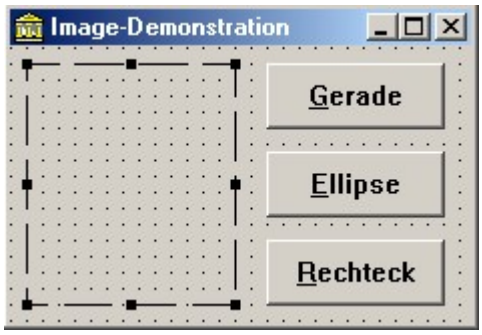
Strukturelemente der Beispiel-Unit	Benennung / Bedeutung
`unit Utest1;`	**Kopfzeile der Unit:** enthält deren Dateinamen
`interface`	**Interface-Teil**: bestimmt, was in der Unit von außen zugänglich ist

`uses`
`SysUtils, WinTypes, WinProcs, Messages,`
`Classes, Graphics, Controls, Forms,`
`Dialogs, StdCtrls;`
	Uses-Teil: Benennung von Units (Prozedurbibliotheken), die von der aktuellen Unit verwendet werden

`type`
`TForm1 = class(TForm)`
`  Edit1: TEdit;`
`  Edit2: TEdit;`
`  Edit3: TEdit;`
`  Label1: TLabel;`
`  Label2: TLabel;`
`  Label3: TLabel;`
`  Button1: TButton;`
`  procedure Button1Click(Sender:`
`TObject);`
`end;`
	Typdeklaration des Formularobjektes:
	
	enthält alle Komponenten, die auf dem Formular angeordnet sind sowie die zum Formular gehörenden Prozeduren

```
end;

procedure TForm1.Button18Click(Sender: TObject);
begin
  Button18.width:=button18.width+10;
end;

procedure TForm1.Button17Click(Sender: TObject);
begin
  form1.color:=clred;
end;
```

Übung: Erstellen Sie ein Projekt mit dem Namen *Image*, das nebenstehendes Aussehen hat:
Bei Klick auf die entsprechenden Button soll jeweils das Gewünschte erscheinen.
Die Click-Proceduren sehen dann folgendermaßen aus:



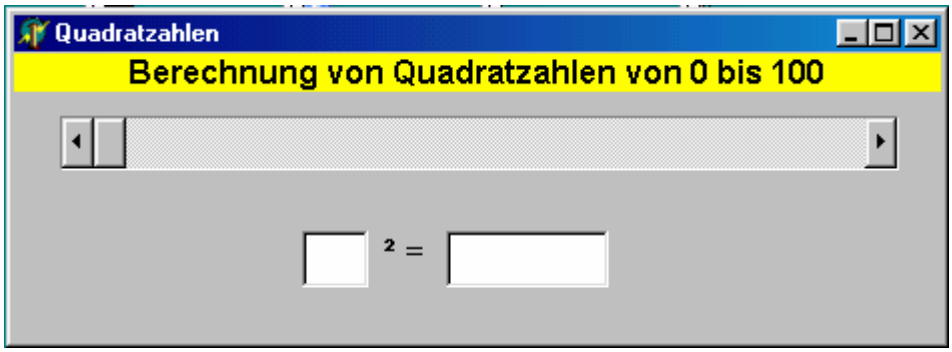

```
procedure
TFormImage.ButtonGeradeClick(Sender:
TObject);
begin
   WITH Image.Canvas DO BEGIN
        MoveTo (0,0);
        LineTo (Image.Width,Image.Height);
   END;
end;

procedure TFormImage.ButtonEllipseClick(Sender: TObject);
begin
   Image.Canvas.Ellipse (0,0,Image.Width,Image.Height);
end;

procedure TFormImage.ButtonRechteckClick(Sender: TObject);
begin
   Image.Canvas.Rectangle (10,10,Image.Width-10,Image.Height-10);
end;
```

Versuchen Sie nun, ein Haus (Wer das nicht kann,...) zu erzeugen!

Übungen:
Gesucht sind Delphi-Programme, die folgendes leisten

1. Die Mitglieder eines Computerclubs wählen ihren Vorsitzenden. Drei Kandidaten stehen zur Wahl. Nach Eingabe ihrer Namen und der erzielten Stimmenzahlen soll der Computer die Stimmenanteile (in Prozent) angeben.
2. Bauer Ignaz plant den Verkauf einer Wiese. Er will nach Eingabe von Länge, Breite und Quadratmeterpreis den Verkaufserlös - mit und ohne Mehrwertsteuer - wissen.
3. Der Geschäftsführer eines Betriebes sucht ein Programm zur Wochenlohnberechnung. Eingaben: Anzahl der Arbeitsstunden, Anzahl der Überstunden, Stundenlohn. Für Überstunden wird ein Zuschlag von 20% auf den Stundenlohn gewährt.
4. Banken benötigen zur Zinsberechnung die Anzahl der Tage zwischen zwei Tagesdaten. Ein Tagesdatum ist durch drei Zahlen für Tag, Monat und Jahr gegeben; jeder Monat hat 30, jedes Jahr 360 Tage (kaufmännische Methode).

5. Erstellen Sie einen Ordner *Zeit*. Der Nutzer soll eine Zeit in Stunden, Minuten und Sekunden eingeben. Die Gesamtzeit in Sekunden ist auszugeben, also z.B. Stunden = 2, Minuten = 10, Sekunden = 30 → Gesamtsekunden = 7830
6. Erstellen Sie einen Ordner *Quersumme*. Der Nutzer soll eine dreistellige Zahl eingeben. Die Quersumme ist zu berechnen!



Übung: Heute geht es um ein Projekt, bei dem das Quadrat einer Zahl zwischen 0 und 100 berechnet werden soll (z.B. $17^2 = 289$). Im Ordner P:\TEXTE\KLASSE10\DELPHI\DOWNLOADS\QUADRAT finden Sie die Datei *quadrat.exe*, die angibt, was wir heute erreichen wollen. Die Rechnung selbst ist sehr einfach, aber wir werden anhand dieses Beispiels den Umgang mit Bildlaufleisten kennen lernen. Deshalb zu Beginn etwas Theorie:

Bildlaufleisten (ScrollBar)

Es gibt zwei Sorten: horizontale und vertikale. Wir können sie verwenden, um Zahlen einzugeben, Lautstärken zu regeln oder um aus einer Optionsliste zu wählen. Zunächst fügt man eine ScrollBar auf dem Formular ein und wählt dann über die Eigenschaft *kind*, ob eine vertikale oder horizontale Bildlaufleiste angezeigt werden soll.

Beide Laufleistenypen bestehen aus drei Bereichen, die angeklickt oder gezogen werden können, um den Wert (scrollposition) zu ändern.

Klick auf **Endpfeile** bewegt die **scroll box** ein kleines Stück, Anklicken der **bar area** bewegt die scroll box ein großes Stück, ziehen der scroll box ermöglicht eine fließende Bewegung. Mit den Eigenschaften einer Bildlaufleiste kann man ihre Arbeitsweise vollständig festlegen.

Eigenschaften

Eigenschaft:	***Beschreibung:***
Max	Der größte Wert. (zwischen -32768 und 32767).
Min	Der kleinste Wert.
SmallChange	Schrittweite, um die sich der Wert ändert, wenn man auf die Endpfeile klickt.
LargeChange	Schrittweite, um die sich der Wert ändert, wenn man auf die bar area

	klickt
Left, Top, Width, Height, Enabled, Visible	wie bei den anderen Steuerelementen

Ereignisse
Wir benutzen zwei Ereignisse bei Bildlaufleisten:

Ereignis	***Beschreibung***
Change	Ereignis wird immer dann ausgeführt, wenn der Wert sich geändert hat.
Scroll	Ereignis wird ständig ausgeführt, wenn die scroll box bewegt wird.

Entwerfen Sie nun zunächst das Formular, fügen Sie bei der Bildlaufleiste als Min 0, als Max 100, als LargeChange 10 ein. Speichern Sie das Projekt in einem Ordner mit dem Namen *Quadrat*. Der Quelltext, der eingegeben werden muss, ist sehr kurz und beinhaltet die Variablenvereinbarung (die scrollbarposition ist ein integer-Wert) und die Rechnung:

```
procedure TForm1.ScrollBar1Change(Sender: TObject);
begin
  edit1.text:= inttostr(scrollbar1.position);
  edit2.text:= inttostr(scrollbar1.position * scrollbar1.position)
end;
```

Im ersten edit-Feld wird also nur der Wert der Bildlaufleiste, im zweiten der berechnete Wert angezeigt. Erweitern Sie Ihr Projekt um die Berechnung der dritten Potenz und der Wurzel! Speichern Sie!

3.2. Übungen zur einseitigen Alternative

Grundstruktur:

```
IF Bedingung THEN Anweisung;
```

Die Anweisung hinter THEN wird nur dann ausgeführt, wenn die vorgegebene Bedingung erfüllt ist.

3.2.1. Programm zur Rabattberechnung

a)

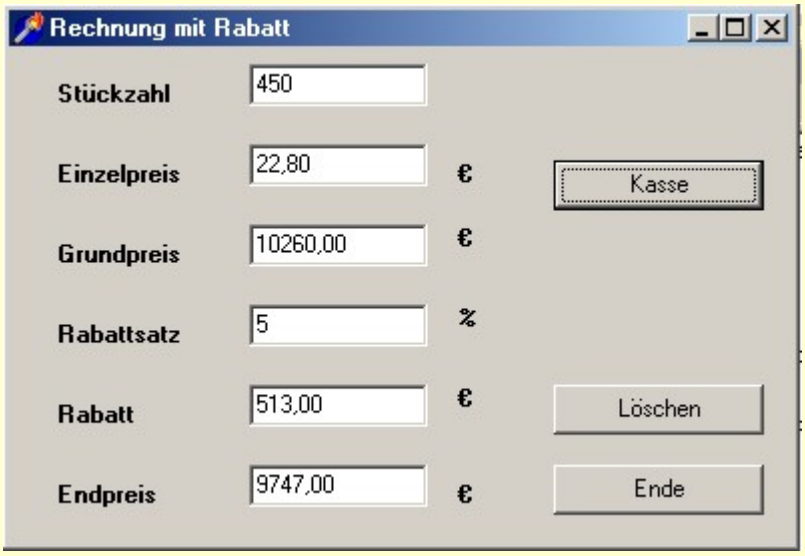
Gegeben sind die Real-Variablen *anzahl* und *einzelpreis*;

Zu berechnen sind *grundpreis*, *rabattsatz*, *rabatt*, und *endpreis*, wenn folgendes gilt:

> Grundsätzlich wird kein Rabatt gewährt.
> Falls die Anzahl gleich oder größer 100 beträgt, wird ein Rabattsatz von 5% angerechnet.

Notieren Sie die nötigen Programmzeilen!

b)



Entwerfen Sie ein Formular gemäß der nebenstehenden Abbildung und fügen Sie den unter 1. erarbeiteten Programmtext zur Ereignisbehandlung von ***Button1Click*** (Kasse) ein!
Beachten Sie, dass als Eingabevariablen nur *anzahl* (über Edit1.Text) und *einzelpreis* (über Edit2.Text) einzulesen sind.
Alle übrigen Textfelder dienen nur der Ausgabe!
Testen Sie mehrfach unter Berücksichtigung der bedingten Rabattvergabe!
Sichern Sie das Projekt in einem Ordner mit dem Namen *Rabatt*!

c)

Das Programm ist unter **Verwendung einseitiger Alternativen** mit folgenden gestaffelten Rabattvergaben zu erweitern:

Anzahl	Rabattsatz
0... 99	0%
100... 499	5%
500... 999	10%
1000...1499	15%
1500...1999	20%
ab 2000	30%

	Sichern Sie nun das erweiterte Projekt unter gleichem Namen!

3.2.2. Berechnung und Auswertung des Kraftstoffverbrauches

Es ist ein Programm zu entwickeln, welches nach Eingabe der gefahrenen Kilometer und verbrauchten Liter Kraftstoff den Kraftstoffverbrauch pro 100 km ermittelt und ausgibt.

Durch bedingte Anweisungen soll das Programm wie folgt reagieren:
- ***falls der Verbrauch > 8 l/100km, dann Ausgabe: 'Verbrauch ist zu hoch!'***
- ***falls der Verbrauch < 4 l/100km, dann Ausgabe: 'Verbrauch ist unrealistisch!'***

a)	Entwerfen Sie unter Delphi ein geeignetes Formular!
b)	Programmieren Sie eine ButtonClick-Ereignisbehandlung, welche obiger Aufgabe entspricht!
c)	Test / Korrektur / Sicherung im Ordner *Kraftstoff*!

3.3. Übungen zu zweiseitigen Alternativen und Verbundanweisungen

Grundstruktur der zweiseitigen Alternative:

```
IF Bedingung THEN Anweisung_1 ELSE Anweisung_2;
```

Die Anweisung hinter THEN wird ausgeführt, wenn die vorgegebene Bedingung erfüllt ist, andernfalls wird die Anweisung hinter ELSE ausgeführt.

Grundstruktur der Verbundanweisung:

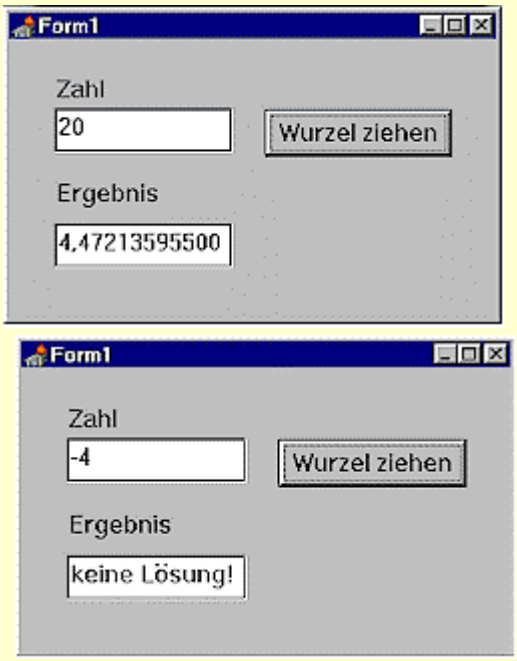
```
IF Bedingung THEN
  BEGIN
    Anweisung_1;
    Anweisung_2;
    ...
    Anweisung_n
  END
ELSE
   ...;
```

Soll in einer syntaktischen Struktur eine Folge von n Anweisungen genau so behandelt werden, als würde es sich nur um eine einzige Anweisung handeln, so verwende man **begin** und **end** im Sinne von mathematischen Klammern.

Zur Bewahrung der Übersicht in zunehmend komplexer werdenden Quelltexten ist es dringend anzuraten, sich unterordnende bzw.

eingeschachtelte Teilstrukturen nach rechts einzurücken.
Im Beispiel ordnet sich das "BEGIN" dem "IF-THEN" unter und die "Anweisung_1" wiederum dem "BEGIN" usw.

3.3.1. Quadratwurzel mit Prüfung auf negativen Eingabewert

Aufgabe:	Ein Delphi-Formular enthalte die Komponenten Edit1, Edit2 und Button1. Das Ereignis "Button1Click" soll folgende Reaktion hervorrufen: ***Ist die Zahl in Edit1 größer oder gleich Null, so wird deren Wurzel berechnet und anschließend in Edit2 als Text ausgegeben. Andernfalls soll in Edit2 eine Fehlermeldung ausgegeben werden.***
Formular:	

3.3.2. Kraftstoffverbrauch Variante II

Das Programm zur Berechnung und Auswertung des Kraftstoffverbrauches ist folgendermaßen zu erweitern:

1. Falls der Verbrauch im normalen Bereich liegt, also ***4 l/100km <= verbrauch <= 8 l/100km*** soll die Ausschrift erfolgen: ***"normaler Verbrauch"***.
2. Zur optischen Unterstützung der Ausgabe soll das entsprechende Edit-Feld bei

normalen Verbrauch grün, bei zu hohem Verbrauch rot und bei unrealistisch geringem Verbrauch gelb eingefärbt werden.

Unter Verwendung der zweiseitigen Alternative und der Verbundanweisung entwerfe man zunächst ein Struktogramm und erweitere danach das bereits vorhandenen "Benzinprogramm".

Ideen zur Erweiterung:

Die vorliegenden Gegebenheiten entsprechen einem "Allerweltsauto" mit 1300cm³ - Benzinmotor und 60 PS, decken also nur eine stark begrenzte Problemklasse ab.

- Ermitteln Sie Verbrauchsnormative für Pkws mit stärkerer und schwächerer Motorisierung, getrennt nach Benzin- und Dieselmotor.
- Modifizieren Sie danach die Sollwerte für Minimal- und Maximalverbrauch (bisher konstant 4 bzw. 8 l/100km) als Variablen mit entsprechenden prozentualen Ab- oder Zuschlägen für Motorart, Leistung und Hubraum.
- Nach Erweiterung der Oberfläche um die zusätzlichen Eingabekomponenten ist der Algorithmus entsprechend zu verfeinern und in das Programm zu implementieren.

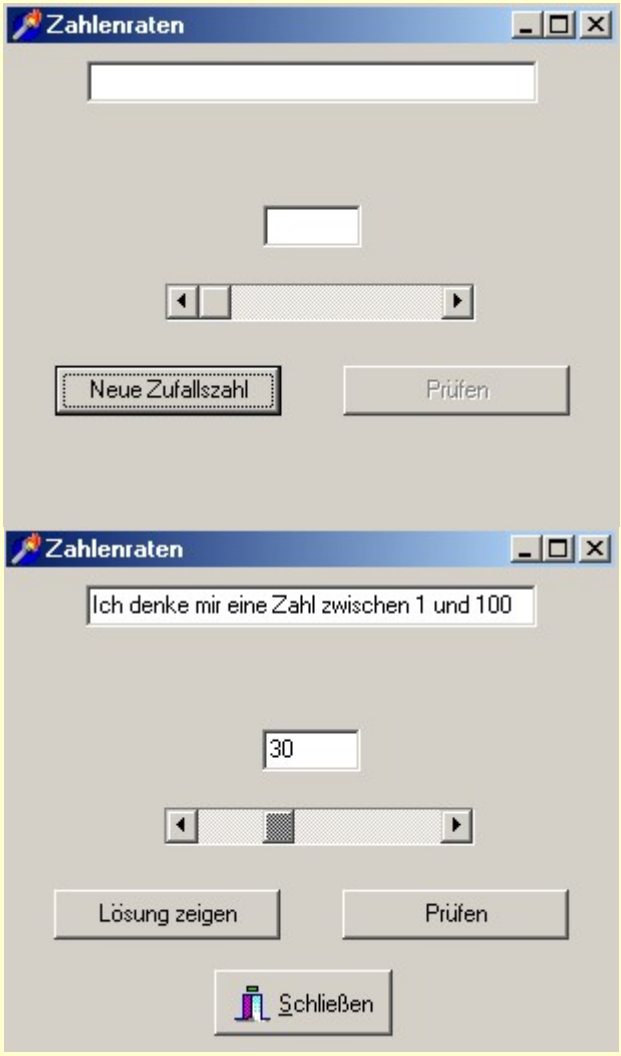
3.3.3. Zahlenraten

kurzer Überblick über Zufallszahlen:

Aufgabe:	Eine Zufallszahl aus dem Bereich 0 bis 9 soll erzeugt werden..
Formular:	

Quelltext:	`procedure TForm1.Button1Click(Sender: TObject);`
	`var x:integer;`
	`begin`
	`  randomize;`
	`  x:=random(10);`
	`  edit1.text:=inttostr(x);`
	`end;`
	
	Mit `randomize` wird der Zufallszahlengenerator initialisiert.
	`random(x)` mit x als integer-Wert gibt eine integer-Zahl mit 0 ≤ Zahl < x zurück, also im obigen Beispiel zwischen 0 und 9. Möchte man andere Zufallszahlen erhalten, kann man auch Rechnungen mit dem `random(x)`- Wert

	durchführen: random (10) +1 erzeugt Zufallszahlen von 1 bis 10 random(5) *6 erzeugt 0; 6; 12; 18; 24 usw.

Aufgabe:	Ich denke mir eine Zahl zwischen 1 und 100. Rate diese Zahl. Als Hilfe sage ich nur, ob die Zahl zu klein oder zu groß ist bzw. stimmt..
Formular:	

Quelltext:

Wenn die Steuerschaltfläche *Neue Zufallszahl* angeklickt wird, muss der Computer folgende Schritte ausführen:

- Bereitstellen einer Zufallszahl zwischen 1 und 100.
- Mitteilung für den Benutzer anzeigen.
- Prüfen-Button aktivieren, um das Raten zu ermöglichen.
- Schließen-Button sichtbar machen.

Und wir ergänzen noch einen Schritt. Wird der Button *Neue Zufallszahl* einmal angeklickt, hat der Button keine weitere Aufgabe, solange bis die Zahl geraten wurde. Aber wir brauchen einen Button, mit dem wir die Lösung gezeigt bekommen, wenn wir das Spiel vorzeitig aufgeben. Dazu benutzen wir den Button *Neue Zufallszahl*. Wir ändern die Caption-Eigenschaft und ergänzen eine Entscheidungslogik, um den Zustand dieser Steuerschaltfläche zu sehen. Wenn bei einem Klick die Aufschrift "Neue Zufallszahl" lautet, führen wir die obigen Schritte aus. Wenn bei einem Klick die Aufschrift "Lösung zeigen" lautet, wird die Lösungszahl angezeigt und alle Steuerelemente wieder auf ihre Ausgangswerte gesetzt.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  If Button1.caption= 'Neue Zufallszahl' then
    begin
      randomize;
      x:=random(100)+1;
      edit1.text:='Ich denke mir eine Zahl zwischen 1 und 100';
      edit2.text:='';
      button2.enabled:=true;
      bitbtn1.visible:=true;
      Button1.caption:='Lösung zeigen';
    end
  else
    begin
      edit1.text:='Die Lösung ist ' + inttostr(x);
      button2.enabled:=false;
      bitbtn1.visible:=false;
      Button1.caption:='Neue Zufallszahl';
    end;
end;
```

Wie man sieht, ist x hier nicht als Variable definiert. Warum ist das hier nicht nötig? Dazu sehen wir uns auch noch den Quelltext des Buttons *Zufallszahl* an:

```
procedure TForm1.Button2Click(Sender: TObject);
var tipp:integer;
begin
   tipp:=strtoint(edit2.text);
   If tipp = x Then
     begin
       edit1.text:='Stimmt!!';
       Button2.Enabled := False;
       Button1.Caption := 'Neue Zufallszahl';
       scrollbar1.position:=0;
       edit2.text:='';
     end;
  If tipp < x Then
    edit1.text:= 'Zu klein!';
```

```
  If tipp > x Then
    edit1.text:= 'Zu groß!';
end;
```

Auch hier taucht x auf. Auch hier wurde es nicht als Variable deklariert. Warum nicht? Ganz einfach: wenn ein und dieselbe Variable in mehr als einer Prozedur benutzt werden soll, muss man sie als **globale** Variable vereinbaren. Das geht im Implementationsteil:

```
implementation
{$R *.DFM}
var x:integer;
```

Den Umgang mit einer scrollbar kennen Sie bereits. Vervollständigen Sie das Projekt und speichern Sie es in einem Ordner mit dem Namen *Zahlenraten*.

Ergänzungen:

1. Ergänze eine Textbox, so dass der Benutzer den Zahlenbereich, in dem die Zahl gesucht werden soll, selbst festlegen kann.
2. Die Mitteilungen an den Benutzer können etwas "genauer" sein. Z.B. wie beim Spiel "Blinde Kuh" die Angaben eiskalt, kalt, warm, heiß. Dabei hilft die Funktion **Abs**, mit der man den Betrag einer Zahl bzw. den Abstand zweier Zahlen bestimmen kann.
3. Eine weitere Möglichkeit besteht darin, aus dem Projekt ein kleines Mathematik-Spiel zu machen, indem man dem Spieler mitteilt, wie weit seine geratene Zahl von der gedachten Zahl entfernt ist.
4. Noch eine Erweiterungsmöglichkeit: Die Anzahl der benötigten Versuche wird gezählt (Variable Versuche). Ist die gedachte Zahl gefunden worden, erhält der Spieler im Mitteilungsfeld eine Bewertung, z.B.

a) Versuche < 5	Mitteilung "Das war reiner Zufall!"
b) 4 < Versuche < 8	Mitteilung "Super!"
c) Versuche > 7	Mitteilung "Schwach!"

Für die Bereichsüberprüfung in b) schreibt man in Delphi den Ausdruck (Versuche > 4) **AND** (Versuche < 8)

Übung: Erstellen Sie einen Ordner mit dem Namen *Teiler*. Erstellen Sie dann folgendes Projekt:

Nach Eingabe der Zahlen a und b soll geprüft werden, ob b ein Teiler von a ist oder nicht. Eine entsprechende Ausgabe soll in einem Label erscheinen. Hinweis: Eine Zahl ist ein Teiler einer anderen Zahl, wenn bei der Division kein Rest bleibt, also ist z.B. 2 ein Teiler von 12, da 12 MOD 2 = 0 ist. Speichern Sie das Projekt im oben genannten Ordner!

Übung: Erstellen Sie einen Ordner mit dem Namen *Vergleich*. Kopieren Sie die Datei P:\TEXTE\KLASSE10\DELPHI\DOWNLOADS\VERGLEICH\VERGLEICH.EXE in Ihren Ordner und probieren Sie sie aus. Offensichtlich kann man hier drei natürliche Zahlen eingeben. Es wird angegeben, welche der Zahlen die kleinste ist und der Mittelwert wird berechnet.
Erstellen Sie dann das Projekt wie in der Vorgabe und speichern Sie alle Projektdateien im oben genannten Ordner!

Übung: Erstellen Sie einen Ordner mit dem Namen *Dreieck*. Kopieren Sie die Datei P:\TEXTE\KLASSE10\DELPHI\DOWNLOADS\DREIECK\DREIECK.EXE in Ihren Ordner und probieren Sie sie aus. Offensichtlich werden die drei Seiten eines Dreiecks eingegeben und es wird getestet, ob das Dreieck rechtwinklig ist oder nicht. Hinweis: Rechtwinkligkeit prüft man mit der Umkehrung des Satzes des Pythagoras. Wenn also $c^2 = a^2 + b^2$ **oder** $a^2 = b^2 + c^2$ **oder** $b^2 = a^2 + c^2$ gilt, so ist das Dreieck rechtwinklig, also `(c*c=a*a+b*b) OR (a*a=b*b+c*c) OR (b*b=a*a+c*c).`
Erstellen Sie dann das Projekt wie in der Vorgabe und speichern Sie alle Projektdateien im oben genannten Ordner! Wenn Sie nun das Programm testen, stellen Sie fest, dass ein Dreieck mit den Seitenlängen 3; 4 und 5 rechtwinklig ist, ein Dreieck mit den Seitenlängen 2; 3 und 4 dagegen nicht. Probieren Sie auch 10; 1 und 1 aus. Dieses Dreieck ist auch nicht rechtwinklig. Versuchen Sie das Dreieck zu zeichnen! Aha, es ergibt sich gar kein Dreieck. Natürlich, die Dreiecksungleichung ist ja auch nicht erfüllt. Beheben Sie das Problem, indem Sie das Programm um die Dreiecksungleichung erweitern!

Übung: Erstellen Sie einen Ordner mit dem Namen *Quaglei*. Kopieren Sie die Datei P:\TEXTE\KLASSE10\DELPHI\DOWNLOADS\QUAGLEI\QUAGLEI.EXE in Ihren Ordner und probieren Sie sie aus. Nutzen Sie für p und q folgende Paare, damit Sie alle Lösbarkeitsfälle ausprobieren: (4; 4) → eine Lösung; (5; 6) → zwei Lösungen und (1; 6) → keine Lösung.
Mit diesem Projekt wird eine quadratische Gleichung gelöst. Hinweis: Die Berechnungsformel für die Diskriminante ist dem Tafelwerk zu entnehmen. In Abhängigkeit von der Diskriminante kann eine quadratische Gleichung keine, eine oder zwei Lösungen besitzen.
Erstellen Sie dann das Projekt wie in der Vorgabe und speichern Sie alle Projektdateien im oben genannten Ordner!

3.4. Übungen zur Mehrfachauswahl

3.4.1. Vorbemerkungen

Sie können in Ihrer Anwendung auch die Anweisung case verwenden, um eine Verzweigung zu den entsprechenden Quelltext-Zeilen vorzunehmen. Die Anweisung case ist einer Folge von if-Anweisungen oftmals vorzuziehen, wenn die auszuwertende Variable oder Eigenschaft ein ordinaler Typ ist. Die Logik einer case-Anweisung ist normalerweise leichter zu verstehen als die von komplex geschachtelten if-Anweisungen, und darüber hinaus wird der Quelltext in case-Anweisungen schneller ausgeführt. Eine case-Anweisung besteht aus einem Ausdruck (Selektor) und einer Reihe von Anweisungen. Jeder Anweisung geht entweder eine oder mehrere Konstanten (sogenannte case-Konstanten) oder das reservierte Wort else voran. Der Selektor muss ordinal sein (also beispielsweise nicht vom Typ String). Alle case-Konstanten müssen ebenfalls von einem ordinalen Typ sein, der zum Typ des Selektors kompatibel ist.

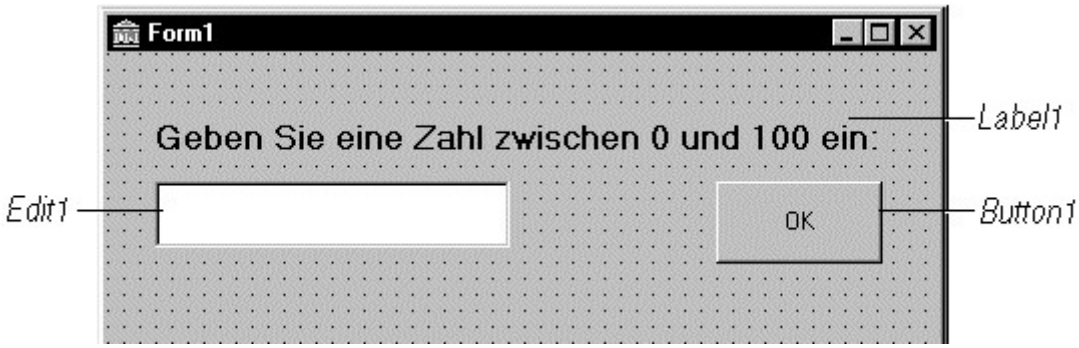
Hier ein Beispiel für case-Anweisungen:

```
case Operator of
  Plus: X := X + Y;
  Minus: X := X - Y;
  Mal: X := X * Y;
end;
```

case-Bereiche dürfen sich nicht überschneiden. Die folgende case-Anweisung ist beispielsweise nicht erlaubt:

```
case MySelector of
  5: Edit1.Text := 'Spezialfall';
  1..10: Edit1.Text := 'Allgemeiner Fall';
end;
```

Nachfolgend ein weiteres Beispiel für die case-Anweisung, basierend auf folgendem Formular, in dem neben den gekennzeichneten Felder noch ein derzeit nicht beschriftetes Label2 existiert.:



Diese Ereignisbehandlungsroutine ist mit dem OKClick-Ereignis der Schaltfläche verknüpft:

```
procedure TForm1.OKClick(Sender: TObject);
var
  Zahl: Integer;
begin
  Zahl := StrToInt(Edit1.Text);
  case Zahl of
    1, 3, 5, 7, 9: Label2.Caption := 'Ungerade Zahl';
    0, 2, 4, 6, 8: Label2.Caption := 'Gerade Zahl';
```

```
    10..100: Label2.Caption := 'Zwischen 10 und 100';
  else
    Label2.Caption := 'Größer als 100 oder negativ';
  end;
end;
```

Wenn Sie die Anwendung ausführen, eine Zahl eingeben und OK auswählen, wird die String-Entsprechung der Zahl in eine Ganzzahl umgewandelt und der Variablen Zahl zugewiesen. Die case-Anweisung verwendet dann den Wert von Zahl um festzustellen, welche Anweisung innerhalb der case-Anweisung ausgeführt wird. In dieser Quelltextzeile wird die Anweisung hinter dem Doppelpunkt ausgeführt, und Label2.Caption empfängt den String 'Ungerade Zahl', wenn der Wert von Zahl 1, 3, 5, 7 oder 9 ist:

1, 3, 5, 7, 9: Label2.Caption := 'Ungerade Zahl';

Wie die Anweisung **if** kann auch die Anweisung **case** optional einen **else**-Teil enthalten. (Achtung: Hier steht vor else ein Semikolon!!) Case-Anweisungen enden mit dem reservierten Wort end.

3.4.2. Einführungsbeispiel

Ein Versicherungsunternehmen bietet den Versicherten am Jahresende eine Beitragsrückerstattung an, die von der Dauer der Versicherung abhängt und folgender Tabelle zu entnehmen ist:

Versicherungsdauer in Jahren	0 bis 5	6 bis 8	9 bis 11	12 bis 14	15 und mehr
Erstattung in %	0	10	13	18	25

Aus Versicherungsdauer D und Beitrag B soll die Rückerstattung R berechnet werden. D ist vom Typ integer, B und R sind vom Typ real.
Mit If-Anweisungen könnte man so vorgehen:

if (D>=0) **and** (D<=5) **then** R := 0;
if (D>=6) **and** (D<=8) **then** R := 0.1 * B;
if (D>=9) **and** (D<=11) **then** R := 0.13 * B;
if (D>=12) **and** (D<=14) **then** R := 0.18 * B;
if (D>=15) **then** R := 0.25 * B;

Der Schreibaufwand ist aber sehr groß. Mit der case-Anweisung vereinfacht sich die Darstellung:

Case D **of**
0 .. 5 : R:=0;
6,7,8 : R:=0.1 * B;
9,10,11 : R:=0.13 * B;
12,13,14 : R: =0.18*B;
else R:=0.25*B;
End;

Übung:

1. Erstellen Sie einen Ordner mit dem Namen *Erstattung* und erstellen Sie dann das Projekt. Arbeiten Sie zunächst nur mit Labeln, Button und Editfeldern.
2. Wenn Sie fertig sind, erstellen Sie den Ordner *Erstattung2*. Kopieren Sie alle Projektdateien von *Erstattung* in *Erstattung2* und verändern Sie das Projekt, indem Sie eine scrollbar zur Eingabe der Versicherungsdauer D nutzen.

kurzer Überblick über CheckBoxen und RadioButton

Das Kontrollkästchen (CheckBox)

Mithilfe von Kontrollkästchen kann man eine Auswahl vornehmen (ankreuzen) oder verschiedene Zustandsgrößen anzeigen lassen. Kontrollkästchen werden üblicherweise in einer GroupBox angeordnet.

Das Optionsfeld (RadioButton)

Mit Optionsfeldern kann man aus einer Gruppe von Möglichkeiten genau eins auswählen. Optionsfelder werden immer in einer GroupBox gruppiert.

3.4.3. Hamburger

Pizza Taxi ist bekannt, aber warum nicht auch ein Bestellservice für Hamburger? Wir programmieren ein PC-Bestellsystem für Hamburger. Erstellen Sie dazu in Ihrem Arbeitsverzeichnis einen Ordner mit dem Namen *Hamburger* und speichern Sie alle Dateien dieses Projektes in dem Ordner ab!

Projektentwurf

Welche Aufgaben sind zu lösen?

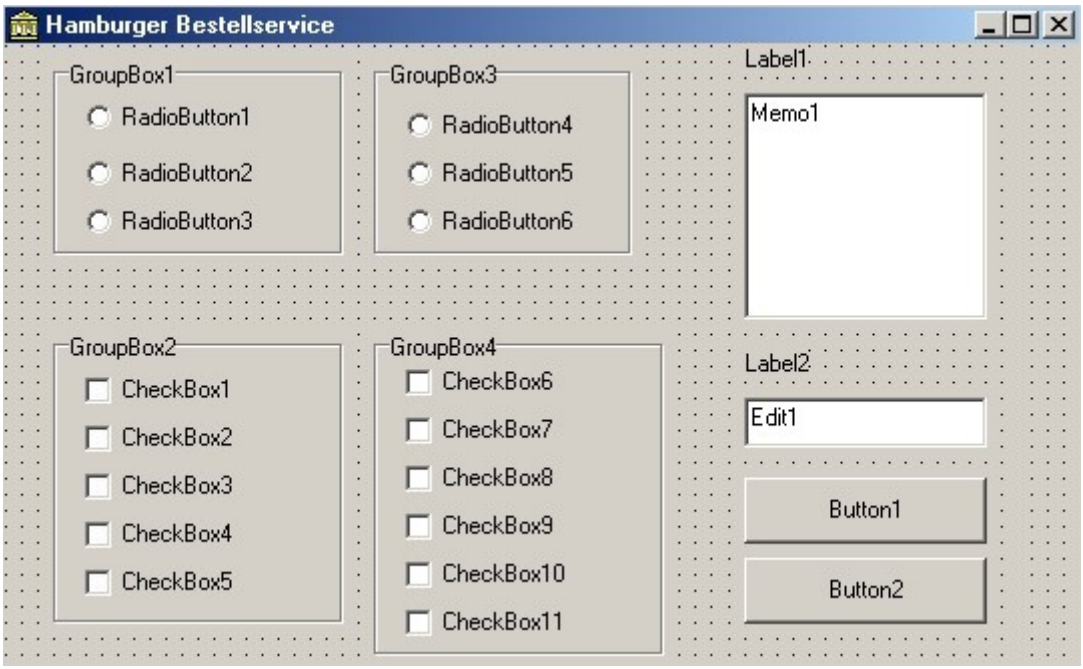
- Drei Brotsorten (eine möglich): Weissbrot, Weizen, Roggen
- Fünf Fleischsorten (mehrere möglich): Rind, Schinken, Puter, Spanferkel, Salami
- Drei Käsesorten (eine möglich): ohne, Gouda, Edamer
- Sechs Beilagen (mehrere möglich): Senf, Mayonnaise, Salat, Tomaten, Zwiebeln, Gurken

Steuerelemente

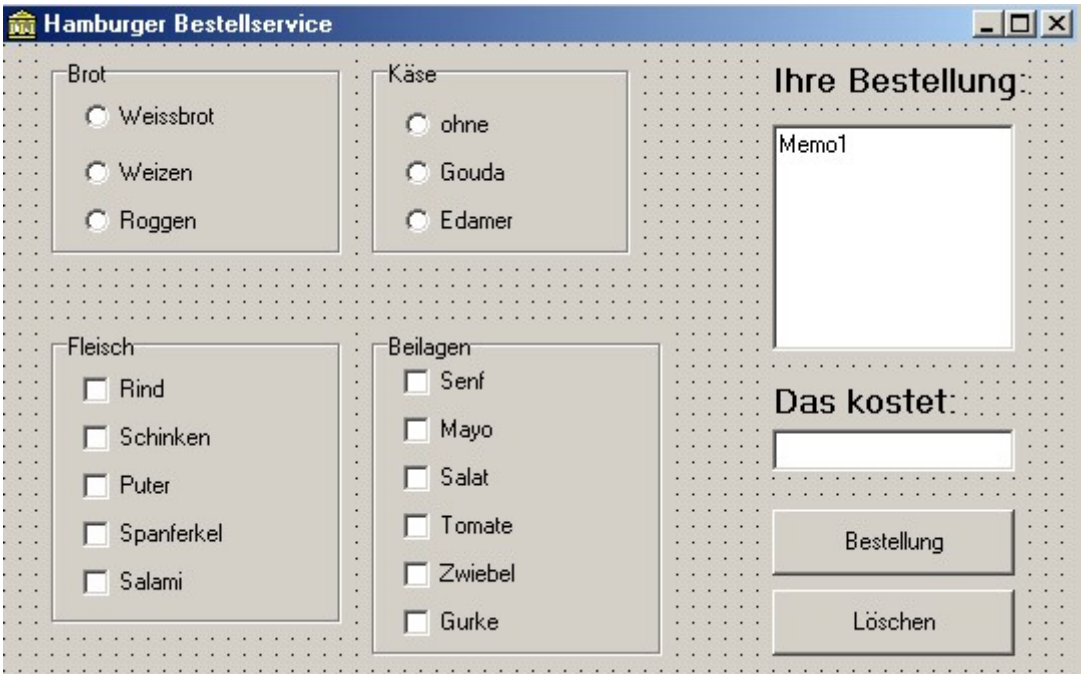
Wir brauchen eine Gruppe mit drei RadioButton für Brot, eine Gruppe mit fünf CheckBoxen für Fleisch, eine Gruppe mit drei RadioButton für Käse und eine Gruppe mit sechs CheckBoxen für Beilagen. Wir ergänzen noch einen Button zum Löschen der Menüauswahl, einen Button für die Zusammenstellung des Hamburgers sowie ein Memofeld (das gleiche wie ein Editfeld, nur über mehrere Zeilen), wo die Bestellung ausgegeben wird und ein Editfeld für den Endpreis.

Anordnung der Steuerelemente

Erstellen Sie einen Ordner mit dem Namen *Hamburger*! Erstellen Sie zunächst die Form! Speichern Sie!



Ändern Sie dann alle Captions so ab, wie es das nachfolgende Bild zeigt:



Programmierung

In diesem Projekt werden die CheckBoxen und RadioButton in den verschiedenen GroupBoxen dazu benutzt, die Zusammenstellung des Hamburgers festzulegen. Wenn die Auswahl erfolgt ist, wird **Bestellung** angeklickt und die gewünschte Zusammenstellung im Editfeld aufgelistet. Mit **Löschen** wird die Menütafel wieder auf ihre Ausgangswerte gesetzt.

Wir benötigen noch zwei Integer-Variablen: **BrotWahl** speichert die ausgewählte Brotsorte, **KaeseWahl** speichert die ausgewählte Käsesorte. Beide Variablen müssen global definiert werden, da sie in mehreren Prozeduren benutzt werden.

```
implementation
  var Brotwahl,Kaesewahl:integer;
```

Außerdem legen wir folgende Zuordnung fest:

Brotwahl		Kaesewahl	
Wert	Brotsorte	Wert	Käsesorte
1	Weiss	0	kein
2	Weizen	1	Gouda
3	Roggen	2	Edamer

Die Verwendung von Variablen zum Speichern der Wahl bei Optionsfeldern ist gängig. Wir ergänzen noch zwei Variablen (**AnzahlFleisch** und **AnzahlBeilagen**), um zu zählen, wie viele Sorten jeweils ausgewählt wurden. Da diese Variablen nur in der Prozedur für die Bestellung benutzt werden, reicht es, die Variablen innerhalb der Bestellungsprozedur zu vereinbaren.

BrotWahl und KäseWahl müssen initialisiert werden, und zwar passend zu den Ausgangswerten der Optionsfelder: **RadioButton1** (Weissbrot) hat den Wert True und **RadioButton4** (kein Käse) hat den Wert True. Damit ergibt sich mit obiger Zuordnungstabelle für die Initialisierung mittels der **TForm1.FormCreate** Prozedur

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  radiobutton1.checked:=true;
  radiobutton4.checked:=true;
end;
```

Nachdem sämtliche Auswahlen an der Menütafel erfolgt sind, wird **Bestellung** angeklickt. An dieser Stelle muss der Computer nun Folgendes tun:

- Entscheiden, welche Brotsorte gewählt wurde.
- Entscheiden, ob und welche Fleischsorten gewählt wurden.
- Entscheiden, ob und welche Käsesorte gewählt wurde.
- Entscheiden, ob und welche Beilagen gewählt wurden.
- Bestellung in das Memofeld und Preis in das Textfeld schreiben.

Hier einige Beispielprozeduren:

```
procedure TForm1.RadioButton1Click(Sender: TObject);
begin
  Brotwahl:=1;
end;
```

die anderen RadioButton entsprechend,

```
procedure TForm1.Button1Click(Sender: TObject);
var
  anzahlfleisch, anzahlbeilagen:integer;
  preis:real;
begin
  case brotwahl of
    1:memo1.text:=memo1.text+'Weissbrot'+' 0.40 € ';
    2:memo1.text:=memo1.text+'Weizen'+' 0.40 € ';
```

```
    3:memo1.text:=memo1.text+'Roggen'+' 0.40 € ';
  end;
    preis:=0.4;

  case Kaesewahl of
    0:memo1.text:=memo1.text+'ohne Käse';
    1:begin
        memo1.text:=memo1.text+'Gouda'+' 0.40 € ';
        preis:=preis+0.4;
      end;
    2:begin
        memo1.text:=memo1.text+'Edamer'+' 0.40 € ';
        preis:=preis+0.4;
      end;
  end;

  anzahlfleisch:=0;
  if checkbox1.Checked then begin
    anzahlfleisch:=anzahlfleisch+1;
    preis:=preis +1;
    memo1.text:=memo1.text+'Rind'+ ' 1.00 € ';
  end;
```

die anderen CheckBoxen entsprechend und für den Gesamtpreis:

```
edit1.text:=edit1.text+floattostrF(preis,ffFixed,6,2)+' €'
```

Ergänzen Sie die fehlenden Klickprozeduren! Wie kann man einen Zeilenumbruch im der Komponente Memo1 erreichen?
Außerdem soll bei Klick auf den Button *Löschen* der Anfangszustand wieder hergestellt werden. Ergänzen Sie die Prozedur! Testen Sie und speichern Sie!
Informieren Sie sich auch über die Komponente *memo1*, die Eigenschaft *memo1.lines* und die Methoden *memo1.lines.add(..)*, *memo1.lines.clear* und *memo1.lines.count.*

Übung: Erstellen Sie einen Ordner mit dem Namen *Erstattung3*. Wandeln Sie die Projekte *Erstattung* bzw. *Erstattung2* so ab, dass Sie nun radiobutton zur Auswahl der Versicherungsdauer D benutzen! Im Ordner *Downloads* finden Sie eine entsprechende startbare Datei.

Übung: Erstellen Sie einen Ordner mit dem Namen *Rechner*. Ein Großhändler räumt beim Kauf von Taschenrechnern Rabatte ein, die folgender Tabelle zu entnehmen sind.

Kauf von mindestens	10	20	50	100
Erstattung in %	10	15	20	25

Der Computer soll nach Eingabe der Stückzahl (radiobutton) und dem Einzelpreis (editfeld) den Rabatt, die Mehrwertsteuer (16% von Einzelpreis abzüglich Rabatt) und den Endpreis ausgeben. Im Ordner Downloads finden Sie eine entsprechende startbare Datei.

---

zum Weiterüben:

1. Eine Versandhandlung tätigt ihre Geschäfte nach folgenden Grundsätzen:
   a. bei einer Auftragshöhe von unter 100€ werden 5€ als Portokostenzuschuss aufgeschlagen.

b. ab 100€ wird ein Rabatt von 2% gewährt.

Gesucht ist ein Programm, das zu gegebener Auftragshöhe den Rechnungsbetrag ausgibt.

2. Zwei natürliche Zahlen haben die gleiche Parität, wenn sie beide gerade oder ungerade sind. Gesucht ist ein Programm, das zu zwei eingegebenen Zahlen feststellt, ob sie gleiche Parität besitzen oder nicht.
   (*Hinweis: Die Funktion* ***odd(x)*** *stellt fest, ob eine natürliche Zahl ungerade ist.)*

3. Gesucht ist ein Programm, das von zwei eingegebenen Namen den alphabetisch ersten wieder ausgibt.
   (*Hinweis: Zeichenketten, Datentyp* `string` *, lassen sich wie Zahlen mittels der Relationen =, <, >, <=, >=, <> vergleichen.*)

4. Eine Versicherung erstattet ihren Mitgliedern einen Teil der Jahresprämie, und zwar bei mindestens fünfjähriger Mitgliedschaft 8%, andernfalls 4%. Ein Programm ist gesucht, das nach Eingabe der Mitgliedsdauer und Jahresprämie den Erstattungsbetrag ausgibt.

5. Die Mitarbeiter einer Firma sollen eine Gehaltserhöhung bekommen. Nach zähen Verhandlungen einigt man sich darauf, die Gehälter um 3,5%, mindestens jedoch um 100€ monatlich zu erhöhen. Gesucht ist ein Programm, das nach Eingabe des alten Gehalts das zugehörige neue ausgibt.

6. Das sogenannte Idealgewicht berechnet sich bei männlichen Personen nach der Formel "Körpergröße (in cm) - 100, davon 95%"; bei weiblichen Personen sind entsprechend 90% zu nehmen. Man schreibe ein Programm, das nach Eingabe von Gewicht, Körpergröße und Geschlecht angibt, ob Über-, Unter- oder Idealgewicht vorliegt. Dabei sind Fehler von ±2% zu berücksichtigen.

7. Gesucht ist ein Programm, welches die Lösung der Gleichung $ax^2+bx+c=0$ unter Berücksichtigung aller Fälle für die Koeffizienten a, b und c ermittelt.

8. Ein Lehrer zensiert nach folgendem Verfahren: Zunächst vergibt er Punkte und setzt eine maximal erreichbare Punktzahl fest. Dann ermittelt er den Prozentwert der tatsächlich erreichten Punkte; die Umrechnung in Noten erfolgt so: weniger als 20% ergibt Ungenügend, von 20 bis unter 40 % ergibt Mangelhaft, 40 bis unter 55% ergibt Ausreichend, 55 bis unter 70% ergibt Befriedigend, 70 bis unter 85% ergibt Gut, 85 bis 100% ergibt Sehr Gut. Gesucht ist ein Programm, das bei der Umrechnung hilft.

9. Beim Bausparen erhalten Verheiratete für Sparleistungen bis zu 800€ eine Prämie, Alleinstehende nur für Sparleistungen bis zur Hälfte des Betrages. Die Prämie beträgt 14% der Sparleistung. Für jedes Kind werden 2% zusätzlich gewährt. Man schreibe ein Programm, welches die Bausparprämie ermittelt.

10. Eine Firma gewährt Großhändlern 15%, Einzelhändlern 10% und normalen Kunden 5% Rabatt. Schreibe ein Programm, das je nach Kundentyp einen Rechnungsbetrag ausgibt.

3.5. Zyklische Strukturen

3.5.1. Vorbemerkungen

Dem Pascal-Programmierer mag es nicht schwer fallen, die Arbeit mit zyklischen Strukturen zu motivieren: man schreibe ein überschaubares lineares Programm und stelle dann die Frage, wie dem armen Nutzer zu helfen sei, der mit diesem Programm Hunderte von Berechnungen durchführen soll.
Die Lösung liegt schnell auf der Hand: man packe das ganze in eine Repeat-Schleife an deren Ende eine Nutzerauswahl "Neue Berechnung?" / "Ende?" erfolgt und entbinde damit den potentiellen Nutzer von der lästigen Aufgabe, für jede neue Berechnung auch einen neuen Programmstart veranlassen zu müssen.

Unter Delphi greift diese Motivierung natürlich nicht, denn bis zum ultimativen "Form1.Close", verbunden mit einem "Ende"-Button, kann ein Nutzer die Funktionalität des Programm-Fensters so oft strapazieren, wie er möchte, da dank der Ereignisorientierung ohnehin ein zyklisches Abfragen eingabesensitiver Komponenten erfolgt.
Bewährt hat sich der **Einstieg über einen einfachen zyklischen Algorithmus**, etwa der Iteration einer Quadratwurzel nach Heron. Hieraus werden für das Verständnis von Schleifen notwendige Erkenntnisse, wie z. B. die Bedeutung der Abbruchbedingung abgeleitet und dann auf weitere Beispiele angewandt.

Nichtabweisender Zyklus / Abweisender Zyklus / Zählzyklus ?
Immer häufiger hört man die Frage *"Müssen die Schüler denn wirklich alle drei Schleifenarten beherrschen?"* Mein Standpunkt: wenn die Zeit dazu vorhanden ist, so hat 's noch keinem geschadet. Jedoch fehlt es meist an eben dieser Zeit. Nach dem Motto **"Weniger ist meistens mehr"** würde ich es für sinnvoller halten, wenn der Schüler im Extremfall mit nur einer einzigen Schleifenart eine Vielzahl von Problemen aufbereiten und lösen kann, als dass er zwar alle Schleifenarten formal kennt, jedoch kaum in der Lage ist, diese selbständig zur Problemlösung zu verwenden.

Die Kunst des Weglassens sollte unbedingt die Überlegung einschließen, dass "Repeat-" und "While-Schleifen" (also Nichtabweisende und Abweisende Zyklen) gegenüber dem Zählzyklus die universelleren Werkzeuge darstellen, d. h. alle auf iterativem Wege lösbaren Probleme lassen sich damit umsetzen, während mit der Zählschleife nur eine Teilmenge dieser Lösungen (sauber) programmiert werden kann.
In meinen Kursen empfand ich es bisher als gangbaren Kompromiss, das **Wesen zyklischer Algorithmen** zunächst anhand der Repeat-Schleife zu verdeutlichen und nachfolgend "so zu tun", als gäbe es nur diese eine Schleifenart. Mit dieser Strukturkenntnis werden alsdann bei zunehmender Selbständigkeit einfache Problemstellungen aus verschiedenen Gebieten bearbeitet und erst am Ende werden eher überblicksartig die beiden verbleibenden Schleifenarten vorgestellt und Vergleiche dazu angestellt

Der Iterationsbegriff:

"Aus dem lat. iteratio = Wiederholung. Verfahren der numerischen Mathematik, das dazu dient, durch wiederholtes Durchlaufen eines bestimmten Rechenverfahrens die Berechnung eines Wertes mit immer größerer Genauigkeit zu erreichen...
Im übertragenen Sinne verwendet man den Ausdruck I. auch für die Wiederholung eines Schleifendurchlaufes in einem Programm.
Wesentlich für jede Iteration ist, dass das Ergebnis des ersten Schrittes als Eingabewert für den folgenden Schritt verwendet wird.
Die Iteration wird im Allgemeinen durch die Zahl der Durchläufe beendet oder durch eine Bedingung, deren Erreichen ebenfalls zum Abbruch der Iteration führt [Abbruchbedingung] ..."

Computer-Enzyklopädie S. 1563

3.5.2. Einführung des Nichtabweisenden Zyklus - Quadratwurzel nach Heron -

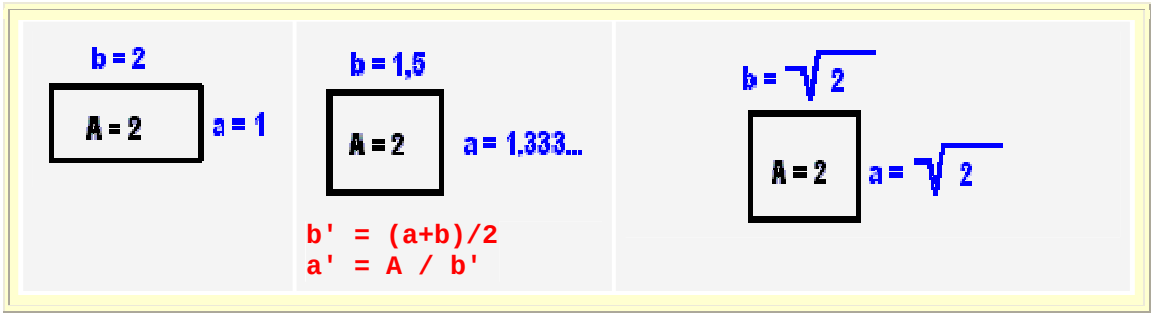
Szenario:

Interessant ist es Schüler der heutigen Generation mit der Frage zu konfrontieren, wie denn wohl unsere Vorfahren ohne Hilfsmittel die Quadratwurzel einer natürlichen Zahl "gezogen" haben. Der Gedanke, dass dies durch Näherungsverfahren geschehen sein muss, ist schnell geäußert; die Frage aber, wie man denn mit vertretbarem Aufwand eine möglichst hohe Genauigkeit erreichen konnte, bleibt meist unbeantwortet.
Ein knapp 2000 Jahre altes Verfahren, entwickelt durch ***HERON von Alexandria*** (ca. 20-62 n. Chr.), löst das Problem auf geniale und aus heutiger Sicht sehr computerfreundliche Weise!

Der Heronsche Algorithmus - an einem Zahlenbeispiel vorgestellt:

Wir stellen uns vor, die Quadratwurzel der Zahl 2 sei gesucht. Von der zu findenden Lösung wissen wir bereits, dass deren Quadrat wiederum 2 ergeben muss.
Als Ausgangspunkt wählen wir ein Rechteck, dessen Seitenlänge a = 1 ist; Seitenlänge b ist so groß wie die zu radizierende Zahl, also 2. Der sich ergebende Flächeninhalt gleicht somit dem Quadrat der gesuchten Zahl.
Über das arithmetische Mittel aus a und b wird jetzt b neu bestimmt, also b'=(a+b) / 2 = 1,5.
Unter Beibehaltung des Flächeninhaltes A=2 ermitteln wir jetzt den neuen Wert für a, also a' = A / b' = 2 / 1,5 = 1,333...
Das Verfahren wird so oft wiederholt, bis eine gewünschte Genauigkeit erreicht ist, d. h. die Differenz aus b und a ausreichend gering ist.
Nach unendlich vielen Iterationsschritten wäre a = b = Quadratwurzel der Zahl 2.

Ausgangspunkt:	***1. Iterationsschritt:***	***... nach unendlich vielen Iterationsschritten:***



Umsetzung des Heronschen Algorithmus mit einer Nichtabweisenden Schleife:

Struktogramm:	***Quelltext:***

Deklariere zahl: ganzzahlig, a, b, fehler: reell.
Eingabe von zahl und fehler
a := 1 b := zahl
Wdh. b := (a+b) / 2; a := zahl / b
bis b - a < fehler
Ausgabe von b
	`...`
	`a:=1; b:=zahl;`
	`repeat`
	`  b:=(a+b)/2;`
	`  a:=zahl/b;`
	`until b-a < fehler;`
	`...`

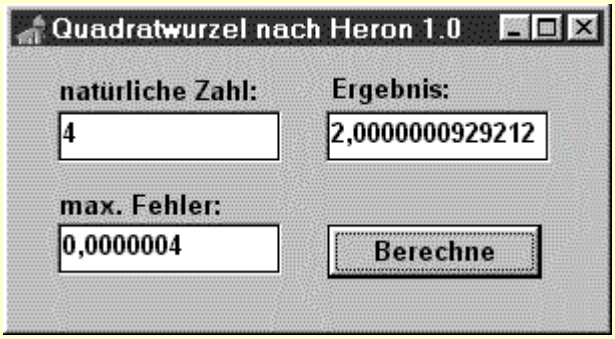
Zur Beachtung:

Die Variable ***fehler*** legt in der Abbruchbedingung die erlaubte Abweichung zwischen ***b*** und ***a*** fest. Wird ***fehler*** z. B. im Rahmen der Eingabe mit 0.0004 belegt, so bewirkt dies eine Genauigkeit des Ergebnisses auf 3 Nachkommastellen.
Der Rechner prüft nach jedem Schleifendurchlauf, ob die sog. **Abbruchbedingung** erfüllt ist (hier: b-a < fehler). Ist diese erfüllt, wird die Schleife verlassen und mit derjenigen Anweisung fortgesetzt, die der Abbruchbedingung folgt. Bei Nichterfüllung der Abbruchbedingung wird der sog. Schleifenkörper erneut durchlaufen.
Wichtig ist es daher, dass über die der Schleife vorangehenden Anweisungen in Verbindung mit dem Schleifenkörper selbst eine Struktur vorliegt, die nach endlich vielen Durchläufen die Abbruchbedingung auch wirklich erreicht.
Andernfalls hätte man eine **"Endlosschleife"** programmiert, die im schlimmsten Fall einen kompletten Rechnerabsturz zur Folge haben könnte.
Spätestens dann stellt man sich die bange Frage: *"Wann habe ich mein Programm eigentlich das letzte Mal gesichert?"* ;-)

Programmierung der Quadratwurzel nach Heron

a) Grundvariante

Oberflächengestaltung:	*Quelltext:*

	`procedure TForm1.Button1Click(Sender: TObject);`
	`  var zahl: integer;`
	`  a,b,fehler:real;`
	`  begin`
	`  zahl := strtoint(edit1.text);`
	`  fehler := strtofloat(edit2.text);`
	`  a:=1; b:=zahl;`
	`  repeat`
	`  b:=(a+b)/2;`
	`  a:=zahl/b`
	`  until b-a < fehler;`
	`  edit3.text := floattostr(b);`
	`  end;`

b) Erweiterte Variante

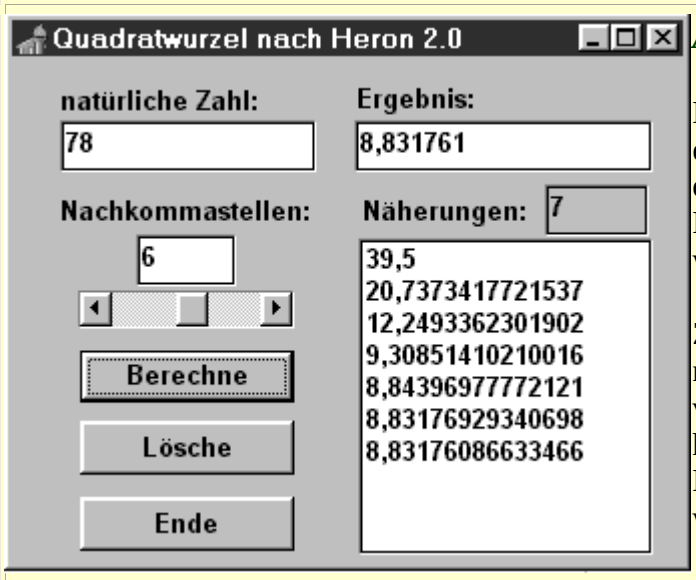
Bezüglich Benutzerfreundlichkeit und Erkenntnisgewinn lässt die Grundvariante des Programms noch einige Wünsche offen:

1. der Heronsche Algorithmus ist nur für natürliche Zahlen **`x>=1`** definiert, das Eingabe-Textfeld erlaubt jedoch auch Einträge von Werten **`x<=0`** (Absturzgefahr / falsches Ergebnis);
   Lösungshinweis: Eingabesicherung

2. im mathematischen Sinne ganzzahlige Ergebnisse sind aufgrund ihrer iterativen Erzeugung mit "fehlerhaften" Nachkommastellen belastet;
   Lösungshinweis: Formatierung mittels FloatToStrF

3. es ist nicht abzusehen, wie viele Schleifendurchläufe zum Erreichen der geforderten Genauigkeit notwendig waren;
   Lösungshinweis: Zählfunktion in Schleifen

4. dem Nutzer müsste erklärt werden, was mit max. Fehler gemeint ist und welche Werte hier als Eingabe sinnvoll sind.
   Lösungshinweis: Eingabefunktion mittels TScrollBar



Aufgabe:

Das Programm ist so zu erweitern, dass möglichst viele der oben aufgeführten Kritikpunkte überwunden werden.

Zur Orientierung kann die nebenstehende Abbildung verwendet bzw. das Programm heron2.exe aus dem Ordner DOWNLOADS downgeladen werden.

Bemerkenswert ist die geringe Anzahl an Iterationen, die zum Erreichen hoher Genauigkeiten erforderlich ist. Immerhin ist nach 7 Schritten die Wurzel der Zahl 78 auf 6 Nachkommastellen genau bestimmt!

Anmerkungen:

Um alle Näherungswerte (im Bsp. der Verlauf von b) anzeigen zu können, empfiehlt sich die Verwendung einer **ListBox-Komponente**. *(Hinweis: Ein Projekt zum Umgang mit Listboxen finden Sie im Anschluss an das Heronverfahren.)* Natürlich muss die "Bestückung" der ListBox *(ListBox1.Items.Add(FloatToStr(b));)* innerhalb der Schleife erfolgen.

Lösungshinweise:

Eingabesicherung

Das Prinzip ist schnell erklärt: der Nutzer ist so oft zur Wiederholung seiner Eingabe aufzufordern, bis sein Eingabewert im erlaubten Bereich liegt!

Beispiel: Es dürfen über Edit1 nur ganzzahlige Werte 10 <= x <= 20 eingelesen werden.

Über ein Dialogfenster soll der Nutzer über seinen Fehler informiert werden; anschließend ist das entsprechende Edit-Feld zu löschen und der Cursor zwecks Neueingabe wieder in diesem zu positionieren. In der zugehörigen Ereignisbehandlungsroutine, meist `Procedure Button*Click(...);` wären an den Anfang folgende Anweisungen einzuarbeiten:

```
x := StrToInt (Edit1.Text);
if (x < 10) OR (x>20) then begin
  ShowMessage ('Wert muss zwischen 10 und 20 liegen!');
  Edit1.Clear;
  ActiveControl := Edit1;
end
```

```
else begin
  {Jetzt folgen die Anweisungen zur Verarbeitung von x}
  {..........}
end;
```

Da die Verarbeitung des x-Wertes im Else-Zweig erfolgt, kommt diese erst nach korrekter Eingabe zustande.

Formatierung mittels FloatToStrF

Während der Programmierer bei der Verwendung der Funktion "FloatToStr" keinen Einfluss auf das Erscheinungsbild der String-Ausgabe des Real-Wertes hat, eröffnen sich durch "**FloatToStrF**" vielfältige Formatierungsmöglichkeiten. So lässt sich Beispielsweise die Anzahl der angezeigten Nachkommastellen in Abhängigkeit der iterierten Genauigkeit einstellen. Näheres zur Verwendung von "FloatToStrF" ist der Delphi-Hilfe zu entnehmen.

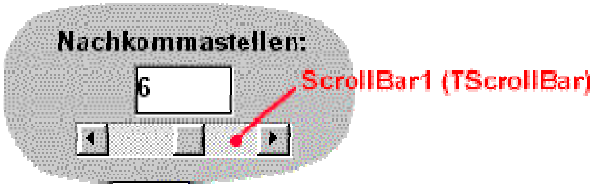
Zählfunktion in Schleifen

So wie Kinder mit den Fingern zählen, kann man's auch den Schleifen "beibringen". Man führe einfach eine ganzzahlige Zählvariable ein, setze deren Wert vor der Schleife auf Null und erhöhe diesen bei jedem Schleifendurchlauf um den Wert 1. Am Ende gibt die Zählvariable Auskunft darüber, wie oft die Schleife durchlaufen wurde.

Beispiel:

```
zaehler := 0;
Repeat
  {...}
  zaehler := zaehler + 1;
Until ... ;
```

Eingabefunktion mittels TScrollBar



Zur Einstellung der Nachkommastellen und damit des zulässigen Fehlers wurde hier eine Komponente **TScrollBar** verwendet, deren aktueller Wert in dem darüber liegenden Textfeld (TEdit) zur Anzeige gelangt.
Um die bei Realtypen verwalteten Nachkommastellen nicht überzustrapazieren, wären über den Objektinspektor für ScrollBar1 die Eigenschaften *Min=1* und *Max=10* zu setzen. Die Eigenschaft *ScrollBar1.Position* gibt den vom Nutzer eingestellten ganzzahligen Wert (zwischen Min und Max) zurück.
Zu beachten ist auch der Zusammenhang zwischen eingestellten Nachkommastellen und dem sich daraus ergebenden zulässigen Fehler. In der Abb. bedeuten z.B. 6 Nachkommastellen einen Fehler von 0.0000004.

c) Zusatzaufgabe

Heron wird zugeordnet, auch einen Algorithmus zur näherungsweisen Berechnung von Kubikwurzeln entwickelt zu haben. Dies ist sehr nahe liegend, denn was mit Rechtecken und Quadraten funktioniert müsste auch auf Quader und Würfel anwendbar sein ...

1. Transformieren Sie mit einem einfachen Zahlenbeispiel die oben gezeigten Lösungsskizzen auf das Problem der Kubikwurzel und entwickeln Sie die entsprechenden Gleichungen bzw. Wertzuweisungen für die Iteration der Ergebnisses!

2. Verallgemeinern Sie die Lösungsidee zu einem Algorithmus in Struktogrammform.

3. Erstellen Sie ein nutzerfreundliches Delphi-Programm zur iterativen Berechnung der 3. Wurzel einer natürlichen Zahl.

3.5.3. Arbeit mit TListBox-Komponenten

Zielstellung und Szenario:

TListBox-Komponenten stellen für einfache Programme ein geeignetes Mittel dar, um die Ausgabe größerer Datenmengen zu visualisieren. Insbesondere bei den nachfolgenden Übungen, die sich mit Schleifen beschäftigen, sollte der sichere Umgang mit Listboxen zum Repertoire der Schüler gehören.
Darüber hinaus verfolgt die vorliegende Übung das Ziel, die effektive Nutzung der interaktiven Hilfe-Funktionen von Delphi weiter zu trainieren.

Aufgabenstellung:

Die Komponente TListBox ist ein Listenfeld in Windows. In einem Listenfeld wird eine Liste von Strings (Zeichenketten) angezeigt, aus der ein oder mehrere Listenelemente ausgewählt werden können

---

1. Erstellen Sie unter Delphi ein Formular gemäß der nachfolgenden Vorgabe!

2. Informieren Sie Sich in der Delphi-Hilfe über die wichtigsten Eigenschaften und Methoden der Komponente TListBox und realisieren Sie danach folgende OnClick-Ereignisbehandlungen:

a) **Button1 (Hinzu)**
Der Text von Edit1 soll an die Liste in ListBox1 angefügt werden. Anschließend ist der Inhalt von Edit1 zu löschen und die Eigenschaft Form1.ActiveControl auf Edit1 zu setzen.
(Eigenschaft Items, Methode Add)

b) **Button2 (Hinweg)**
Der gerade markierte Listeneintrag von ListBox1 (z.B. Hans) soll aus der Liste entfernt werden.
(Eigenschaft Items, Methode Delete sowie Eigenschaft Itemindex)

c) **Button3 (Sortiere)**
Die ListBox soll in sortierter Form erscheinen und alle folgenden Einträge sollen in die Sortierung eingefügt werden.
(Eigenschaft Sorted)

d) **Button4 (Lösche alles)**
Die gesamte ListBox soll gelöscht und die Sortierung aufgehoben werden.
(Methode Clear, Eigenschaft Sorted)

3. Speichern Sie das Projekt unter "Listbox1.dpr" und drucken Sie Sich die Prozeduren zur Ereignisbehandlung aus!

Zusatzaufgabe:

Fügen Sie im fertiggestellten Programm die Zahlen von 1 bis 20 in umgekehrter Reihenfolge in das Listenfeld ein und betätigen Sie anschließend den "Sortiere-Button"!
Achten Sie auf die sich ergebende Sortier-Reihenfolge und begründen Sie deren Zustandekommen!
Finden Sie Möglichkeiten, die Zahlen trotzdem chronologisch zu sortieren?!

Auszug aus der Delphi-Hilfe

Komponente TListBox

Unit StdCtrls

Beschreibung

Die Komponente TListBox ist ein Listenfeld in Windows. In einem Listenfeld wird eine Liste angezeigt, aus der ein oder mehrere Listenelemente ausgewählt werden können.
Diese Liste ist der Wert der Eigenschaft **Items**. Die Eigenschaft **ItemIndex** zeigt an, welches Listenelement gerade ausgewählt wurde.
Mit den Methoden **Add**, **Delete** und **Insert** des Objekts Items, das vom Typ TStrings ist, lassen sich Listenelemente anfügen, löschen und einfügen. So würde man zum Beispiel einen String in einem Listenfeld mit folgender Programmzeile anfügen:

```
ListBox1.Items.Add('Neues Element');
```

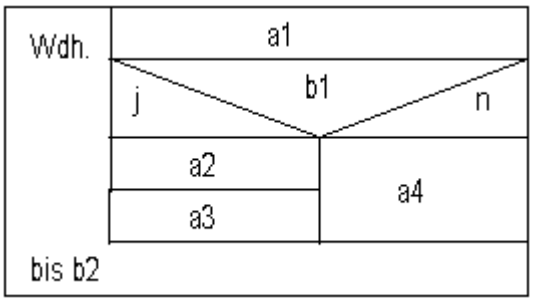
Auch das Erscheinungsbild des Listenfelds ist änderbar. So kann man ein mehrspaltiges Listenfeld durch Änderung des Wertes der Eigenschaft Columns erzeugen. Die Eigenschaft **Sorted** ermöglicht eine Sortierung der Listenelemente.
...

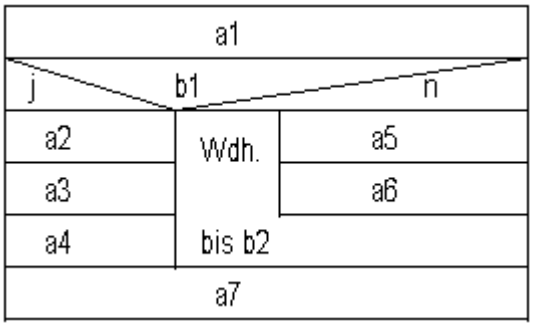
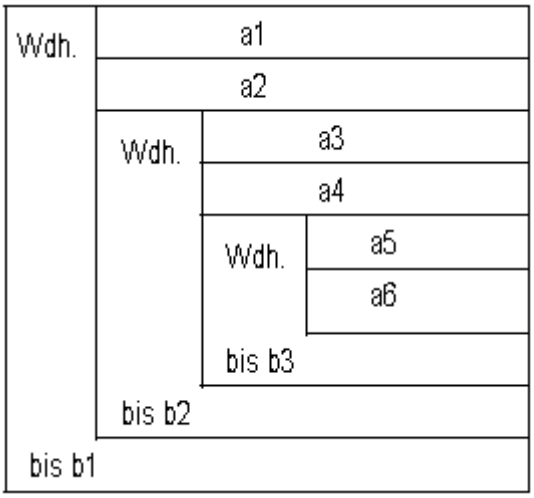
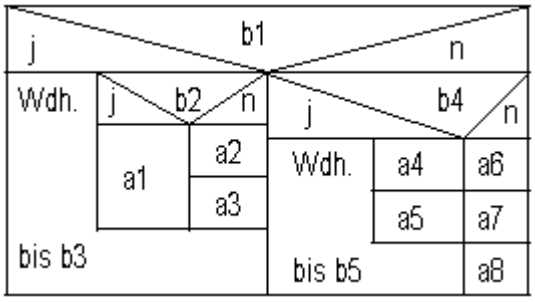
3.5.4. Übungen

3.5.4.1. Übertragung Struktogramm in Quelltext

Als algorithmische Kontrollstrukturen kennen wir von den vorangegangenen Seiten ***Sequenzen, Alternativen*** und ***nichtabweisende Zyklen***. Diese lassen sich in Problemlösungsalgorithmen in nahezu beliebiger Weise und Tiefe miteinander kombinieren bzw. ineinander schachteln. Die nachfolgenden Struktogramme sind gemäß Beispiel a) 1:1 in pascalgerechte Notationen zu übersetzen, wobei *"a"* für eine allgemeine Anweisung und *"b"* für eine allgemeine Bedingung steht.

	Struktogramm	Pascal-Quelltext

a)		`repeat`
		`  a1;`
		`  if b1 then`
		`  begin`
		`  a2; a3`
		`  end`
		`  else`
		`  a4`
		`until b2;`

b)		**?**
c)		**?**
d)		**?**

3.5.4.2. "Schreibtischtest" von zyklischen Strukturen

Zweck: 1. Erkennen der Variablenbelegungen für verschiedene Eingabewerte,
2. Nachweis, ob eine Abbruchbedingung bei bestimmten Eingabewerten erreicht wird oder nicht.

3.5.4.2.1. Beispielaufgabe und Lösung

a) Zu testen ist folgender nichtabweisender Zyklus für die Eingabe edit1.text := "4" !

```
x := StrToInt(edit1.text);
n := 0; y := 1;
REPEAT
  n := n+1;
  y := y*2
UNTIL n = x;
edit2.text := IntToStr(y);
```

b) Welche mathematische Funktion wird von der Struktur realisiert ?

c) Was würde geschehen, wenn man edit1.text mit "-1" belegt ? - Schlussfolgerung ?

Lösung + Hinweise:

zu a) Der Schreibtischtest erfolgt am besten in Tabellenform. Hier werden alle im Algorithmus relevanten Variablen eingetragen und hinsichtlich der Änderung ihrer Werte beim (gedanklichen) Ablauf des Programmes untersucht. Sind Schleifen im Spiel, so muss nach jedem fiktiven Durchlauf das Erreichen der Abbruchbedingung überprüft werden.

Variablen:		x	n	y
Vorgabe:		*4*	*0*	*1*
Durchläufe:	1.	*4*	*1*	*2*
	2.	*4*	*2*	*4*
	3.	*4*	*3*	*8*
	4.	*4*	*4*	*16*

zu b) Hierbei ist die Abhängigkeit der Ausgabevariable(n) von der/den Eingabevariable(n) zu untersuchen, im Beispiel also y = f(x). Erkennt man die Abhängigkeit nicht sofort, führe man weitere "Schreibtischtests" mit geänderten Eingabevariablen durch!
Im Beispiel beträgt $y = 2^x$

zu c) Da n bereits im ersten Schleifendurchlauf mit dem Wert 1 belegt ist und in jedem weiteren Durchlauf um 1 erhöht wird, entstünde bei Eingabe von -1 für x eine sog. "Endlosschleife", weil die Abbruchbedingung **n=x** theoretisch niemals erreicht wird.
In der Praxis endet das Ganze entweder mit einer Fehlermeldung (Bereichsüberschreitung / Stacküberlauf etc.) oder mit einem tobsüchtigen Nutzer, der im schlimmsten Fall irgendwann die Reset-Taste betätigt ;-)

Fazit:

1. Insbesondere zyklische Strukturen sollten vor dem Programmstart

gedanklich daraufhin getestet werden, welche Eingabewerte zu Endlosschleifen o. a. Fehlern führen könnten.
2. Mit Hilfe einer zu programmierenden Eingabesicherung "zwinge" man den Nutzer dazu, nur "erlaubte" Werte einzugeben.
3. Da 1. und 2. nie 100% Sicherheit bieten, speichere man jedes in Entwicklung befindliche Programm vor dem Compilieren bzw. Starten.

3.5.4.2.2. Aufgabe zum Schreibtischtest

a) Das nachstehende Programmstück ist per "Schreibtischtest" für den Eingabewert "5" zu untersuchen !

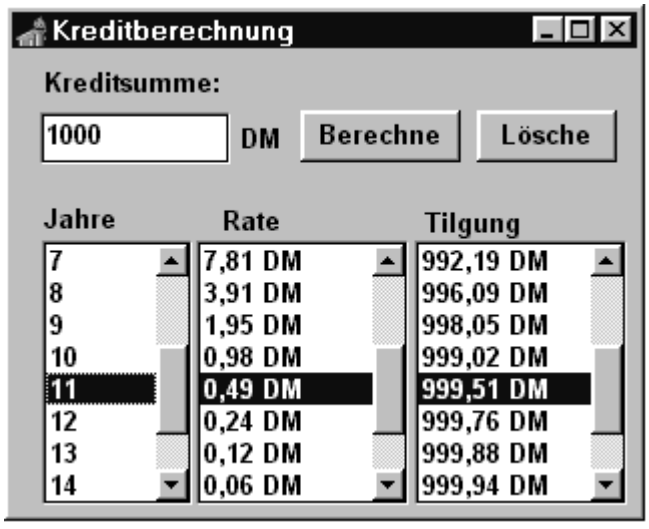
```
x := StrToInt(edit1.text);
f := 1;
REPEAT
  f := f * x;
  x := x-1
UNTIL x<1;
edit2.text := IntToStr(f);
```

b) Welche mathematischen Funktion wird hierbei realisiert ?

c) Welcher Wertebereich darf nicht zur Eingabe verwendet werden und weshalb?

3.5.4.2.3. Programmieraufgabe „Eigenwilliger Kredit“

a) Ein nicht gewinnsüchtiger Vater mit nennenswerten Ersparnissen trifft mit seinem finanzbedürftigen Sohn eine eigenwillige "Kreditvereinbarung":



Der Sohn erhält einen Kredit in gewünschter Höhe. Es werden keine Zinsen erhoben. Nach einem Jahr soll der Sohn die Hälfte der Kreditsumme zurückzahlen. Nach jedem weiteren Jahr beträgt die Rückzahlung jeweils die Hälfte der Rückzahlung des Vorjahres bis nach n Jahren der Kredit getilgt ist.

Man entwerfe einen Algorithmus (Struktogramm) und schreibe ein Programm, welches für beliebige Kreditsummen bei Rundung auf ganze Cent die Anzahl der Jahre ermittelt, die bis zur vollständigen Tilgung benötigt werden sowie die jährlich zu zahlende Rate auflistet.

Benötigte Variablen: *jahre* : ganzzahlig; *kredit, rate, tilgung* : reelle Zahlen.

Obenstehende Abbildung zeigt einen Vorschlag zur Oberflächengestaltung, wobei die Ausgabe über drei nebeneinander stehende TListBox-Komponenten erfolgt.

Lösungshinweise:

1. Synchronisation von TListBox-Komponenten
2. Ausgabe im Währungsformat

Lösungshinweise:

Synchronisation von TListBox-Komponenten

Falls nebeneinander angeordnete TListBox-Komponenten zusammen gehörende Wertereihen enthalten, erwartet der Nutzer eine gewisse Synchronität der Anzeige. Wird im obigen Beispiel etwa in der ListBox1 auf das Jahr Nr. 11 geklickt, so sollen auch in ListBox2 und ListBox3 die zugehörigen Einträge markiert erscheinen.

Die nachfolgende Prozedur passt den jeweiligen ItemIndex (markierten Eintrag) von ListBox2 und ListBox3 an den in ListBox1 markierten Eintrag an.

```
procedure TForm1.ListBox1Click(Sender: TObject);
  begin
    listbox2.itemindex:=listbox1.itemindex;
    listbox3.itemindex:=listbox1.itemindex;
  end;
```

Ausgabe im Währungsformat

Der Delphi-Hilfe ist unter dem Stichwort "FloatToStrF" u. a. folgendes zu entnehmen:

*Funktion **FloatToStrF**(Value: Extended; Format: TFloatFormat; Precision, Digits: Integer): String;*

Beschreibung

***FloatToStrF** konvertiert einen durch Value gegebenen Fließpunktwert in seine String-Darstellung.*
Der Parameter Format bestimmt das Format des resultierenden Strings.
Der Parameter Precision bestimmt die Genauigkeit des übergebenen Wertes. Er sollte für Werte vom Typ Single auf 7 oder weniger gesetzt sein, für Werte vom Typ Double auf 15 oder weniger und bei Extended-Werten auf höchstens 18.
Die Bedeutung des Parameters Digits hängt vom gewählten Format ab.

***ffCurrency** Währungsformat. Der Wert wird in einen String konvertiert, der eine*

Währungsbetrag angibt. Die Umwandlung wird durch die globalen Variablen CurrencyString, CurrencyFormat, NegCurrFormat, ThousandSeparator und DecimalSeparator gesteuert, die alle mit den Werten initialisiert werden, die in der Windows-Systemsteuerung, Abschnitt Ländereinstellungen angegeben wurden. Die Anzahl Ziffern nach dem Dezimalpunkt wird durch den Parameter Digits bestimmt und muss zwischen 0 und 18 liegen.

Im "Kreditprogramm" könnte das Währungsformat wie folgt in die Ausgabeschleife implementiert werden:

```
Listbox2.Items.Add (FloatToStrF (rate, ffCurrency, 8, 2));
```

b) Schon eher etwas für Fortgeschrittene:

Eine günstige Stunde ausnutzend, erreicht der Sohn folgende nicht ganz unwesentliche Modifizierung des Kreditvertrages:

Die Abzahlungen können in umgekehrter Reihenfolge stattfinden, d. h. im ersten Jahr erfolgt die Überweisung der kleinsten Rate. Dies steigert sich bis zum letzten Abzahlungsjahr, in dem die Hälfte der ursprünglichen Kreditsumme fällig wird.

Die Problemlösung von a) ist in Struktogramm- und Programmform entsprechend zu modifizieren!

---

zum Weiterüben:

1. Ein Bestand (z.B. eine Population oder ein Kapital) wachse um p% jährlich. Nach wie viel Jahren ist der doppelte Bestand erreicht?

2. Eine Firma sucht ein Programm zum Schreiben von Rechnungen. Es soll mehrere Einzelposten (Artikel und Einzelpreis) verarbeiten können, um anschließend den Gesamtbetrag auszuweisen.

3. Der Benutzer soll von einigen ihm auf dem Bildschirm gezeigten Zahlen entscheiden, ob sie durch 7 teilbar sind oder nicht. Gelingt ihm dies, hat er gewonnen, andernfalls verloren.

4. Ein Programm ist zu entwickeln, das folgendes leistet: Der Benutzer gibt ganze Zahlen ein; das Programm nennt die Anzahl sowie die größte und die kleinste der eingegebenen Zahlen.

5. In einem Teich leben X Forellen, doch ist Platz und Futter für K Forellen vorhanden. Die Forellen vermehren sich mit einer Jahresrate von p Prozent der Differenz zwischen Höchstbestand K und aktuellem Bestand X. Wie entwickelt sich die Population, wenn am Ende eines Jahres z Prozent des Bestandes entnommen werden.

6. Ein Fahrscheinautomat gibt Wechselgeld bis zu 9,99€ (in Münzen) zurück und kommt dabei mit möglichst wenig Münzen aus. (Beispiel: Bei einem Fahrpreis von 13,20€ und Eingabe eines 20€ Scheines werden drei 2€ Münzen, eine 50 Cent Münze und 3 10 Cent Münzen zurückgegeben.) Schreibe ein Programm, das dieses Verhalten nachahmt.