

Programmieren macht Spass!



Martin Schreiber

Version 0.1

Inhaltsverzeichnis

1. Die Rechenkünstlerin	5
1.1. Wir starten	5
1.2. Weiter geht's	10
1.3. Daten, was ist das?	14
1.4. Ein Bit ist zuwenig	16
1.5. Ein Byte ist nicht genug	19
2. Der Logikakrobat	23
2.1. Die Frage	23
2.2. Die Antwort?	24
2.3. Optimierungen	25
2.4. Es funktioniert nicht!	31
3. Der PanoRamatograph, wir simulieren	33
3.1. Die Idee	33
3.2. Die Ausführung	35
3.3. Die Politur	38
3.4. Die Optimierung	39
4. Das Sprachgenie	42
4.1. Programmaufbau	42
4.2. Unitaufbau	42
4.3. Was macht der Compiler?	47
4.4. Texte in Programmen	47
4.5. Gewinnung eines Ziffernwertes	49
4.6. Umwandlung der gesamten Zahl	50
5. Die Aufzugssteuerung	55
5.1. Darstellung	55
5.2. Mausklick	59
5.3. Stockwerke zeichnen	63
5.4. Mausbehandlung für die Stockwerke	63
5.5. Aufzugsmotor	67
5.6. Motorregelung	70
5.7. Steuerung	70
5.8. Mängel	70

A. Installation der Entwicklungsumgebung	78
B. Verweise	79

1. Die Rechenkünstlerin

1.1. Wir starten

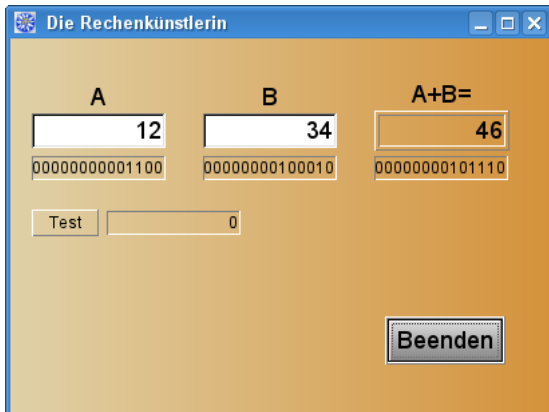


Bild 1: Die Rechenkünstlerin.

Als erstes machen wir ein Programm, welches zwei Zahlen zusammenzählen kann (Bild 1).

Zum Programmieren braucht man Werkzeuge. Zange, Hammer und Sichel sind nicht geeignet, wir brauchen Werkzeug-Programme. Unser Programmierwerkzeug ist MSEide. Wird MSEide das erste mal gestartet, sieht es aus wie in Bild 2.

Zuerst legen wir für die Rechenkünstlerin ein Projekt an, dazu verwenden wir eine Vorlage (Template). Klicke 'Project'- 'New'- 'From Template' (Bild 3, Bild 4).

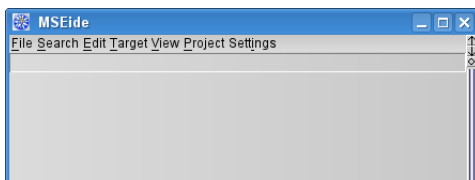


Bild 2: MSEide.

Wir wählen **default.prj** (default meint standard).

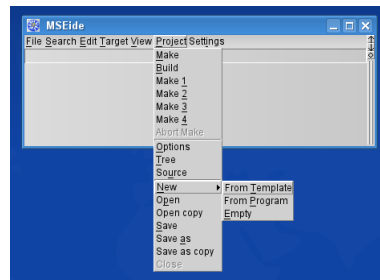


Bild 3: Neues Projekt.

Im nächsten Eingabefenster erstellen wir mit New dir ein Verzeichnis (Bild 5), nennen wir es mseguiprojekte. In mseguiprojekte erzeugen wir mit New dir ein Unterverzeichnis **rechenkunst** (Bild 6). Das Projekt erhält den Namen **rechenkunst** (Bild 7). Klicke **OK**, nun wird unser erstes Projekt angelegt (Bild 8).

Oben links die "Kommandozentrale". MSEide kann mit **File-Exit** geschlossen werden. MSEide merkt sich, mit welchem Projekt zuletzt gearbeitet wurde und lädt es beim nächsten Start automatisch. Die Auswahl eines anderen Projektes geschieht durch **Project-Open** (öffnen). Die zuletzt geöffneten Projekte werden durch Klick auf den Ab-Pfeil bei **Name** zur Auswahl aufgelistet (Bild 9).

Statt durch Maus-Klicken können Menüpunkte auch durch Drücken der Alt-Taste und der entsprechenden Taste des unterstrichenen Buchstabens aktiviert werden. Beispielsweise geschieht das Schliessen von MSEide auch durch drücken von **Alt+F** - **Alt+X**. Diese Art der Menübedienung ist

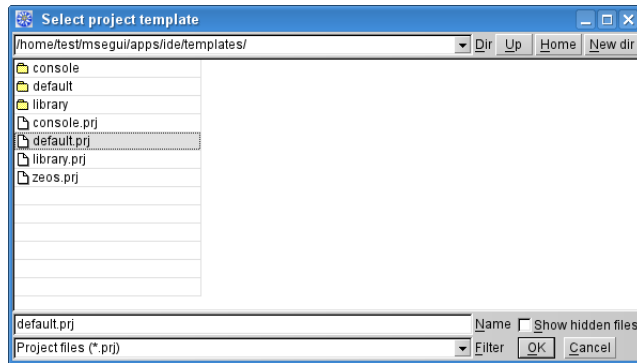


Bild 4: Vorlage auswählen..

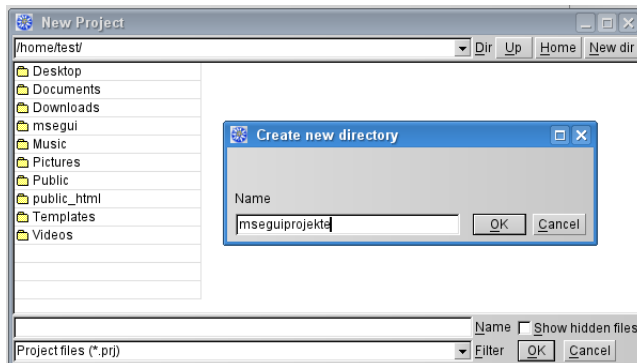


Bild 5: Verzeichnis für MSeGui-Projekte erstellen.

weniger ermüdend als stundenlanges Maus-schubsen.

Um auch grössere Arbeiten rasch ausführen zu können, verwenden wir vorgefertigte Programmkomponenten. Unsere Komponentenbibliothek heisst MSeGui. In **Component Palette** stehen die Komponenten zur Verfügung. Zuerst benötigen wir eine Komponente zur Anzeige des Rechnungsergebnisses. Fahre mit der Maus auf die einzelnen Komponenten, in einem kleinen Fenster erscheint die Bezeichnung der entsprechenden Komponentenart. Aus der Abteilung **Widget** nehmen wir **tintegerdisp**, die 5. Komponente von links, sie ist mit 123 bezeichnet. 'Widget' ist ein Kunstwort und bedeutet 'window gadget' (Fenster-Objekt). Klicke auf das **tintegerdisp** 123-

Symbol in **Component Palette**, es wird "eingedrückt" (Bild 10).



Bild 10: tintegerdisp markieren.

In der Bildschirmmitte hat MSeide bereits ein leeres Formular bereitgestellt. Falls es nicht sichtbar ist, drücke F12 oder benütze das Menü der Kommandozentrale **View-main.mfm**. Klicke in das Formular, die **tintegerdisp** Komponente wird in das Formular kopiert (Bild 11).

Die Programmkomponenten besitzen Eigenschaften ('properties'), welche im Fens-

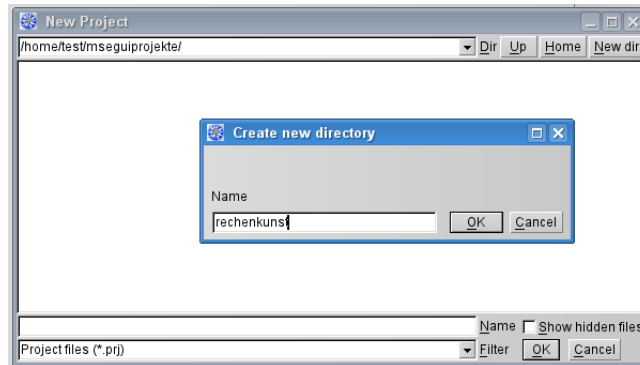


Bild 6: Unterverzeichnis rechenkunst.

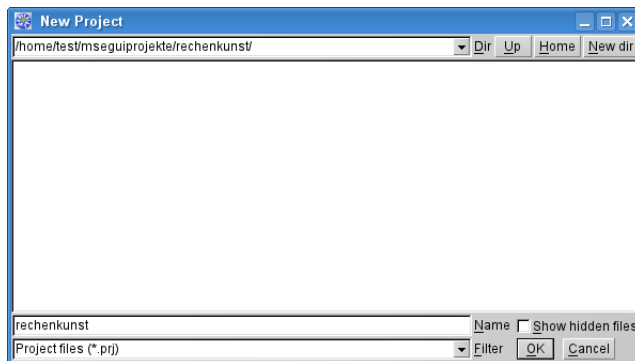


Bild 7: Projekt rechenkunst.

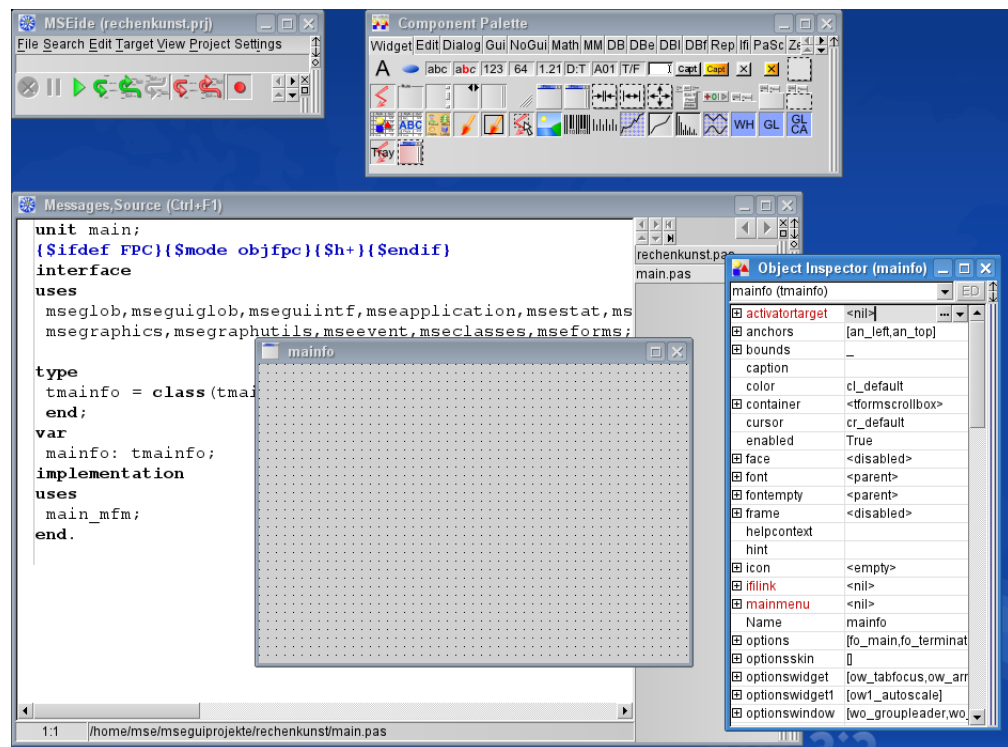


Bild 8: Unser erstes Projekt.

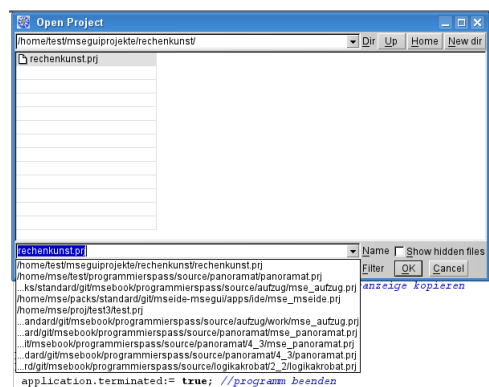


Bild 9: Projekt öffnen.

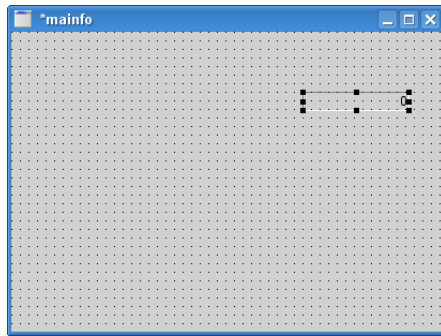


Bild 11: tintegerdisp im Formular.

ter Object Inspector verändert werden können. Der Objektinspektor zeigt jeweils die Eigenschaften der im Formular aktivierten Komponenten. Das Aktivieren geschieht durch Maus-links-Klick auf die Komponente. Das Objektinspektor Fenster wird durch F11 oder View-Object Inspector oder Alt+V-Alt+I sichtbar gemacht, wobei F11 zwischen Formular und Objektinspektor umschaltet. Doppelklick auf eine Komponente aktiviert den Objektinspektor ebenfalls. Im Objektinspektor sind in der linken Spalte die Namen der Eigenschaften aufgeführt, in der rechten Spalte die Werte. Als erstes setzen wir den Wert von Name auf `ergebnis`, damit wir die Komponente im Programm ansprechen können (Bild 12). Stelle sicher, dass die Komponente mit dem ursprünglichen Namen `tintegerdisp1` aktiviert ist, drücke nach der Eingabe die Enter-Taste um sie abzuschliessen. Solange Enter nicht gedrückt und ein Eingabefeld noch nicht verlassen wurde, kann eine versehentliche Eingabe durch drücken der Esc-Taste rückgängig gemacht werden.

Namen müssen mit `a..z`, `A..Z` oder `_` beginnen und dürfen nur `a..z`, `A..Z`, `_` und `0..9` enthalten, Umlaute sind nicht gestattet! Gross- und Kleinschreibung wird nicht unterschieden. `ERGBNIS` `ergebnis` und `ErGeBnIs` bedeuten dasselbe.

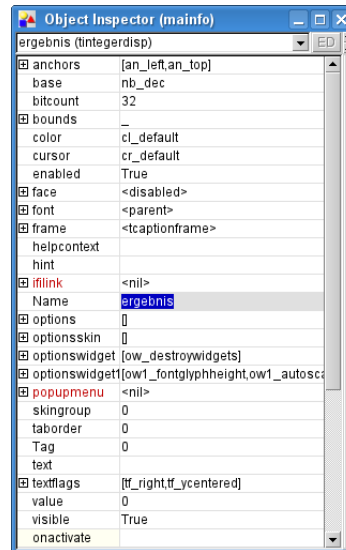


Bild 12: Einstellung von Name im Objektinspektor.

Nun definieren wir noch die Fensterüberschrift unseres Programmes und schon kann die Rechenkünstlerin gestartet werden! Die Fensterüberschrift ist die Eigenschaft `caption` (Beschriftung, Titel) des Formulars. Klicke in die freie Fläche des Formulars, der Objektinspektor zeigt in der Überschrift `maininfo` (`tmaininfo`). Gib in der rechten Spalte bei `caption` Die Rechenkünstlerin ein (Bild 13). In der Kommandozone befinden sich am unteren Rand eine Anzahl Knöpfe zur Programmbedienung. Zum Starten unseres Programmes klicken wir auf den grünen Pfeil, die Rechenkünstlerin wird in Maschinensprache übersetzt und gestartet (Bild 14, Bild 15).

Ein Klick auf das schwarze Kreuz im roten Kreis der Kommandozone bricht das laufende Programm ab (Bild 16). Sollten diese Knöpfe nicht sichtbar sein, View-Debugger (oder Alt+V-Alt+D) bringt sie wieder zum Vorschein. Das Programm lässt sich auch mit der F9-Taste starten.

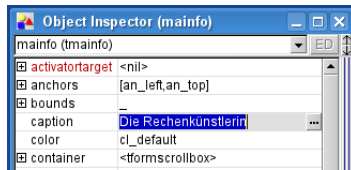


Bild 13: Fensterüberschrift

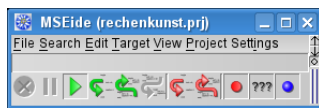


Bild 14: Programm Start

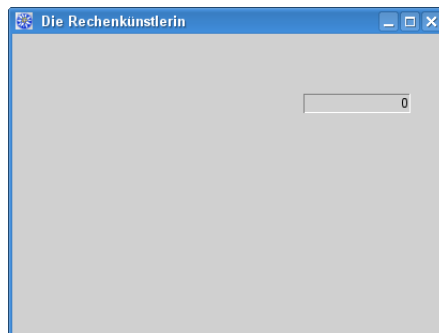


Bild 15: Erster Start der Rechenkünstlerin

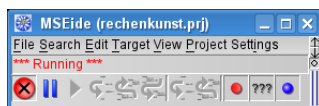


Bild 16: Programm abbrechen.

- Die Entwicklungsumgebung heisst MSEide.
- Die Komponentenbibliothek heisst MSEgui.
- MSEide wird mit Alt+F-Alt+X beendet.
- Namen müssen mit a..z, A..Z oder _ beginnen und dürfen nur a..z, A..Z, _ und 0..9 enthalten, Umlaute sind nicht gestattet.
- Gross- und Kleinschreibung wird nicht unterschieden.
- Programmstart mit dem grünen Pfeil in der Kommandozone oder F9.
- Programmabbruch mit dem schwarzen Kreuz im roten Kreis.

1.2. Weiter geht's

Das Anzeigefeld soll eine Beschriftung erhalten. Doppelklicke im Formular auf die **ergebnis** Komponente, der Objektinspektor wird mit den Eigenschaften der **ergebnis** Komponente geöffnet. Klicke auf das + bei **frame** (Rahmen), es erscheinen die Rahmeneigenschaften. Gib bei **caption** **A+B=** ein (Bild 17). Die Bezeichnung erscheint auf dem Formular (Bild 18).

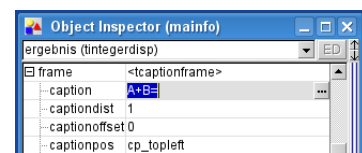


Bild 17: Beschriftung der ergebnis Komponente.

Den Rahmen um **ergebnis** machen wir mit den **frame** Eigenschaften **framewidth**

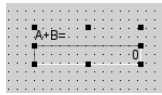


Bild 18: Resultat der Beschriftung der ergebnis Komponente.

= 2, leveli = -1 und levelo = 1 deutlicher (**Bild 19**). Der "Trauerrand" gefällt uns nicht, daher wählen wir in colorframe cl_transparent. ergebnis sieht nun aus wie in **Bild 20**. Leider bleibt jetzt nicht mehr genug Raum für die Höhe der Ziffern in der ergebnis Komponente. Die Funktion Rechts-Klick auf die Komponente, Sync. to Font Height (passe an die Schrifthöhe an) bringt die Sache wieder in Ordnung (**Bild 21**).

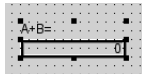


Bild 19: Trauerrand.

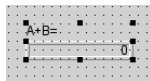


Bild 20: Rahmen von ergebnis.

Die Ausgabe der Rechenkunst ist nun soweit einsatzbereit. Fügen wir zur Eingabe der zu addierenden Zahlen (A und B) zwei Eingabefelder hinzu. Eingabefelder befinden sich in Component Palette in der Abteilung Edit (bearbeiten), wir nehmen tintegeredit, die vierte Komponente mit der 123 Bezeichnung (**Bild 22**). Platziere zwei Exemplare von tintegeredit auf dem Formular (**Bild 23**).

Im Objektinspektor setzen wir die Name Eigenschaft des linken Eingabe Feldes auf a, die des rechten Feldes auf b. frame.caption setzen wir links auf A, rechts auf B. Starte das Programm, nun können wir bereits Zahlen eingeben (**Bild 24**). Gerechnet wird allerdings noch nichts. Statt das

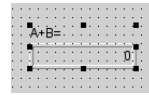


Bild 21: ergebnis nach Höhenanpassung.

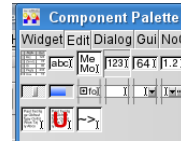


Bild 22: Eingabefeld tintegeredit.

Programm in der Kommandozone abbrechen, kann auch das Die Rechenkünstler-Fenster geschlossen werden.

Wie soll es weitergehen? Sobald die Anwenderin eine Zahl in die Eingabefelder a oder b eingibt, müssen die eingegeben Zahlen zusammengezählt und das Resultat im Feld ergebnis angezeigt werden. Aktiviere a. Im Objektinspektor gibt es zuunterst eine Reihe von farblich hervorgehobenen Eigenschaften, deren Namen mit on beginnen. Das sind die Ereigniseigenschaften. Gib in der rechten Spalte bei ondataentered dataentered (Daten eingegeben) ein, drücke Enter (**Bild 25**).

Drücke ein zweites Mal die Enter-Taste. MSEide hat ein Stück Programm angelegt, welches nach dem Eingeben einer Zahl in die Komponente a aufgerufen wird (**Bild 26**).

In der Mitte unter type bei tmainfo sehen wir auch die in das Formular kopierten Komponenten aufgeführt. Auf die Komponenteneigenschaften können wir nicht nur im Objektinspektor sondern auch im Pro-

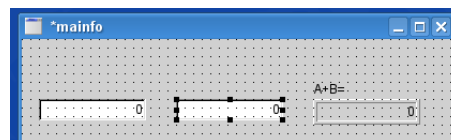


Bild 23: Eingabefelder.

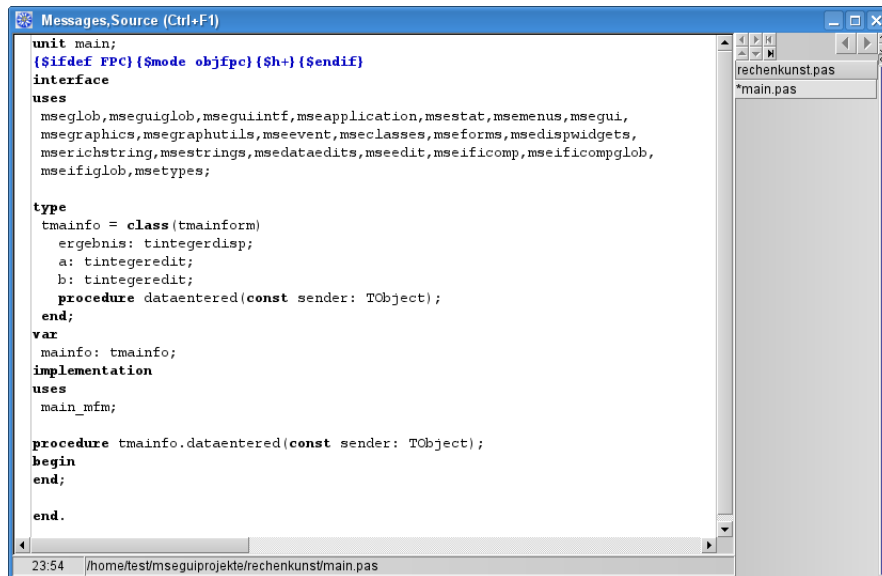


Bild 26: Programmtext für das Ereignis ondataentered.

```

procedure tmaininfo.dataentered(const sender: TObject);
begin
  ergebnis.value:= a.value + b.value;
end;

```

Listing 1: Berechnung der Summe

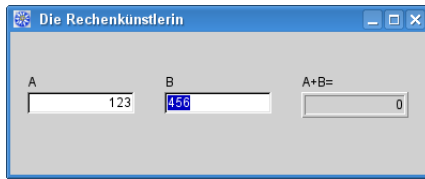


Bild 24: Zahleneingabe.

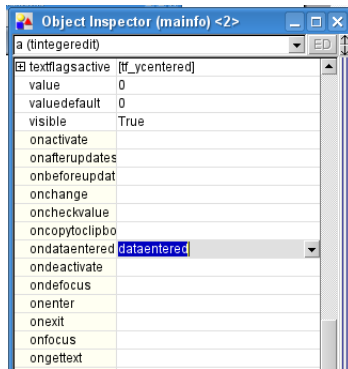


Bild 25: Ereignisseigenschaft ondataentered

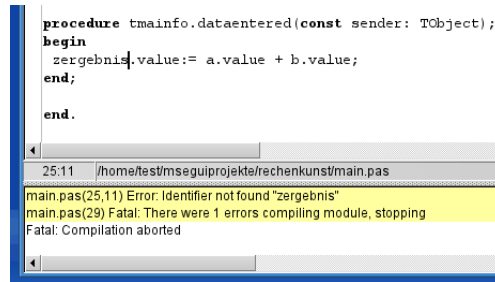


Bild 27: Compilierungsfehler.

gramm zugreifen. Dazu schreiben wir zuerst den Komponentennamen, dann einen Punkt und danach den Namen der Eigenschaft. Die eingegebene Zahl steht in einer `tintegeredit` Komponente in der Eigenschaft `value`. Eine `tintegerdisp` Komponente zeigt die in der Eigenschaft `value` gespeicherte Zahl an. Um der Rechenkünstlerin zur Kunst zu verhelfen schreiben wir

```
ergebnis.value:= a.value + b.value;
```

(siehe Listing 1). `:=` bezeichnet eine Wertzuweisung, ein Befehl wird mit `;` abgeschlossen. Starte das Programm mit F9.

Falls du dich bei der Eingabe von `ergebnis.value:= a.value + b.value;` vertippt haben solltest, reklamiert der Compiler (Bild 27).

Der Compiler ist das Werkzeugprogramm, welches unsere Kommandos in für den Computer verständliche Anweisungen übersetzt, wir werden später darauf zurückkommen. Korrigiere die Fehler und drücke F9 erneut.

Gib in der Eingabemaske bei A 123 ein, drücke Enter, in A+B= erscheint 123 ($123 + 0 = 123$). Eine Eingabe in b bewirkt allerdings noch nichts. Das ist ja auch kein Wunder, da wir für das `b.ondataentered` Ereignis noch gar keine Aktion eingetragen haben. Holen wir es gleich nach. Beende die Rechenkünstlerin, aktiviere im Formular die b Komponente, gehe im Objektspektor zu `ondataentered` - es befindet sich im unteren Teil der Liste. Zur Behandlung der Dateneingabe in b können wir die selbe Funktion wie für b verwenden. Klicke in die rechte Spalte bei `ondataentered`, klicke auf den Abwärtspfeil und wähle `dataentered` aus. `ergebnis.value:= a.value + b.value;` wird nun für Eingaben sowohl in a als auch in b ausgeführt. Teste es!

- Die Eigenschaft `Name` bestimmt die Bezeichnung unter der auf die Komponenten zugegriffen werden kann.
- Das `ondataentered` Ereignis wird für Dateneingabekomponenten aufgerufen, nachdem die Anwenderin einen Wert eingegeben und bestätigt hat.
- `.` trennt die Eigenschaftsnamen von den Namen der Komponenten.
- `:=` bezeichnet eine Wertzuweisung zu einer Eigenschaft.
- `;` schliesst ein Kommando ab.

1.3. Daten, was ist das?

Daten sind gespeicherte Informationen. Zur Speicherung von Daten müssen dazu passende Gefässe benutzt werden. Zur Speicherung für Daten in Form von Steinen könnten wir z.B. ein Sieb verwenden. Für flüssige Daten wäre das Sieb weniger geeignet. ;-)

Heutige Computer verwenden zur Darstellung von Information elektrische oder magnetische Signale. Gespeichert werden sie z.B. in elektronischen Bausteinen (Chips) oder auf Magnetplatten (Harddisk). Ebenfalls üblich sind optische Informationen auf Scheiben (CD-ROM, DVD...).

Die kleinste mögliche Informationseinheit ist ein Bit. Ein Bit kann gerade mal speichern, ob etwas wahr oder falsch ist, es hat also nur zwei Zustände. Den einen Zustand bezeichnen wir mit `false` (falsch), den anderen mit `true` (wahr).

Eine solche Informationsgrundeinheit

verwenden wir nun um die Rechenkünstlerin eleganter zu beenden. Die MSEgui Komponentenbibliothek hat eine immer vorhandene und automatisch erzeugte Komponente namens `application` (Anwendung), welche das Programm organisiert. `application` besitzt die Eigenschaft `terminated` (beendet). Wird `application.terminated` auf `true` gesetzt, wird das Programm beendet.

Ausgelöst werden soll das Beenden durch Klicken auf eine Schaltfläche ('Button'). Setze ein `tbutton` (die graue mit `Capt` bezeichnete Komponente) aus dem Bereich `Widget` der Komponenten-Palette auf das Formular. Ändere im Objektinspektor `Name` auf `beenden`, setze `caption` auf `Beenden`. Gib bei `onexecute beendenexe` ein, drücke zwei mal `Enter`. MSEide hat ein Programmstück erzeugt, welches beim drücken des Knopfes ausgeführt werden wird. Wir programmieren `application.terminated:=true;` (**Listing 2**). Beachte auch den mit `//` abgetrennten Kommentar. Alles nach `//` in der Zeile ist Kommentar und wird vom Compiler nicht berücksichtigt. Auch Text zwischen `{` und `}` ist Kommentar, diese Kommentarform kann mehrere Zeilen enthalten.

Bitte teste, ob dein Programm beendet wird, wenn du auf den `Beenden`-Knopf klickst.

Machen wir das Aussehen der `beenden`-Schaltfläche hübscher. Aktiviere die `beenden`-Komponenten im Formular, ziehe mit der Maus an den kleinen schwarzen Quadraten um die Komponente grösser zu machen. Zuerst erzeugen wir einen Rahmen um den Knopf. Der Rahmen einer Komponente wird durch die Eigenschaft `frame` bestimmt. Alle visuellen Komponenten haben diese Eigenschaft. Bei `a`, `b`, und `ergebnis` haben wir `frame` bereits benutzt. `tbutton`

```

procedure tmaininfo.beendenexe(const sender: TObject);
begin
  application.terminated:= true; //programm beenden
end;

```

Listing 2: Programm beenden.

benötigt `frame` normalerweise nicht, darum müssen wir `frame` zuerst aktivieren. Klicke im Objektinspekttor auf `<disabled>` in der Zeile `frame` und danach auf den erschienenen kleinen Knopf mit dem `...`-Symbol. `<disabled>` ändert sich zu `<tcaptionframe>`, die Eigenschaft wurde erzeugt. Durch einen weiteren Klick auf `...` wird sie nach Bestätigung wieder gelöscht. Klicke auf das `+` bei `frame`, um den “Baum” zu öffnen.

`frame` besitzt einige Eigenschaften, welche die Rahmengestalt beeinflussen, z.B. Rahmenhöhe und Rahmenbreite, sie werden in ‘Pixel’ angegeben. Das Bild auf dem Computerbildschirm wird aus in regelmässigen Abständen angeordneten kleinen Punkten zusammengesetzt. Wenn du ganz genau hinschaust oder eine Lupe zur Hand nimmst, siehst du sie. Wenn du noch genauer hinschaust, siehst du, dass ein einzelner Punkt jeweils einen roten grünen und blauen Bereich hat. Dies wird zur Erzeugung der gewünschten Farbe durch Mischung verwendet. In einem weissen Bildschirmbereich sind die drei Bereiche gleich hell - wir werden darauf zurückkommen. Ein solcher rot-grün-blau Punkt wird ‘Pixel’ genannt.

Die `frame`-Eigenschaft `levelo` ist die “Höhe” des äusseren (‘outer’) Rahmens in Pixel. Wir setzen sie auf 1. `leveli` bestimmt die Höhe des inneren Rahmens, wir tragen -1 ein, damit wieder die Anfangshöhe erreicht wird. `framewidth` bezeichnet die Rahmenbreite in Pixeln, wir belassen sie bei 0. Unser beenden Button sieht nun aus wie in

Bild 28.



Bild 28: Rahmen um Schaltfläche

Eine weitere Eigenschaft aller sichtbaren Komponenten ist `face` (Gesicht, Oberfläche). Wir verwenden `face` um dem Button eine gewölbte Oberfläche zu verpassen. Wie `frame` muss `face` zuerst durch Klick auf `...` aktiviert werden. Aktiviere `face` der `beenden`-Komponente und öffne die `face`-Eigenschaften. Innerhalb von `face` öffne `fade` (meint Farbverlauf). Klicke in die Mitte der rechten Spalte bei `color.count`. Klicke auf den kleinen `...`-Knopf, es öffnet sich das Fenster des Farbverlauf-Editors (**Bild 29**). Klicke in die Mitte des waagrechten weiss-grau-schwarz-Bereiches, es wird ein grauer Farbverlauf angelegt (**Bild 30**). Klicke `OK`. Der Farbverlauf wird auf dem Button dargestellt (**Bild 31**). Um die Illusion einer erhabenen Fläche zu erzielen, setzen wir `face.fade_direction` auf `gd_down` (**Bild 32**).

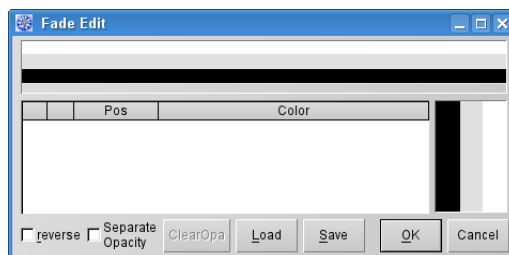


Bild 29: Farbverlaufs-Editor

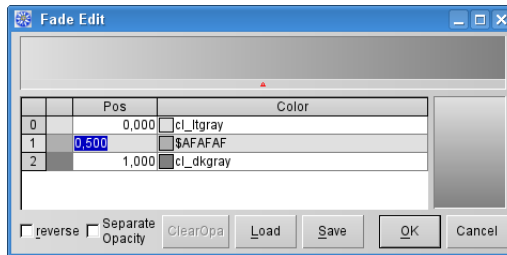


Bild 30: Farbverlauf.



Bild 31: Farbverlauf auf Schaltfläche

- Die Informations-Grundeinheit heisst Bit.
- Ein Bit kann zwei Zustände darstellen, z.B. wahr und falsch oder 0 und 1.
- Die Punkte aus denen das Bild des Computermonitors zusammengesetzt ist werden Pixel genannt.
- Das Ereignis `tbutton.onexecute` wird beim Klicken der Schaltfläche ausgeführt.
- Alle visuellen MSEgui Komponenten besitzen die Eigenschaft `frame`, welche das Aussehen des Rahmens bestimmt.
- Alle visuellen MSEgui Komponenten besitzen die Eigenschaft `face`, welche das Aussehen der Oberfläche bestimmt.
- `frame` und `face` müssen gegebenenfalls vor Verwendung aktiviert werden.
- Text nach `//` bis zum Zeilenende ist Kommentar.
- Text zwischen `{` und `}` ist Kommentar.

1.4. Ein Bit ist zuwenig

Wir haben gesehen, dass die Informations-Grundeinheit Bit lediglich zwei Zustände einnehmen kann. Die Rechenkünstlerin kann aber mit mehr Zuständen umgehen, z.B. kann sie 12 zu 34 addieren und das Ergebnis (46) darstellen. Wie macht sie das? Man könnte auf die Idee kommen, zwei Bits zu einer Gruppe zu kombinieren. Dann kann das erste Bit zwei Zustände einnehmen und das Zweite unabhängig davon ebenfalls zwei.

Zustand	Bit 1	Bit 0
0	0	0
1	0	1
2	1	0
3	1	1

Nun können wir bereits zwei Zustände von Bit 0 mit Bit 1 = 0 und zusätzlich die zwei Zustände von Bit 0 mit Bit 1 = 1 das heisst $2 * 2 = 4$ verschiedene Zustände darstellen! Wenn wir ein weiteres Bit hinzunehmen sind es $2 * 2 * 2 = 8$ Zustände.

Zustand	Bit 2	Bit 1	Bit 0
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

Wenn wir die darstellbaren Zustände als Zahlen auffassen, geben wir den Bits verschiedene Wertigkeiten. Wie bei unseren gewöhnlichen Zahlen im Zehnersystem bekommt die am meisten rechts stehende Stelle die kleinste Wertigkeit; statt verzehnfacht wird Wertigkeit von Stelle zu Stelle verdoppelt.



Bild 32: Gewölbte Schaltfläche.

Bitnummer	Wertigkeit	=
0	1	2^0
1	2	2^1
2	4	2^2
3	8	2^3
4	16	2^4
5	32	2^5
6	64	2^6
7	128	2^7
8	256	2^8
9	512	2^9

Um das Resultat von 46 darstellen zu können, benötigen wir mindestens 6 Bit. 5 Bit ($1 + 2 + 4 + 8 + 16 = 31$) ist zu wenig, 6 Bit ($31 + 32 = 63$) reichen. Diese Art der Zahlendarstellung wird Binärsystem genannt (binär = Zweiwertig).

Wir können der Rechenkünstlerin beim Verwenden des Binärsystems auf die Finger schauen. Platziere je ein `tintegerdisp` unter `a,b` und `ergebnis` auf dem Formular - zur Erinnerung, `tintegerdisp` ist im dem Bereich Widget der Komponenten-Palette und mit 123 bezeichnet. Ändere Name auf `bina`, `binb` respektive `binergebnis`. Klicke im Formular auf die `bina`-Komponente, sie wird aktiviert. Klicke mit gedrückter Ctrl-Taste auf `binb`, `binb` ist nun ebenfalls aktiviert. Klicke mit gedrückter Ctrl-Taste auf `binergebnis`, nun sind alle drei Komponenten aktiviert (**Bild 33**).

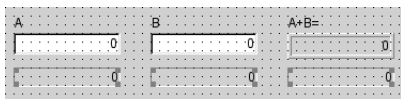


Bild 33: Mehrere Komponenten aktiviert.

Im Objektinspektor setze `base` (Basis) auf `nb_bin`, dies wirkt sich auf alle aktivierten Komponenten aus, sie zeigen nun die Werte im Binärsystem an. Im `dataentered` Programmabschnitt ergänzen wir die

Befehle zur Übertragung der Werte (siehe **Listing 3**).

Das Programm in **Listing 4** ist falsch. Warum? Hier wird der Wert von `ergebnis` kopiert bevor das Ergebnis überhaupt berechnet wurde! Die Werte von `ergebnis` und `binergebnis` werden also nicht übereinstimmen, wenn sich der Wert der Berechnung ändert.

Starte dein Programm, um beim binären Rechnen zuzusehen (**Bild 34**). Wer will kann das binäre Rechnen auf dem Papier nachvollziehen, es funktioniert wie unser übliches schriftliche Zusammenzählen, nur einfacher. Computer können nicht viel, sie sind nur sehr schnell. Viele einfache Dinge sehr schnell erledigt ergibt schlussendlich trotzdem eine beeindruckende Leistung. Leider kann ein Computer auch sehr schnell sehr viele Fehler machen, falls er schlecht programmiert ist. ;-)

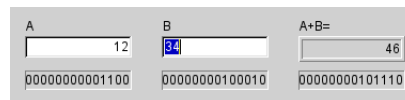


Bild 34: Binärsystem.

Eine Gruppe von 8 Bits wird als **byte** oder Oktet bezeichnet. Die meisten Computer arbeiten mit Datengrößen von Verdoppelungen von Bytes. Warum sich gerade 8 Bit als Basis eingebürgert hat, kann auf Wikipedia nachgelesen werden. Ein 64 Bit-Computer ist zur Verarbeitung von 8 Byte Dateneinheiten eingerichtet, ein 32 Bit-Computer für 4 Bytes grosse Einheiten. Auch ein 32 Bit-Computer kann 64 Bit Werte verarbeiten, er muss die Daten lediglich in 32 Bit Häppchen aufteilen.

Wir werden nun mit Datenwerten experimentieren. Setze einen weiteren `tbutton` auf das Formular. `name = test`, `caption = Test`, `onexecute = testexe`. Daneben platziere ein `tintegeredit`, `name = testergebnis`.

```

procedure tmainfo.dataentered(const sender: TObject);
begin
    ergebnis.value:= a.value + b.value;
    binergebnis.value:= ergebnis.value; //werte in binaeranzeige kopieren
    bina.value:= a.value;
    binb.value:= b.value;
end;

```

Listing 3: Werte in Binäranzeige kopieren.

```

procedure tmainfo.dataentered(const sender: TObject);
begin
    binergebnis.value:= ergebnis.value; //werte in binaeranzeige kopieren
    bina.value:= a.value;
    binb.value:= b.value;
    ergebnis.value:= a.value + b.value; //falsche reihenfolge!
end;

```

Listing 4: Falsche Reihenfolge

Das Programmstück `testexe` erweitern wir zu **Listing 5**.

Wenn wir Datenspeicher (Variablen) verwenden wollen, müssen wir das dem Compiler mitteilen. Variablen werden vor den Befehlen in einem mit `var` bezeichneten Bereich definiert. Eine Variablendefinition hat die Form `variablenname : datatype ;`. Für Variablennamen gelten die gleichen Einschränkungen für die erlaubten Schriftzeichen wie für Komponentennamen. Für `datatype` können nur dem Compiler bekannte Bezeichner verwendet werden, da ist er unerbittlich. Der Befehlsteil beginnt mit `begin` und endet mit `end`; . Bitte beachte auch die Einrückungen unter `var` und zwischen `begin` und `end`. Sie sind zwar nicht obligatorisch, erleichtern jedoch das Verständnis der Programme.

Starte das ergänzte Programm, gib verschiedene Zahlen in A ein (nicht vergessen, die Eingabe durch **Enter** abzuschliessen!), klicke **Test** um die Variablenzuweisung auszuführen. Solange die Zahlen kleiner als 256 sind, geht alles gut. Testen wir aber mit 256, wird 0 angezeigt! Was ist passiert?

Die Variable `by1` ist vom typ `byte`, das heisst, sie ist eine Gruppierung von 8 Bits. Zur Darstellung von 256 bräuchten wir 9 Bits ($0+0+0+0+0+0+0+0+256 = 256$), in Binärdarstellung 100000000, wir haben aber nur 8 Bits, die führende Eins wird abgeschnitten, übrig bleibt 0! Solche Bereichsüberläufe in Steuerprogrammen lassen schon Raketen explodieren.

Testen wir mit 257 ergibt sich das Resultat 1 ($257 = \text{binär } 100000001$, ohne neuntes Bit $00000001 = 1$).

```

procedure tmainfo.testexe(const sender: TObject);
var
  by1: byte;
begin
  by1:= a.value;
  testergebnis.value:= by1;
end;

```

Listing 5: Byte Variable.

- Ein Byte ist eine Gruppe von 8 Bits, welche maximal $2^8 = 256$ verschiedene Werte darstellen kann. Der darstellbare Zahlenbereich ist 0..255.
- Ein Variablendefinitionsbereich wird mit **var** eingeleitet.
- Der Wert von Variablen ist ohne Zuweisung undefiniert und enthält zufällige Werte.
- Der Befehlsbereich wird mit **begin** eingeleitet und durch **end**; abgeschlossen.

1.5. Ein Byte ist nicht genug

Einen weiteren Aspekt der Datenspeicher für Zahlen müssen wir berücksichtigen. Mit einem **byte** können Zahlen von 0 bis 255 dargestellt werden. Was aber, wenn wir negative Zahlen benötigen? Eine Lösung heisst "Zweierkomplement". Für 0 bis 127 bleibt alles wie gehabt (Binärdarstellung 00000000 bis 01111111). Für -1 nehmen wir von binär 00000000 1 weg. Was ergibt sich? 11111111. ($0 - 1 = 1$, behalte -1, $0 - 1 = 1$, behalte -1...). Gegenprobe: $11111111 = 255 + 1 = 256 = 100000000$, neuntes Bit abgeschnitten = 00000000. :-)

-2 ist dementsprechend 11111110, -3 = 11111101, -127 = 10000001, -128 = 10000000 = 2^7 . Wenn wir noch einen

Schritt weiter gehen, bekommen wir für -129 01111111. Das ist aber wieder dieselbe Bitkombination wie wir für +127 vorgesehen haben und darf daher nicht verwendet werden. Der darstellbare Bereich einer 8 Bit Zahl kann dementsprechend auch -128 bis 127 betragen. Am höchstwertigen Bit sehen wir, ob eine Zweierkomplementzahl negativ ist. Variablen welche die Bits in dieser Art verwenden werden als **integer**-Zahlen bezeichnet. Variablen welche nur positive Werte darstellen können werden **cardinal**-Zahlen genannt.

Je nach Aufgabe haben wir verschiedene Variablentypen zur Verfügung, siehe **Tabelle 1.1**. In der Regel werden **card32** (= **cardinal**) und **int32** (= **integer**) benutzt, da unsere Computer damit gut arbeiten können und der Wertebereich für die meisten Aufgaben ausreicht.

Die Eingabefelder **a** und **b** sind vom Typ **tintegeredit** und können **int32** (**integer**) Werte verarbeiten. Im Moment ist der Eingabebereich auf positive Werte begrenzt. Falls du auch negative Werte eingeben möchtest, ändere die **min** Eigenschaften von **a** und **b** auf -214783648.

Zum Abschluss machen wir den langweiligen grauen Hintergrund der Rechenkünstlerin etwas ansehnlicher. Das Formular hat ein eingebettetes Widget, worin wir unsere Komponenten platziert haben, erreichbar ist es über die Eigenschaft **container** des Formulars. Doppel-klicke in die Formular-

Bitzahl	cardinal -Bereich	integer -Bereich	Cardinal Typen- Bezeichnung	Integer Typen- Bezeichnung
8	$0..2^8 - 1 =$ 0..255	$-2^7..2^7 - 1 = -128..127$	card8, byte	int8
16	$0..2^{16} - 1 =$ 0..65535	$-2^{15}..2^{15} - 1 =$ -32768..32767	card16	int16
32	$0..2^{32} - 1 =$ 0..4294967295	$-2^{31}..2^{31} - 1 =$ -214783648..214783647	card32, cardinal	int32, integer
64	$0..2^{64} - 1 =$ 0.. 18446744073709551615	$-2^{63}..2^{63} - 1 =$ -9223372036854775808.. 9223372036854775807	card64	int64

Tabelle 1.1.: Zahlen Bereiche.

fläche, der Objektinspektor wird mit dem Formular als bearbeitete Komponente aktiviert. Öffne die **container**-Eigenschaft.

Wie alle visuellen MSEGUI-Komponenten hat auch das Container-Widget eine **face**-Eigenschaft, welche wir zur Gestaltung des Hintergrundes verwenden können. Erzeuge für **container.face** einen Farbverlauf wie bei 1.3 auf Seite 15 für die Beenden-Schaltfläche beschrieben (**Bild 35**).

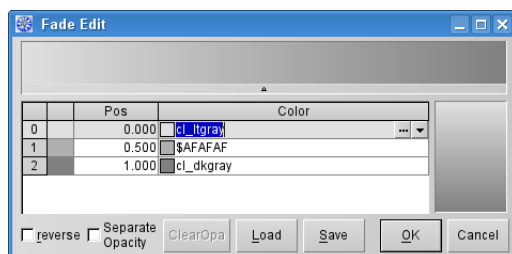


Bild 35: Grauer Farbverlauf.

Statt grau möchten wir eine buntere Farbe haben. Klicke auf ... in der **Color**-Spalte der Zeile 0, der Farben-Editor öffnet sich (**Bild 36**).

Wir haben verschiedene Möglichkeiten, die gewünschte Farbe zu wählen. Bei **red**, **green** und **blue** können wir die Helligkeit der einzelnen Farb-Teilflächen der Pixel einstel-

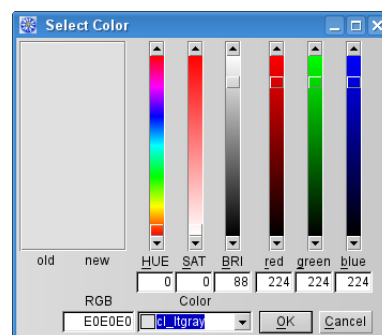


Bild 36: Farben-Editor.

len. Du erinnerst dich, ein Pixel ist ein einzelner Punkt auf dem Bildschirm, der aus je einer roten, grünen und blauen Teilfläche zusammengesetzt ist. Für die Speicherung der drei Helligkeitswerte benötigt der Computer drei Bytes, wir haben also den Wertebereich von 0 bis 255 zur Verfügung. **red**, **green** und **blue** = 255 ergibt weiss, **red**, **green** und **blue** = 0 ergibt schwarz. Eine andere Möglichkeit ist die Verwendung von **HUE**, **SAT** und **BRI**. **HUE** ist der Winkel im Farbkreis (0..360), **SAT** die Farbsättigung (0..100) und **BRI** die Helligkeit (0..100). Der Farben-Editor rechnet die Werte auf die vom Computer benötigten **red**, **green** und **blue** Werte um. Wir wählen **HUE** = 44, **SAT**

= 25 und BRI = 88 oder eine andere Farbe, die dir gefällt. Drücke OK, der Farbverlauf sieht nun aus wie in **Bild 37**.

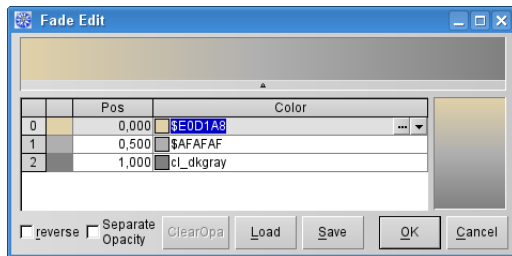


Bild 37: Farbverlauf.

Die Zeile 1 benötigen wir nicht, aktiviere sie und drücke **Ctrl+Delete**. Nun haben wir nur noch die Zeilen 0 und 1, setze die Farbe in der Zeile 1 mit dem Farben-Editor auf **HUE = 34, SAT = 72** und **BRI** auf 83 schliesse Farb- und Verlaufs-Editor mit OK, das Formular sieht aus wie in **Bild 38**.

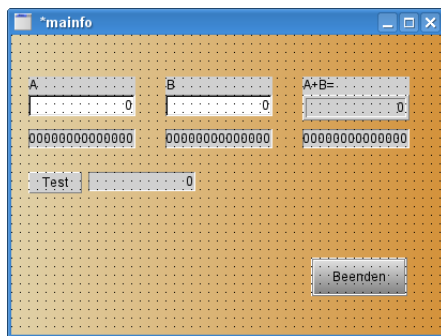


Bild 38: Formular mit Farbverlauf.

Aktiviere alle Komponenten ausser **beenden**, setze die **color**-Eigenschaft auf **cl_transparent**, die Komponenten werden durchsichtig (**Bild 39**).

Drücke **Esc** um die Aktivierung aufzuheben, es wird die übergeordnete Komponente (das Formular) aktiviert. Machen wir die Schrift grösser. Viele Komponenten haben die Eigenschaft **font** (Schriftart), welche normalerweise nicht aktiviert ist. Nicht aktivierte **font**-Eigenschaften verwen-

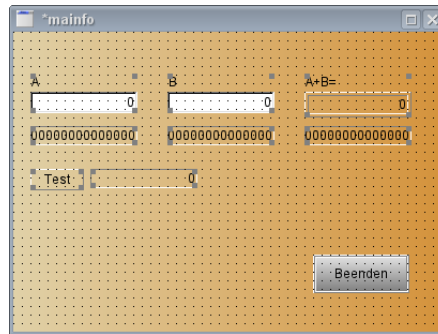


Bild 39: Transparente Komponenten.

den die Schriftart des übergeordneten Widgets das die Komponente enthält. Änderungen von **font** des Formulars wirken sich auf alle Komponenten aus welche keine eigene aktivierte **font**-Eigenschaft haben. Nicht aktivierte **font** werden im Objektinspektor durch **<parent>** (Vorgänger) angezeigt und können wie **frame** und **face** aktiviert werden.

Aktiviere und öffne **font** des Formulars, ändere **height** auf 18. Dies ist die Schriftgröße in Pixel. Die Schrift in den Binäranzeigen ist nun zu gross, wir sehen zu wenige Stellen, darum stellen wir für sie wieder die normale Schriftart ein. Aktiviere **bina**, **binb**, **binergebnis**, **testergebnis** und den **test**-Button. Aktiviere und öffne **font**, ändere **height** auf 0. **height = 0** bedeutet normale Schriftgröße.

Aktiviere **ergebnis**, betätige die **PfeilAuf**-Taste mit gedrückter **Ctrl**-Taste bis Ergebnis in der gleichen Flucht wie die beiden Eingabefelder steht. Die Pfeiltasten kombiniert mit **Ctrl** verschieben Komponenten pixelweise, kombiniert mit der **Shift**-Taste wird die Grösse pixelweise geändert.

Die Beschriftungen sollten zentriert über den Widgets stehen. Aktiviere **a**, **b**, und **ergebnis**, öffne **frame**, ändere **frame.captionpos** auf **cp_top**.

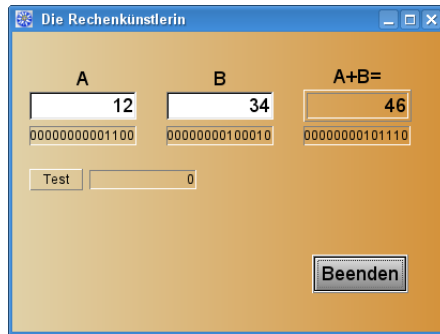


Bild 40: Das Ergebnis.

- Datentypen welche nur positive Zahlen darstellen können werden Cardinal-Zahlen genannt.
- Datentypen die auch negative Zahlen darstellen können werden Integer-Zahlen genannt.
- Für Integer-Zahlen wird das Zweierkomplement-System verwendet.
- Für Cardinal-Zahlen werden meist die 32 Bit grossen `card32` oder `cardinal` verwendet.
- Für Integer-Zahlen werden meist die 32 Bit grossen `int32` oder `integer` verwendet.
- Ctrl+Pfeil verschiebt aktive Komponenten.
- Shift+Pfeil verändert die grösse aktiver Komponenten.

2. Der Logikakrobat

2.1. Die Frage

Bist du logisch veranlagt? Schauen wir mal. Im Kapitel 1 haben wir `application.terminated` auf `false` gesetzt, um das Programm zu beenden. Daten welche nur zwei Zustände haben werden `boolean` genannt. Häufig müssen wir auf Grund von Aussagen oder Tatsachen Entscheidungen treffen; auch in Computerprogrammen ist das notwendig. In unseren Programmen können wir `boolean` Werte verwenden, um Ja/Nein- Zustände abzubilden.

Die einzelnen Eingangsinformations-`boolean` werden mit verschiedenen Operationen verknüpft, um zu einer Entscheidung zu gelangen. Nehmen wir an, das Programm einer Liftsteuerung muss entscheiden, ob die Lifttüre geöffnet werden soll oder nicht. Ein Teil der Entscheidungsfindung ist, dass die Türe nur dann geöffnet werden darf, wenn der Lift steht und eine Öffnungsanforderung vorliegt. Wir bilden eine Tabelle mit allen möglichen Eingangskombinationen und dem gewünschten Ergebnis. Mit zwei Eingangs-`boolean` haben wir $2^2 = 4$ verschiedene Möglichkeiten, das funktioniert wie bei den Bits im vorhergehenden Kapitel. Schreiben wir für `false` F und für `true` T.

Anforderung	Lift steht	Tür auf
F	F	F
F	T	F
T	F	F
T	T	T

Die `and` (und) Verknüpfung hat diese Eigenschaft, das Resultat ist nur dann `true`, wenn beide Eingangswerte `true` sind. Im Steuerungsprogramm könnte beispielsweise stehen:

```
tuerauf:= anforderung and liftsteht;
```

Die Information `liftsteht` haben wir nicht zur Verfügung, wir haben `motorlaeuft` und wissen, dass der Lift steht, wenn der Motor nicht läuft.

Motor läuft	Lift steht
F	T
T	F

`not` (nicht) macht diese Umkehrung. Die Anweisungen lauten:

```
liftsteht:= not motorlaeuft;
```

```
tuerauf:= anforderung and liftsteht;
```

oder direkt:

```
tuerauf:= anforderung and not motorlaeuft;
```

Das Licht in der Kabine muss leuchten, wenn die Türe geöffnet ist oder wenn die Kabine in Bewegung ist. So ganz perfekt ist diese Regel nicht, sei's drum!

Motor läuft	Türe auf	Licht an
F	F	F
F	T	T
T	F	T
T	T	T

Die entsprechende Verknüpfung heisst `or`.

```
lichtan:= motorlaeuft or tuerauf;
```

Mit not, and und or können wir alle logischen Probleme lösen, gegebenenfalls müssen wir zusätzlich die Reihenfolge der Abarbeitung mit Klammern festlegen. Du glaubst es nicht? z.B. gibt es zur Bequemlichkeit eine weitere boolean Operation xor, welche true bei unterschiedlichen Eingangswerten liefert.

a	b	a xor b
F	F	F
F	T	T
T	F	T
T	T	F

a xor b können wir problemlos durch a and not b or not a and b ersetzen. Wir müssen noch wissen, dass not am stärksten und and stärker bindet als or, (a and (not b)) or ((not a) and b) bedeutet dasselbe wie a and not b or not a and b. Es gibt noch weitere Formen durch die xor ersetzt werden kann. Sogar and kann durch not und or ersetzt werden, ebenfalls or durch not und and, so dass man mit not und and oder not und or für alle Fälle gerüstet ist.

Lege ein neues default Projekt mit neuem Verzeichnis und Namen logikakrobat an. Setze fünf tbooleanedit (das kleine Quadrat mit Kreuz) vom Edit-Bereich der Komponenten-Palette in einer Reihe am linken Rand des Formulars untereinander. Name und frame.caption = a, b, c, d, e, ondatataentered = dataentered. Nochmal fünf tbooleanedit rechts daneben, Name = antw1, antw2, antw3 antw4, antw5. Darunter ein tbutton, name = pruefen, caption = Prüfen.

```
antw1.frame.caption =
a and b and c and d or e
antw2.frame.caption =
not a and not (b or c) and not (d and e)
antw3.frame.caption =
```

```
not (e or a) and (a and b and not (c or d))
antw4.frame.caption = a or b or c or d or e
antw5.frame.caption =
not (not a and not b and not c and not d and not e)
```

Aktiviere a, b, c, d und e, antw1 bis antw5.ondataentered = dataentered.

```
pruefen.ondataentered = pruefene-
xe(Bild 41).
```

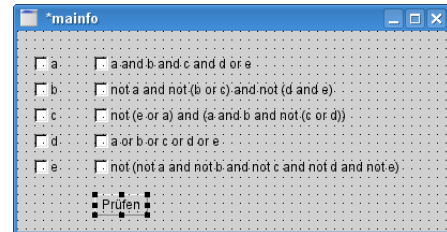


Bild 41: Komponenten Logikakrobat.

- Es gibt die boolean Operationen not, and, or und xor.
- not bringt die Umkehrung.
- and ist true wenn beide Eingänge true sind.
- or ist true wenn mindestens ein Eingang true ist.
- xor ist true, wenn die Eingänge unterschiedliche Werte haben.
- not bindet am stärksten.
- and bindet stärker als or und xor.

2.2. Die Antwort?

Wir wollen einen Logiktester programmieren. Auf drücken des Prüfen-Buttons hin sollen die eingegebenen Werte in antw1 bis antw5 mit den aus den Werten a bis e errechneten Werten verglichen werden. Bei

Übereinstimmung wird das Ergebnis grün angezeigt, bei Fehler rot. Durch Verändern der Werte a bis e sollen die Farben zurückgestellt werden. Beginnen wir damit, es ist lediglich eine Fleissarbeit. Ergänze die `dataentered` Routine wie **Listing 6**.

`cl_default` ist eine dem Compiler bekannte Farbkonstante, wir werden auf Programmkonstanten zurückkommen. `cl_default` meint "nimm das übliche".

In testexe berechnen wir zuerst die richtigen Ergebnisse und speichern sie in Zwischenvariablen (**Listing 7**). Vertippe dich nicht, sonst wirst du mit den Prüflingen Probleme bekommen. ;-)

Dann vergleichen wir die berechneten Werte mit den Eingegebenen und schalten die Farbe der Antworten entweder auf grün oder rot:

```
if antw1.value = erg1 then begin
  antw1.color := cl_green;
end
else begin
  antw1.color := cl_red;
end;
```

if erwartet einen boolean-Wert, ist er `true`, wird der `then`-Bereich ausgeführt, für `false` wird der `else`-Teil ausgeführt. `=` ist eine Vergleichsoperation, sie liefert `true`, wenn die Werte beider Seiten übereinstimmen. Der `else`-Teil kann auch wegfallen.

```
if antw1.value = erg1 then begin
  antw1.color := cl_green;
end;
```

würde auf grün für richtige Antworten schalten und die Farbe bei falscher Antwort nicht verändern, der `else`-Teil ist daher hier notwendig. Eine Alternative ist

```
antw1.color := cl_red;
if antw1.value = erg1 then begin
  antw1.color := cl_green;
```

```
end;
```

Setze `caption` des Formulars auf `Logik-Akrobat?` und starte das Programm .

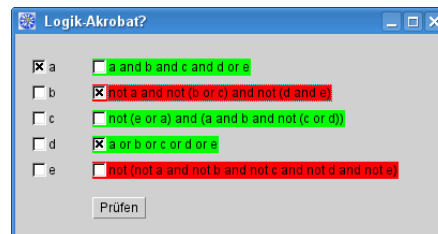


Bild 42: Logiktester.

Leere Quadrate bedeuten `false` Kreuze stehen für `true`. Funktioniert das Programm richtig?

- `=` ist eine Vergleichsoperation und bringt `boolean true` für übereinstimmende Werte.
- In `if then else` wird für `true` der `if`-Teil ausgeführt, für `false` der `else`-Teil.

2.3. Optimierungen

Platziere zwei `tintegerdisp` auf dem Formular, `Name` und `frame.caption = richtig, falsch`.

Die `if then else` Auswertung zur Farbeinstellung ist mehrmals vorhanden. Falls wir einen falsch/richtig-Zähler einbauen wollen, müssen wir die Zählerfunktion für jede Frage einzeln programmieren. Das ist langweilig und falls wir bei der Zählermethode einen Fehler machen, müssen wir ihn auch für jede Frage einzeln korrigieren. Die Chance ist gross, dass wir vergessen, einen Fehler an allen notwendigen Stellen des Programmes zu korrigieren. Programme sollten daher so konstruiert werden, dass Fehler nur an einem Ort behoben

```

procedure tmainfo.dataentered(const sender: TObject);
begin
  antw1.color:= cl_default; //farben zuruecksetzen
  antw2.color:= cl_default;
  antw3.color:= cl_default;
  antw4.color:= cl_default;
  antw5.color:= cl_default;
end;

```

Listing 6: Farben rücksetzen.

```

procedure tmainfo.pruefenexe(const sender: TObject);
var
  erg1,erg2,erg3,erg4,erg5: boolean;
begin
  //ergebnisse berechnen
  erg1:= a.value and b.value and c.value and d.value or e.value;
  erg2:= not a.value and not (b.value or c.value) and not (d.value and e.value);
  erg3:= not (e.value or a.value) and (a.value and b.value and
    not (c.value or d.value));
  erg4:= a.value or b.value or c.value or d.value or e.value;
  erg5:= not (not a.value and not b.value and not c.value and
    not d.value and not e.value);

  if antw1.value = erg1 then begin //antworten ueberpruefen
    antw1.color:= cl_green; //richtig
  end
  else begin
    antw1.color:= cl_red; //falsch
  end;
  if antw2.value = erg2 then begin
    antw2.color:= cl_green;
  end
  else begin
    antw2.color:= cl_red;
  end;
  if antw3.value = erg3 then begin
    antw3.color:= cl_green;
  end
  else begin
    antw3.color:= cl_red;
  end;
  if antw4.value = erg4 then begin
    antw4.color:= cl_green;
  end
  else begin
    antw4.color:= cl_red;
  end;
  if antw5.value = erg5 then begin
    antw5.color:= cl_green;
  end
  else begin
    antw5.color:= cl_red;
  end;
end;

```

Listing 7: Richtige Werte berechnen und Antworten überprüfen.

werden müssen.

Die `procedure`-Anweisung ist eine Möglichkeit dazu, diesem Ziel näher zu kommen (**Listing 8**).

`procedure` hat den Aufbau:

```
procedure procedurname(
    parametername1: typ1;
    parametername2: typ2);
begin
    anweisung;
    ...
end;
```

Die Parameterdefinitionen wirken wie Variablendefinitionen deren Werte beim Aufruf der Prozedur gesetzt werden. Die Anzahl der Parameter ist variabel, sie können mitsamt den Klammern auch ganz entfallen.

Auch `pruefenexe` ist eine `procedure`. Sie gehört zum Formular, daher der Name `tmainfo.pruefenexe`.

Wir erweitern die `pruefenexe` Prozedur um einen Falsch- und einen Richtig-Zähler (**Listing 9**). Die `var`-Definition der Zählervariablen muss vor der Definition von `vergleicheantwort` stehen, sonst reklamiert der Compiler. Grundsätzlich müssen alle Programmelemente vor der Verwendung definiert werden. Zur Erinnerung, Variablen besitzen am Anfang zufällige Werte und müssen initialisiert werden, darum

```
richtigzahl:= 0; //initialisieren
falschzahl:= 0;
```

vor dem Aufruf von `vergleicheantwort`. `richtigzahl` und `falschzahl` können nicht in `vergleicheantwort` definiert werden, da deren Inhalt dann zwischen den Aufrufen von `vergleicheantwort` verloren ginge.

Teste das Programm(**Bild 43**)!

Wir können weiter vereinfachen. Da die Gesamtzahl der Fragen bekannt ist, müssen wir nur die richtigen Antworten zählen, die

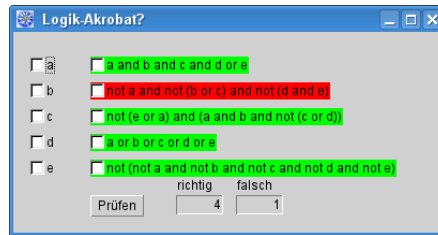


Bild 43: Falsch/Richtig Anzeige.

Anzahl der falschen Antworten kann berechnet werden. Damit wir die Fragenzahl einfach verändern können, auch wenn die Zahl mehrfach im Programm vorkommen sollte, definieren wir eine Konstante statt die Zahl direkt zu verwenden (**Listing 10**).

```
const
    fragenzahl = 5;
...
falsch.value:= fragenzahl - richtigzahl;
```

Auch hier gilt das Prinzip, Programme so zu schreiben, dass Fehler nur einmal korrigiert werden müssen und sich vorgenommene Änderungen im Programm an allen entsprechenden Orten auswirken.

```

procedure tmainfo.pruefenexe(const sender: TObject);

procedure vergleicheantwort(soll: boolean; ist: tbooleanedit);
begin
  if ist.value = soll then begin //antworten ueberpruefen
    ist.color:= cl_green;        //richtig
  end
  else begin
    ist.color:= cl_red;          //falsch
  end;
end;

var
  erg1,erg2,erg3,erg4,erg5: boolean;
begin
  //ergebnisse berechnen
  erg1:= a.value and b.value and c.value and d.value or e.value;
  erg2:= not a.value and not (b.value or c.value) and not (d.value and e.value);
  erg3:= not (e.value or a.value) and (a.value and b.value and
                                             not (c.value or d.value));
  erg4:= a.value or b.value or c.value or d.value or e.value;
  erg5:= not (not a.value and not b.value and not c.value and
                                             not d.value and not e.value);

  vergleicheantwort(erg1,antw1);
  vergleicheantwort(erg2,antw2);
  vergleicheantwort(erg3,antw3);
  vergleicheantwort(erg4,antw4);
  vergleicheantwort(erg5,antw5);
end;

```

Listing 8: Vergleichs- `procedure` .

```

procedure tmainfo.pruefenexe(const sender: TObject);
var
    richtigzahl: integer;
    falschzahl: integer;

procedure vergleicheantwort(soll: boolean; ist: tbooleanedit);
begin
    if ist.value = soll then begin //antworten ueberpruefen
        ist.color:= cl_green;      //richtig
        richtigzahl:= richtigzahl + 1;
    end
    else begin
        ist.color:= cl_red;        //falsch
        falschzahl:= falschzahl + 1;
    end;
end;

var
    erg1,erg2,erg3,erg4,erg5: boolean;
begin
    //ergebnisse berechnen
    erg1:= a.value and b.value and c.value and d.value or e.value;
    erg2:= not a.value and not (b.value or c.value) and not (d.value and e.value);
    erg3:= not (e.value or a.value) and (a.value and b.value and
                                                not (c.value or d.value));
    erg4:= a.value or b.value or c.value or d.value or e.value;
    erg5:= not (not a.value and not b.value and not c.value and
                not d.value and not e.value);

    richtigzahl:= 0; //initialisieren
    falschzahl:= 0;
    vergleicheantwort(erg1,antw1);
    vergleicheantwort(erg2,antw2);
    vergleicheantwort(erg3,antw3);
    vergleicheantwort(erg4,antw4);
    vergleicheantwort(erg5,antw5);
    richtig.value:= richtigzahl; //ergebnisse anzeigen
    falsch.value:= falschzahl;
end;

```

Listing 9: Falsch/Richtig-Zähler.

```

procedure tmainfo.pruefenexe(const sender: TObject);
var
    richtigzahl: integer;

procedure vergleicheantwort(soll: boolean; ist: tbooleanedit);
begin
    if ist.value = soll then begin //antworten ueberpruefen
        ist.color:= cl_green; //richtig
        richtigzahl:= richtigzahl + 1;
    end
    else begin
        ist.color:= cl_red; //falsch
    end;
end;

const
    fragenzahl = 5;
var
    erg1,erg2,erg3,erg4,erg5: boolean;
begin
        //ergebnisse berechnen
        erg1:= a.value and b.value and c.value and d.value or e.value;
        erg2:= not a.value and not (b.value or c.value) and not (d.value and e.value);
        erg3:= not (e.value or a.value) and (a.value and b.value and
            not (c.value or d.value));
        erg4:= a.value or b.value or c.value or d.value or e.value;
        erg5:= not (not a.value and not b.value and not c.value and
            not d.value and not e.value);

        richtigzahl:= 0; //initialisieren
        vergleicheantwort(erg1,antw1);
        vergleicheantwort(erg2,antw2);
        vergleicheantwort(erg3,antw3);
        vergleicheantwort(erg4,antw4);
        vergleicheantwort(erg5,antw5);
        richtig.value:= richtigzahl; //ergebnisse anzeigen
        falsch.value:= fragenzahl - richtigzahl;
end;

```

Listing 10: Konstante für Fragenzahl.

- Variablen und allgemein alle Programmelemente müssen definiert werden bevor sie verwendet werden können.
- Variablen müssen vor Verwendung initialisiert werden.
- Programme sollten so geschrieben werden, dass Fehler nur einmal korrigiert werden müssen.
- Prozedurparameter wirken wie Variablendefinitionen und werden mit den beim Prozeduraufruf angegebenen Werten initialisiert.
- **const** leitet einen Konstanten-Definitionsbereich ein.

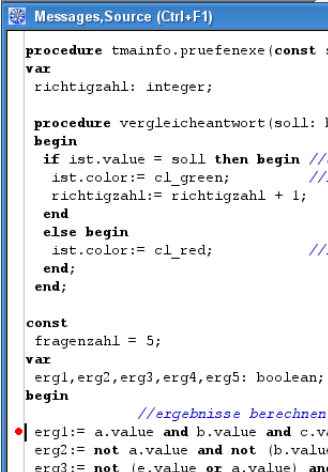
2.4. Es funktioniert nicht!

Programme leisten selten auf Anhieb was sie sollten. Es soll sogar Programme geben, welche überhaupt nie perfekt funktionieren. ;-)

MSEide stellt viele Werkzeuge zur Verfügung welche die Fehlersuche in Programmen vereinfachen und in verzweifelten Situationen erst ermöglichen. Ein wichtiges Hilfsmittel sind 'Breakpoints' (Haltepunkte). Falls wir beim laufenden Logikakroben auf Prüfen klicken, nehmen wir an, dass `pruefenexe` aufgerufen wird. Aber ist das auch wirklich so?

Klicke mit der linken Maustaste im Programmfenster am linken Rand in die Zeile `erg1:=` unter `//ergebnisse berechnen`. Es wird ein Haltepunkt gesetzt, angezeigt durch einen roten Punkt (Bild 44).

Damit Breakpoints wirksam sind, muss in der Kommandozone das Feld mit dem roten Punkt aktiviert ("eingedrückt") sein, bitte kontrolliere es. Drücke F9 um das Pro-



```

procedure tmainfo.pruefenexe(const s
var
richtigzahl: integer;

procedure vergleicheantwort(soll: b
begin
if ist.value = soll then begin //a
ist.color:= cl_green; //r
richtigzahl:= richtigzahl + 1;
end
else begin
ist.color:= cl_red; //f
end;
end;

const
fragenzahl = 5;
var
erg1,erg2,erg3,erg4,erg5: boolean;
begin
//ergebnisse berechnen
erg1:= a.value and b.value and c.value
erg2:= not a.value and not (b.value
erg3:= not (e.value or a.value) and

```

Bild 44: Haltepunkt.

gramm zu starten. Klicke im Logikakroben auf Prüfen.

Das Programm wird am Breakpoint angehalten (Bild 45).

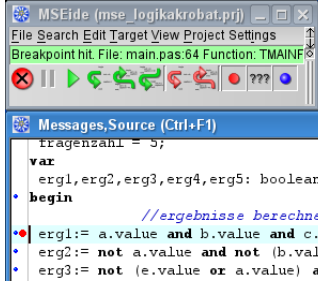


Bild 45: Gestoppt am Haltepunkt.

Wenn wir mit der Maus auf `erg1` fahren, wird der Wert der Variable angezeigt (Bild 46).

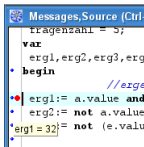


Bild 46: Wertanzeige.

32? Dort sollte doch true oder false stehen? Na klar, Variablen enthalten anfäng-

lich zufällige Werte. Drücke F8, das Programm führt einen Schritt aus (Bild 47). Schon besser, `erg1` zeigt jetzt false.

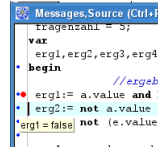


Bild 47: Ein Programm-Schritt.

Drücke solange F8, bis das Programm bei `vergleicheantwort(erg2, antw2)` ankommt. Wenn wir in eine aufgerufenen Prozedur hinein springen wollen, müssen wir F7 statt F8 drücken. Drücke F7. Das Programm stoppt in der Prozedur `vergleicheantwort` (Bild 48).

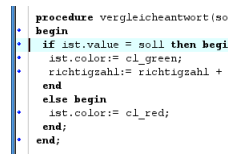


Bild 48: In die Prozedur hinein mit F7.

Drücke solange F8 bis das Programm die Prozedur verlässt. Drücke F7, `vergleicheantwort` wird nun mit `erg3` und `antw3` aufgerufen. Falls uns der weitere Verlauf in `vergleicheantwort` nicht mehr interessiert, können wir Shift+F7 drücken, Das Programm wird dann bei der Rückkehr zur aufrufenden Stelle gestoppt. Mit F9 wird ein gestopptes Programm wieder fortgeführt. Ein Breakpoint wird durch nochmaliges Klicken vorübergehend deaktiviert, (Bild 49) ein Doppelklick entfernt den Breakpoint.

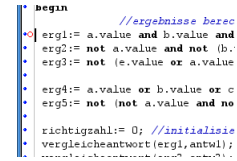


Bild 49: Deaktivierter Haltepunkt.

- F8 führt einen Programmschritt aus und überspringt dabei Prozeduraufrufe.
- F7 führt einen Programmschritt aus und springt in aufgerufene Prozeduren hinein.
- Shift+F7 Verlässt die aktuelle Prozedur.
- F9 lässt ein gestopptes Programm weiterlaufen.

3. Der PanoRamatograph, wir simulieren

3.1. Die Idee

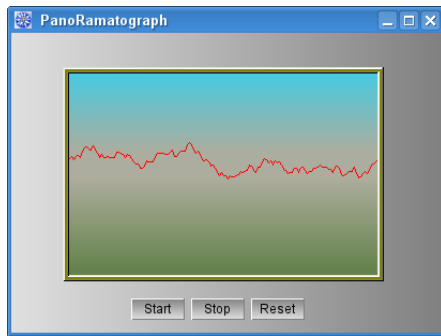


Bild 50: PanoRamatograph

Der PanoRamatograph ist ein Programm zur automatischen Erstellung von Panoramabildern. Die Idee: Wir simulieren die Erosion um eine Horizontlinie zu bilden. Dazu teilen wir den Horizont in gleichmässig verteilte Höhenpunkte ein, welche wir mit Linien verbinden. In regelmässigen Zeitabständen verändern wir die Panoramahöhen um einen für alle Punkte unterschiedlichen Erosionswert, so können wir dem Wachsen von Gebirge und Tälern zusehen.

Das Abtragen des Gebirges erledigen wir in einer Prozedur. Diese Prozedur muss in regelmässigen Abständen aufgerufen werden. Die Komponente `ttimer`, der Wecker der Abteilung `NoGui` der Komponentenpalette kann das. Platziere ein `ttimer` auf dem Formular, `Name = timer`, `onexecute = timerexe`. `timerexe` wird im Rhythmus `interval` aufgerufen. `interval` ist in Mikrosekunden, wir wollen das Gebirge zehn mal pro Sekunde abtragen, darum setzen wir `interval` auf

100000. Platziere drei `tbutton` am unteren Rand des Formulars, `Name = start`, `stop`, `reset`, `caption = Start`, `Stop`, `Reset`, `onexecute = startexe`, `stopexe`, `resetexe` (Bild 51).

In der `startexe` Prozedur starten wir den Timer, in der `stopexe` Prozedur stoppen wir ihn. Zum Testen schreiben wir in `timerexe`:

`guibeeep()`; (Listing 11)

Teste das Programm, nach Klicken von **Start** sollte etwas im Lautsprecher zu hören sein und nach Klicken von **Stop** wieder verstummen. Wenn es funktioniert können wir `guibeeep()`; wieder entfernen.



Bild 51: Timer Test.

Für jeden Panoramapunkt benötigen wir einen entsprechenden Erosionswert. Die Erosionswerte sind Gleitkommazahlen. Wie Integer- und Cardinal-Zahlen gibt es auch Gleitkommazahlen in verschiedenen Bitgrössen.

```

procedure tmainfo.timerexe(const sender: TObject);
begin
    guibeep(); //erzeuge signal im lautsprecher
end;

procedure tmainfo.startexe(const sender: TObject);
begin
    timer.enabled:= true;
end;

procedure tmainfo.stopexe(const sender: TObject);
begin
    timer.enabled:= false;
end;

procedure tmainfo.resetexe(const sender: TObject);
begin
end;

```

Listing 11: Timer Test.

Bits	Bereich	Gültige Stellen	Name
32	$1.5 * 10^{-45} .. 3.4 * 10^{38}$	7-8	flo32
64	$5.0 * 10^{-324} .. 1.7 * 10^{308}$	15-16	flo64, float

Normalerweise verwenden wir flo64 (float). Wer mehr über den Aufbau von Float-Datentypen erfahren will, kann auf Wikipedia unter IEEE 754 nachlesen.

Auch für die Reihe der Horizontpunkte können wir einen Dataentypen definieren:

```

array [ 0 ..
    panoramabreite-1 ] of float ;

```

array ist eine Aneinanderreihung von Elementen des gleichen Datentyps, wobei die einzelnen Elemente über den angegebenen Index-Bereich angesprochen werden können (Index = "Elementnummer"). Meist besitzt das erste Element die Nummer 0, das Letzte bekommt dann die Nummer um eins kleiner als die Anzahl der Elemente. Den Panoramapunkte-array-Typen benötigen wir mehrmals, darum definieren wir ihn symbolisch einem

Typen-Definitions-Bereich. Wir können die Panoramavektor-Variablen nicht in der timerexe Prozedur definieren, da ihr Inhalt sonst zwischen den Aufrufen verloren ginge. Zudem benötigen wir den Zugriff darauf auch in anderen Programmteilen. Darum definieren wir hoehe und erosion vor den Prozeduren (**Listing 12**).

Variablen welche nicht zu einer Prozedur gehören werden globale Variablen genannt. Globale Variablen sollten sparsam eingesetzt werden, da von überall her darauf zugegriffen werden kann. Es kommt vor, dass globale Variablen aus Versehen verändert oder verwechselt werden. Zudem belegen globale Variablen während der gesamten Programmlaufzeit Speicherplatz; meistens sind lokale Prozedur-Variablen besser geeignet.

div ist die Divisionsoperation für ganze Zahlen, **/** funktioniert nur mit Gleitkommazahlen.

```

implementation
uses
  main_mfm;

const
  panoramaschritt = 2; //pixel pro panoramapunkt
  panoramabreite = 300 div panoramaschritt; //anzahl panoramapunkte
  panoramahoehe = 200; //pixel
  erosionsgeschwindigkeit = 5; //pixel pro schritt

type
  panoramaarrayty = array[0..panoramabreite - 1] of float;

var
  hoehe: panoramaarrayty;
  erosion: panoramaarrayty;

```

Listing 12: Array Typen-Definition.

- **div** ist die Divisionsoperation für ganze Zahlen.
- **type** leitet einen Typen-Definitions-Bereich ein.
- **array [Startindex .. Endindex] of Datentyp ;** bezeichnet eine Aneinanderreihung von Elementen des gleichen Datentyps.

3.2. Die Ausführung

Zunächst benötigen wir ein Element um das Panoramabild anzuzeigen. Platziere eine `tpaintbox` - der Pinsel aus der Abteilung `Widget` - auf dem Formular, `name = panorama`.

Dann müssen wir `hoehe` auf die Ursprungswerte (alles 0.0) initialisieren. Die Initialisierung soll beim Programmstart und beim Klicken von `Reset` durchgeführt werden. Machen wir eine Prozedur `rueckstellen()`. Wir könnten schreiben:

```

hoehe[0] := 0.0;
hoehe[1] := 0.0;
hoehe[2] := 0.0;
...

```

Zum Glück geht das bequemer:

```

for i1 := 0 to high(hoehe) do begin
  hoehe[i1] := 0.0;
end;

```

`i1` ist eine zu definierende Integer Variable, `high(hoehe)` bringt den Index des letzten Elementes von `hoehe`. Ansonsten ist ja wohl klar, was die `for`-Anweisung macht.

Die `rueckstellen` Prozedur sieht aus wie Listing 13. `mainfo.panorama.invalidate()`; sagt dem Panoramabild, dass es verändert wurde und sich neu zeichnen muss. `rueckstellen()` ist kein Bestandteil vom Formular, darum müssen wir dem Compiler sagen, dass wir `panorama` von `mainfo` meinen. `mainfo` ist die sogenannte Instanzvariable des Formulars worüber das Programm darauf zugreifen kann.

Rufe `rueckstellen()` aus `resetexe()` heraus aus auf. Um `rueckstellen()` beim Programmstart aufzurufen, verwenden wir das Ereignis `oncreate` des Formulars. `oncreate` wird einmalig beim Erstellen des Formulars aufgerufen. Wähle in `oncreate` ebenfalls `resetexe` (Listing 14).

In `startexe()` müssen die Erosionswer-

```

procedure tmainfo.resetexe(const sender: TObject);
begin
  rueckstellen();
end;

```

Listing 14: Initialisierung.

```

procedure rueckstellen();
var
  i1: integer;
begin
  for i1:= 0 to high(hoehe) do begin
    hoehe[i1]:= 0;
  end;
  mainfo.panorama.invalidate();
end;

```

Listing 13: Panoramahöhen rückstellen.

te erzeugt werden. Zuerst füllen wir ein Array mit Zufallszahlen im Bereich -1.0 bis +1.0. Da es unwahrscheinlich ist, dass sich die Erosionswerte in benachbarten Punkten stark unterscheiden, legen wir für jeden Punkt in `erosion` den Durchschnitt einiger benachbarten Zufallszahlen ab (**Listing 15**).

Der Fachausdruck dafür heisst “moving average”.

Wird `durchschnittzahl` grösser als `panoramabreite` gewählt kann das Programm nicht richtig funktionieren. In diesem Fall muss es durch `halt()` unverzüglich abgebrochen werden, dabei kann eine Fehlernummer angegeben werden. `random()` liefert eine Zufalls-Gleitkommazahl im Bereich von 0.0 bis 1.0. `mod` ist die modulo Operation für Ganzzahlen. `a mod b` bedeutet “schneide von `a` `b`-lange Stücke ab und gib mir den Abschnitt”; `mod` liefert den Rest der Division von `a` durch `b`.

Nun können wir in `timerexe()` die Höhenpunkte der Erosion nachführen (**Listing 16**).

Bleibt noch die Anzeige der simulierten Werte. Im Programm haben wir die

Panoramahöhe mit 200 Pixeln und die Panoramabreite mit 300 Pixeln bestimmt. Die Abmessungen der Widgets wird in mehreren mit `bounds` beginnenden Eigenschaften festgelegt: `bounds_x`, `bounds_y`, `bounds_cx`, `bounds_cy`... `_x` und `_y` ist die Position der linken oberen Ecke in Pixel, `_cx` und `_cy` ist Breite und Höhe. Der Ursprung der Position ist die linke obere Ecke des übergeordneten Widget (das Widget, welches unser Widget enthält). Der Objektspektor trennt Eigenschaftsnamen zur besseren Übersicht bei `_` und stellt sie als Baum dar (**Bild 52**).

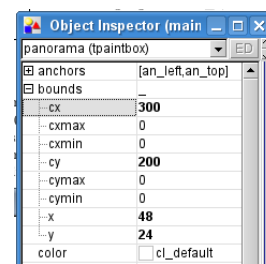


Bild 52: `bounds_` Eigenschaften.

Setze `bounds_cx` der `panorama` Komponente auf 300 und `bounds_cy` auf 200 um die passende Anzeigefläche zu erzeugen.

Zum Zeichnen verwenden wir das Ereignis `onpaint` der `panorama`-Komponente, nenne die Prozedur `paintexe` (**Listing 17**). Über `sender` können wir auf die Komponente zugreifen von der das Ereignis stammt, in unserem Fall ist es die `panorama`-Komponente. `acanvas` (`canvas` = Leinwand) ist ein Objekt vom Typ `tcanvas`, welches die zum Zeichnen benötigten Funktionen zur Verfügung stellt. Klicke links bei gedrück-

```

procedure tmainfo.startexe(const sender: TObject);
const
  durchschnittzahl = 128;
var
  i1: integer;
  durchschnitt: float;
  zufall: panoramaarrayty;
begin
  if durchschnittzahl > panoramabreite then begin
    halt(1); //fataler fehler, programm muss mit fehler nummer abgebrochen werden
  end;
  for i1:= 0 to high(zufall) do begin
    zufall[i1]:= erosionsgeschwindigkeit*(2*random() - 1);
    //zufallszahl im bereich -1..+1
  end;
  durchschnitt:= 0.0;
  for i1:= 0 to durchschnittzahl - 1 do begin //bilde startwert
    durchschnitt:= durchschnitt + zufall[i1];
  end;
  for i1:= 0 to high(erosion) do begin
    erosion[i1]:= durchschnitt/durchschnittzahl;
    //erhält den durschnittswert der folgenden werte
    durchschnitt:= durchschnitt - zufall[i1]; //letzten wert entfernen
    durchschnitt:= durchschnitt + zufall[(i1+durchschnittzahl) mod
    panoramabreite];
    //neuen wert hinzufügen, kreis schliessen
  end;
  timer.enabled:= true;
end;

```

Listing 15: Zufällige Erosionszahlen erzeugen.

```

procedure tmainfo.timerexe(const sender: TObject);
var
  i1: integer;
begin
  for i1:= 0 to high(hoehe) do begin
    hoehe[i1]:= hoehe[i1] + erosion[i1]; //wachstums und erosionswirkung
  end;
  panorama.invalidate; //panorama muss neue gezeichnet werden
end;

```

Listing 16: Die Erosion.

ter Ctrl-Taste auf tcanvas. MSEide zeigt die Definition von tcanvas, scrolle nach unten, hier gibt es einiges zu entdecken. Ein Klick auf den blauen Pfeil in der rechten oberen Fensterecke bringt dich wieder zurück an den Ursprungsort.

tcanvas hat die Funktion drawlines(), welche ein Array von Punkten mit Linien verbindet. Definiert ist sie in tcanvas als (etwas abgekürzt)

```
procedure drawlines(const apoints:
                        array of pointty).
```

Um die Position von Punkten anzugeben, benötigen wir die x- und die y-Koordinaten, das heisst, den waagrechten und senkrechten Abstand vom Zeichnungsursprung. pointty ist daher definiert als

```
type
  pointty = record
    x,y: integer;
  end;
```

record end definiert einen zusammengesetzten Datentypen. x und y sind die Koordinaten in Pixel, der Ursprung ist oben links, d.h. x nimmt von links nach rechts zu, y von oben nach unten. Zuerst berechnen wir die Koordinaten der Punkte der Panoramalinie und speichern sie im panoramapixel-Array. Dann stellen wir die Zeichnungsfarbe ein und übergeben das Pixel-Array an acanvas.drawline().

Die Vollständige Zeichenroutine ist in **Listing 18**. round() (runden) liefert den nächstliegenden Integer-Wert des übergebenen Float-Wertes. Teste das Programm aus!

- **for** Ganzzahlvariable := Startwert **to** Endwert **do** begin
end ;
erhöht Ganzzahlvariable der Reihe nach von Startwert bis zu Endwert um eins und führt für jeden Wert die Anweisungen zwischen begin und end aus.
- **for** Ganzzahlvariable := Startwert **downto** Endwert **do** begin
end ;
macht dasselbe, nur wird Ganzzahlvariable bei jedem Schritt um eins verkleinert.
- high(Arrayvariable) liefert den Index des letzten Elementes von Arrayvariable.
- low(Arrayvariable) liefert den Index des ersten Elementes von Arrayvariable.
- mod ist der modulo Operator und liefert den Rest der Division.
- record ... end definiert einen zusammengesetzten Datentyp.

3.3. Die Politur

Ein tolles Bild kommt nur mit einem entsprechenden Rahmen zur Geltung. Aktiviere panorama.frame. levelo = 2, leveli = -2, framewidth = 2, colorframe = cl_dkyellow, so entsteht ein wunderschöner Goldrahmen. Für den Bildhintergrund aktiviere panorama.face und gestalte face.fade geschmackvoll. ;-)

Zu guter Letzt Farbverläufe für die Buttons und den Formularhintergrund. Alle Vier bekommen den gleichen Farbver-

```

procedure tmaininfo.paintexe(const sender: twidget; const acanvas: tcanvas);
begin
end;

```

Listing 17: paintexe()

```

procedure tmaininfo.paintexe(const sender: twidget; const acanvas: tcanvas);
const
  verschiebung = panoramahoehe div 3 ;
var
  panoramapixel: array[0..panoramabreite-1] of pointty;
  i1: integer;
begin
  for i1:= 0 to high(panoramapixel) do begin
    panoramapixel[i1].x:= i1 * panoramaschritt;
    panoramapixel[i1].y:= verschiebung + round(hoehe[i1]);
  end;
  acanvas.color:= cl_red;
  acanvas.drawlines(panoramapixel);
end;

```

Listing 18: Zeichenroutine.

lauf, darum verwenden wir eine Vorlage. Aus der Abteilung Gui nimm tfaccomp, Name = facetemplate. Aktiviere container.face des Formulars, in container.face.template wähle facetemplate. Aktiviere face der Button, wähle ebenfalls facetemplate in face.template.

Die in facetemplate.template gemachten Einstellungen gelten nun für alle verknüpften face-Eigenschaften. Erzeuge einen Farbverlauf (**Bild 53**). Für die Button können wir die Richtung des Farbverlaufs zusätzlich auf gd_up stellen um den Effekt zu verbessern (**Bild 54**).

3.4. Die Optimierung

Selbstverständlich sind auch hier weitere Optimierungen möglich. z.B. erscheinen die kleinen spitzigen Zacken im Panorama als unnatürlich. Dies lässt sich durch eine weitere Glättung der Erosionswerte in der unmittelbaren Umgebung der Panorama-punkte verbessern. Im Prinzip können wir dieselbe Operation zweimal anwenden, ein-

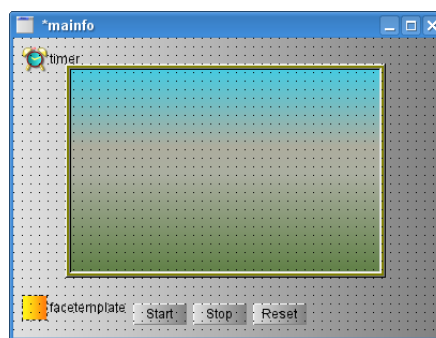


Bild 53: Farbverlauf über Vorlage.

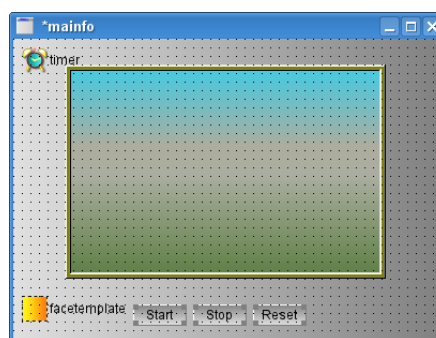


Bild 54: Spezielle Verlaufsrichtung für die Schaltflächen.

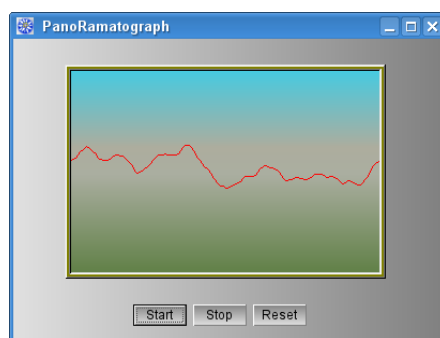
mal mit 128 Durchschnittswerten wie bereits gemacht und ein weiteres Mal mit z.B. 5 Durchschnittswerten für die Glättung der unmittelbaren Nachbarschaft. Darum programmieren wir eine Prozedur für den Vorgang, welche wir zwei mal mit verschiedenen Parametern aufrufen können (**Listing 19**).

Mit `const` vor `quelle` geben wir an, dass wir `quelle` nicht verändern werden. `out` vor `ziel` bedeutet, dass wir `ziel` zum schreiben verwenden wollen und nicht lesen werden. Durch die Angabe von `const` muss der Compiler beim Aufruf von `glaetten()` die Daten von `quelle` nicht kopieren, sondern kann direkt mit der Variable der aufrufenden Prozedur arbeiten. Es besteht ja keine Gefahr, dass die Daten verändert werden. Wir müssen uns allerdings an das Versprechen, die Daten nicht zu verändern, halten.

`out` bewirkt dass direkt mit der Variable der aufrufenden Prozedur gearbeitet wird, wir können somit Daten an die aufrufende Prozedur zurückliefern. `var` vor einer Parameterdefinition macht dasselbe wie `out`, nur teilen wir dem Compiler damit mit, dass wir die Daten der aufrufenden Prozedur auch zu lesen beabsichtigen.

Die `startexe()`-Prozedur vereinfacht sich nun zu **Listing 20**.

- `const`-Parameter dürfen nicht verändert werden.
- `out`-Parameter dienen zur Datenrückgabe.
- `var`-Parameter dienen ebenfalls zur Datenrückgabe und verwenden direkt die Variable der aufrufenden Prozedur.
- Parameter ohne `const`, `out` oder `var` entsprechen einer lokalen Variablendeklaration in der aufgerufenen Prozedur, worin die Daten der aufrufenden Prozedur kopiert werden.



Listing 21: Geglättetes Panorama.


```

procedure glaetten(durchschnittlaenge: integer;
                   const quelle: panoramavektorty; out ziel: panoramavektorty);
var
  i1: integer;
  durchschnitt: float;
begin
  if durchschnittlaenge > panoramabreite then begin
    halt(1); //fataler fehler, programm muss mit fehler nummer abgebrochen werden
  end;
  durchschnitt:= 0.0;
  for i1:= 0 to durchschnittlaenge - 1 do begin //bilde startwert
    durchschnitt:= durchschnitt + quelle[i1];
  end;
  for i1:= 0 to high(quelle) do begin
    ziel[i1]:= durchschnitt/durchschnittlaenge;
    //erhält den durchschnittswert der folgenden werte
    durchschnitt:= durchschnitt - quelle[i1]; //letzten wert entfernen
    durchschnitt:= durchschnitt + quelle[(i1+durchschnittlaenge) mod
                                         panoramabreite];
    //neuen wert hinzufügen, kreis schliessen
  end;
end;

```

Listing 19: Gätten Prozedur.

```

procedure tmainfo.startexe(const sender: TObject);

const
  durchschnittzahl = 128;
  glaettungszahl = 5;
var
  zufall: panoramavektorty;
  erosionroh: panoramavektorty;
  i1: integer;
begin
  for i1:= 0 to high(zufall) do begin
    zufall[i1]:= erosionsgeschwindigkeit*(2*random() - 1);
    //zufallszahl im bereich -1..+1
  end;
  glaetten(durchschnittzahl,zufall,erosionroh);
  glaetten(glaettungszahl,erosionroh,erosion);
  timer.enabled:= true;
end;

```

Listing 20: Doppelte Glättung der Erosionswerte.

4. Das Sprachgenie

4.1. Programmaufbau

Erstelle ein neues Projekt mit dem Namen sprachgenie (Project-New-From Template, default.prj usw.). Woraus besteht das Projekt?

Der Compiler baut das Programm aus verschiedenen Dateien zusammen. Die Hauptdatei ist sprachgenie.pas (Bild 55), welche vom Compiler zuerst ausgewertet wird. Der Text beginnt mit dem Schlüsselwort `program` und dem Programmnamen abgeschlossen mit `;`. Danach folgen einige Verwaltungsanweisungen an den Compiler als Kommentare in blau. Die Bedeutung dieser Anweisungen ist in der Compiler-Dokumentation beschrieben [2].

Dann folgt `uses` und die Aufzählung der Programmteile (`units`), die zum Programm gehören. `cthreads` ist ein internes Modul, welches nur für Linux benötigt wird. `msegui` ist das Modul worin `application` definiert ist. `main` schlussendlich ist unser Formular-Modul, worin wir die Programm-Anweisungen schreiben werden.

Setze einen Haltepunkt in Zeile 10 und starte das Programm, es wird am Breakpoint angehalten (Bild 56).

Drücke F8 (= Programmschritt ohne in Prozeduren hineinzuspringen), die `createform()` Prozedur der `application` Komponente erzeugt das Formularfenster, wobei lediglich der Rahmen von Linux oder Windows gezeichnet wird (Bild 57).

Drücke nochmals F8, nun läuft das Programm in der `application.run()` Prozedur (Bild 58).

Schliesse das Formularfenster des laufenden Programmes, `application.run()` wird verlassen (Bild 59).

Drücke nochmals F8, der Lebenszyklus des Programms ist beendet (Bild 60).

4.2. Unitaufbau

Ziel des Sprachgenie-Programmes ist das Ausfüllen eines Schecks, wo der Betrag sowohl als Ziffern als auch in Worten ausgeschrieben werden muss. Die Ziffern sollen durch Leerzeichen bei jeder dritten Stelle und die Worte durch “-” gegliedert werden. Zugegeben, allzu genial ist das nicht, aber du kannst ja die Funktion genial erweitern. ;-)

Um den gewünschten Betrag einzugeben, verwenden wir eine `tintegeredit`-Komponente - wir werden nur ganze Zahlen berücksichtigen. Setze ein `tintegeredit` auf das Formular. MSEide hat einen entsprechenden Eintrag in `tmaininfo` vorgenommen (Listing 22). `tintegeredit1` wurde als provisorischer Name eingesetzt.

```
type
  tmaininfo = class(tmainform)
    tintegeredit1: tintegeredit;
  end;
```

Listing 22: main.pas mit `tintegeredit1` Eintrag.

Im Objektinspektor ändern wir den Name-Wert auf `eingabe`, MSEide führt den Programmtext entsprechend nach (Listing 23).

Die Informationen über den Aufbau

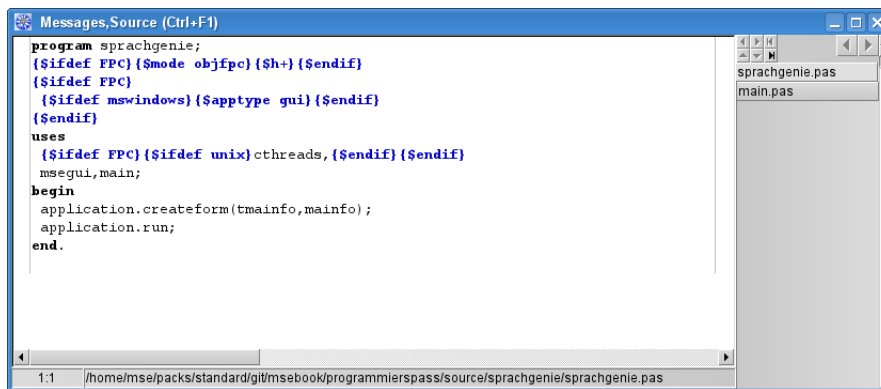


Bild 55: Programm-Hauptdatei.

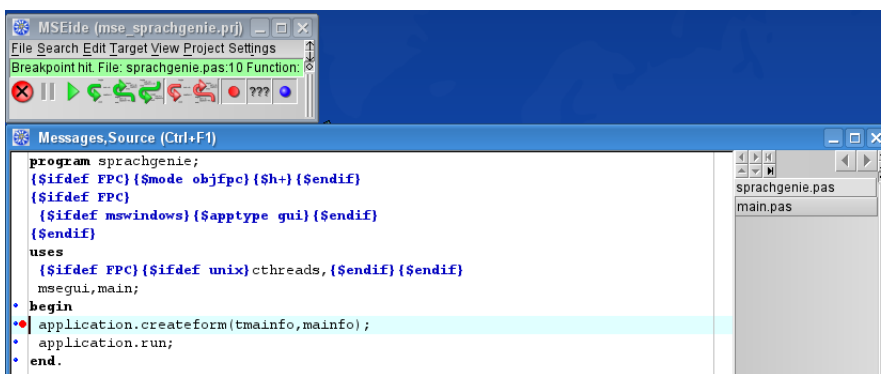


Bild 56: Programmstart.

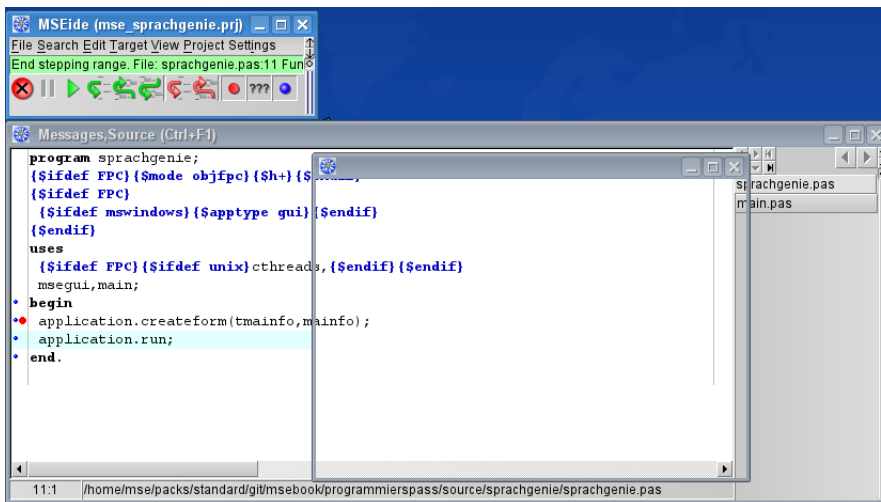


Bild 57: Formular ist erzeugt.

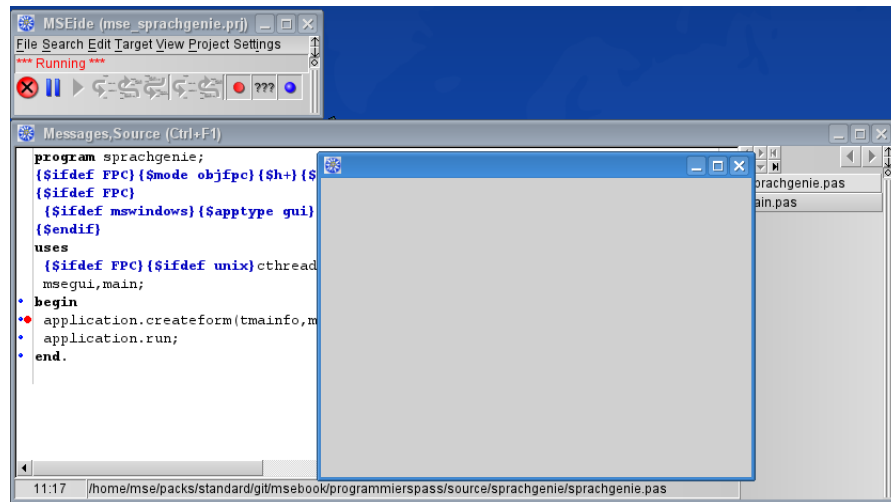


Bild 58: Programm läuft.

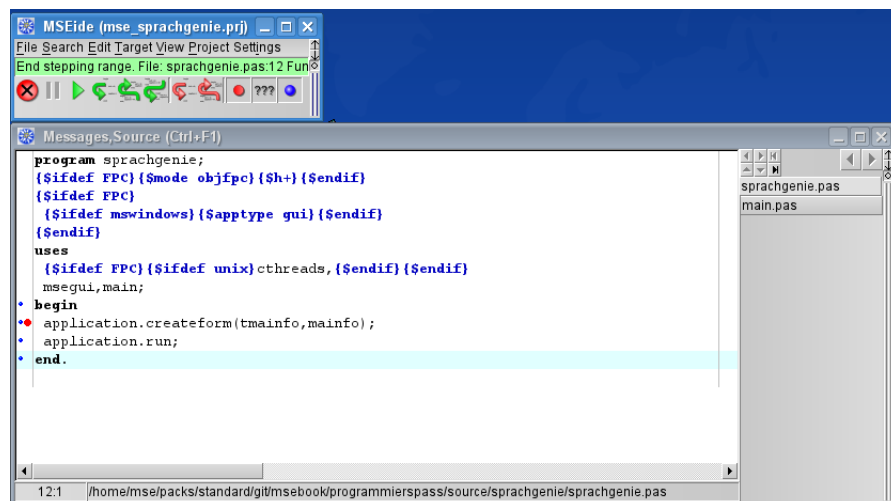


Bild 59: Formularfenster geschlossen.

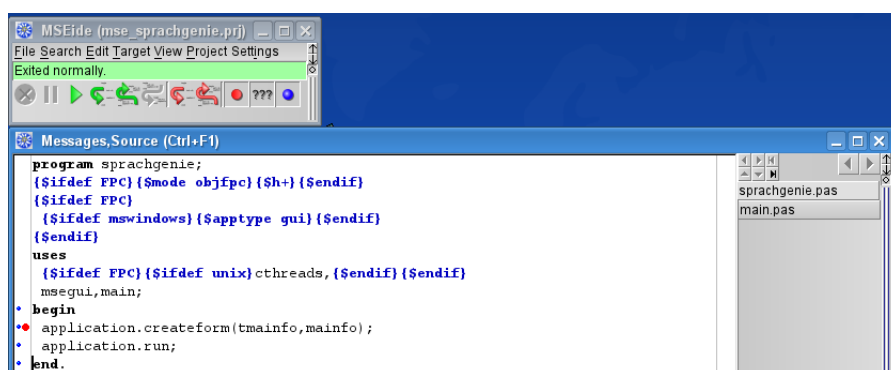


Bild 60: Programm beendet.

```

type
  tmainfo = class(tmainform)
    eingabe: tintegeredit;
  end;

```

Listing 23: main.pas mit eingabe Eintrag.

des main-Formulares werden in der Datei main.mfm festgehalten. Mache einen rechts-Klick auf der Formularfläche, wähle Show as text (**Bild 61**), das Formular wird geschlossen und als Text angezeigt (**Listing 24**).

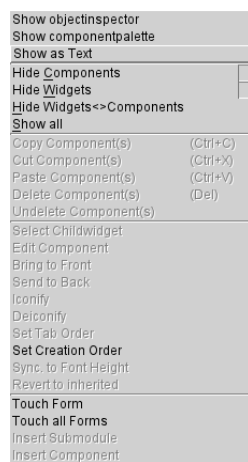


Bild 61: Auf Formular-Textanzeige umstellen.

Hier sind die eingesetzten Komponenten und ihre Eigenschaften aufgeführt. Die verwendete Sprache wird vom Compiler nicht verstanden und muss von MSEide vor dem Kompilieren umgewandelt werden. Bitte schliesse die main.mfm Datei durch drücken von Ctrl+F4.

Der vollständige Programmtext der Datei main.pas ist in **Listing 25** aufgeführt. Eine Unit beginnt mit dem Schlüsselwort **unit** gefolgt von Unitnamen und **;**. Danach wieder einige blaue Verwaltungsinformationen für den Compiler, dem Schlüsselwort **interface** und ein erstes mal **uses**. Weiter unten kommt **implementation** und dann nochmals **uses**.

Unter **uses** werden Units aufgeführt, deren **interface**-Teil mitverwendet werden kann. Der Teil zwischen **interface** und **implementation** stellt gewissermaßen die Schnittstelle zur Unit dar, unterhalb **implementation** wird die eigentliche Arbeit gemacht. Dieser Teil ist ausserhalb der Unit main nicht sichtbar. Das zweite **uses** im **implementation**-Teil kann dazu benützt werden, um weitere Units zu verknüpfen, welche im oberen **interface**-Teil noch nicht benötigt werden. Dies hat den Vorteil, dass die Gefahr des gegenseitigen Aufrufs von Units geringer wird.

Beispielsweise können **Listing 26** und **Listing 27** nicht zusammenarbeiten, da in **uses** von a b und in **uses** von b wiederum a aufgeführt ist, was zum "Kurzschluss" oder "Circular unit reference" führt.

Der Compiler liest a.pas, findet b in **uses** von a, liest daher b.pas, findet a in **uses** von b, liest daher a.pas, findet b in **uses** von a...

Listing 28 hingegen funktioniert, da Unit a **uses** im **implementation**-Teil von b nicht sieht.

In **uses** im **interface**-Teil von main.pas hat MSEide bereits häufig benötigte Units aufgeführt.

```

type
  tmainfo = class(tmainform)
    eingabe: tintegeredit;
  end;

```

beschreibt die Schnittstelle zu unserem Formular, sie wurde in application.createform() bereits verwendet.

```

var
  mainfo: tmainfo;

```

Hier wird die Instanzvariable definiert worüber wir auf das erzeugte Formular zugreifen können, auch sie wurde in

```

object mainfo: tmainfo
  bounds_x = 291
  bounds_y = 247
  bounds_cx = 403
  bounds_cy = 280
  container.bounds = (
    0
    0
    403
    280
  )
  moduleclassname = 'tmainform'
object eingabe: tintegeredit
  frame.dummy = 0
  bounds_x = 48
  bounds_y = 24
  reffontheight = 14
end
end

```

Listing 24: Formulartext.

```

unit main;
{$ifdef FPC}{$mode objfpc}{$h+}{$endif}
interface
uses
  msetypes,mseglob,mseguiglob,mseguintf,mseapplication,msestat,msemenus,msegui,
  msegraphics,msegraphutils,mseevent,mseclasses,mseforms,msedataedits,mseedit,
  mseifcomp,mseifcompglob,mseifiglob,msestrings;

type
  tmainfo = class(tmainform)
    eingabe: tintegeredit;
  end;
var
  mainfo: tmainfo;
implementation
uses
  main_mfm;
end.

```

Listing 25: main.pas.

```

unit a;
interface
uses
  b;

```

Listing 26: Unit a (a.pas).

```

unit b;
interface
uses
  a;

```

Listing 27: Unit b (b.pas).

```

unit b;
interface
implementation
uses
a;

```

Listing 28: Unit b (b.pas) ohne “Circular unit reference”.

application.createform() bereits verwendet.

```

implementation
uses
    main_mfm;

```

main_mfm.pas ist die von MSEide aus main.mfm gebildete für den Compiler verständliche Datei, welche die Eigenschaftswerte des Formulars enthält. Klicke mit gedrückter Ctrl-Taste auf main_mfm, MSEide öffnet main_mfm.pas (**Listing 29**).

Diese Form ist wiederum für Menschen schwer verständlich. ;-)

Drücke Ctrl+F4 um die Datei main_mfm.pas zu schliessen.

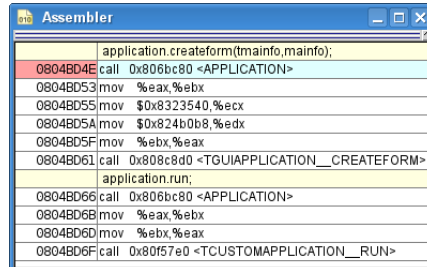


Bild 62: Prozessor-Befehle.

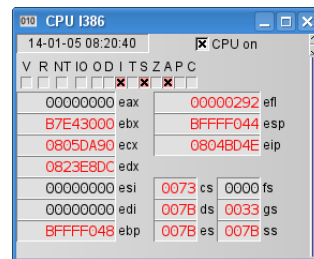


Bild 63: Prozessor-Register.

4.3. Was macht der Compiler?

Starte das Programm, es wird wieder beim gesetzten Breakpoint angehalten (**Bild 56**). Wie bereits erwähnt, liest der Compiler die zum Teil von uns geschriebenen Anweisungen aus verschiedenen Dateien und übersetzt sie in für den Computer verständliche Daten und Befehle.

View-Assembler zeigt die für die aktuelle Programmanweisung notwendigen Befehle (**Bild 62**). Ausgeführt werden die Befehle vom Prozessor (auch CPU = Central Processing Unit genannt).

In die CPU können wir ebenfalls hineinschauen; View-CPU zeigt ein Fenster, worin die Prozessorregister mit ihren Inhalten abgebildet sind (**Bild 63**).

Klicke in das Assembler-Fenster, mit F8 kann der Programmablauf Schritt für

Schritt verfolgt werden.

Bitte entferne den Breakpoint nach diesen Experimenten durch Doppelklick auf den roten Punkt in Zeile 10 von sprachgenie.pas.

4.4. Texte in Programmen

Auch zum Abspeichern und Verarbeiten von Texten stellt der Compiler Datentypen bereit, es sind die **string** (Zeichenkette)-Typen. Wie für die Zahlentypen gibt es mehrere verschiedene **string**-Typen, wir verwenden ausschliesslich **mstring**. Eine Definition und Zuweisung eines Textes an eine **mstring**-Variable sieht folgender-

```

unit main_mfm;
{$ifdef FPC}{$mode objfpc}{$h+}{$endif}

interface

implementation
uses
  mseclasses,main;

const
  objdata: record size: integer; data: array[0..201] of byte end =
    (size: 202; data: (
      84,80,70,48,7,116,109,97,105,110,102,111,6,109,97,105,110,102,111,8,
      98,111,117,110,100,115,95,120,3,35,1,8,98,111,117,110,100,115,95,121,
      3,247,0,9,98,111,117,110,100,115,95,99,120,3,147,1,9,98,111,117,
      110,100,115,95,99,121,3,24,1,16,99,111,110,116,97,105,110,101,114,46,
      98,111,117,110,100,115,1,2,0,2,0,3,147,1,3,24,1,0,15,109,
      111,100,117,108,101,99,108,97,115,115,110,97,109,101,6,9,116,109,97,105,
      110,102,111,114,109,0,12,116,105,110,116,101,103,101,114,101,100,105,116,7,
      101,105,110,103,97,98,101,11,102,114,97,109,101,46,100,117,109,109,121,2,
      0,8,98,111,117,110,100,115,95,120,2,48,8,98,111,117,110,100,115,95,
      121,2,24,13,114,101,102,102,111,110,116,104,101,105,103,104,116,2,14,0,
      0,0)
    );

  initialization
    registerobjectdata(@objdata,tmainfo,"");
  end.

```

Listing 29: Formulardatei main_mfm.pas.

massen aus:

```

var
  s1: mstring;
...
s1:= 'Der Text';

```

Der abzuspeichernde Text wird zwischen Hochkommas geschrieben. Soll ein Text ein Hochkomma enthalten, muss es verdoppelt werden.

```
s1:= 'Auf geht''s!';
```

speichert `Auf geht's!` in `s1`. Im Gegensatz zu Programmbezeichnern und Komponentennamen können Strings alle Schriftzeichen enthalten. Strings sind vergleichbar mit `array` als Aneinanderreihung von Zeichen. Im Gegensatz zu `array`, wo der Index des ersten Elementes meistens 0 ist, hat das erste Zeichen in einem String immer den Index 1. Die Elemente von `mstring`

haben den Typ `msechar`. `msechar` ist eine 16 Bit Zahl, deren Bedeutung als Schriftzeichen durch das Unicode-Normengremium festgelegt wurde [6]. Beispielsweise wurden für die Ziffern 0 bis 9 die Werte 48 bis 57 festgelegt.

Nicht immer entspricht ein `msechar` einem einzelnen Schriftzeichen, z.B. kann ein ö auch durch ein o mit zusätzlichem " dargestellt werden oder Zeichen mit Nummern, welche nicht in 16 Bits passen, werden auf zwei `msechar` aufgeteilt.

Strings können mit `+` zusammengesetzt werden.

```

s1:= 'abcde';
s1:= s1 + 'FGHIJ';

```

speichert `abcdeFGHIJ` in `s1`.

- Für Zeichenketten verwenden wir immer den Typ `mstring`.
- Das Zeichenelement des Typs `mstring` ist der Typ `msechar`.
- Der Index des ersten Zeichens in einem String ist 1.
- Strings können mit `+` zusammengefügt werden.

4.5. Gewinnung eines Ziffernwertes

Die eingegebene Zahl, welche den auszugebenden Betrag bestimmt, ist ein integer-Wert. Die Modulo 10 Operation liefert den Zehnerwert der niederwertigsten Ziffer. Dieser Wert liegt immer im Bereich 0 bis 9, darum definieren wir dafür ordnungshalber einen eigenen Bereichstyp:

```
type
  zifferty = 0..9;
var
  zehnerwert: zifferty;
...
zehnerwert:= wert mod 10;
```

Nun können wir den Zehnerwert in das entsprechende Schriftzeichen umwandeln. Dazu verwenden wir eine Funktion (**Listing 30**).

Eine Funktion ist eine Prozedur, welche einen Wert zurückgibt. Mit `round()` haben wir bereits eine Funktion verwendet. Wenn wir selber Funktionen programmieren wollen, schreiben wir statt `procedure` `function` und geben am Schluss des Kopfes nach `:` den Typ des zurückgegebenen Wertes an. Zugreifen auf den zurückgegebene Wert können wir innerhalb der Funktion mittels der Pseudovariablen `result`.

Wir könnten dasselbe Resultat mit der Prozedur in **Listing 31** erreichen. Die Verwendung einer Funktion statt einer Prozedur hat den Vorteil, dass wir das Resultat der Funktion in Ausdrücken verwenden können, also z.B.

```
s1:= s1 + zifferzuzeichen(zehnerwert);
```

statt

```
zifferzuzeichenproc(zehnerwert,c1);
s1:= s1 + c1;
```

wo zur Zwischenspeicherung des Resultats eine zusätzliche `msechar` Variable notwendig ist.

Zur Bestimmung des Schriftzeichens, welches zur entsprechenden Ziffer gehört, nutzen wir die Eigenschaft aus, dass die Nummernwerte der Schriftzeichen '0','1','2'... aufeinander folgen. Die Funktion `ord()` liefert die zu einem `msechar` gehörende 16 Bit Nummer. `ord('0') + ziffer` ist die gesuchte Nummer des zum Zehnerwert `ziffer` gehörenden Schriftzeichens. Mit `msechar()` wird für den Compiler die Nummer in ein Schriftzeichen (`msechar`) zurück gewandelt, welches als Resultat der `zifferzuzeichen()`-Funktion zurückgegeben wird (**Listing 30**).

Die Umwandlung der Zehnerwerte in Worte könnte man folgendermassen vornehmen:

```
if ziffer = 0 then begin
  result:= 'null';
end
else begin
  if ziffer = 1 then begin
    result:= 'eins';
  else begin
    ...
```

Zum Glück geht auch das bequemer:

```
case ziffer of
```

```
function zifferzuzeichen(ziffer: zifferty): msechar;
begin
  result:= msechar(ord('0') + ziffer);
end;
```

Listing 30: Ziffer-Umwandlungsfunktion.

```
procedure zifferzuzeichenproc(ziffer: zifferty; out result: msechar);
begin
  result:= msechar(ord('0') + ziffer);
end;
```

Listing 31: Prozedur statt Funktion.

```
0: begin
  result:= 'null';
end;
1: begin
  result:= 'eins';
end;
...
else begin
  result:= 'Fehler!';
end;
end;
```

Die **case** Anweisung führt aufgrund eines Schlüsselwertes verschiedene Befehlsgruppen aus. Die Schlüsselwerte dürfen nur einmal vorkommen. Falls der Schlüssel zu keinem “Schlüsselloch” passt, wird der **else**-Teil ausgeführt. Der **else**-Teil kann auch entfallen.

Auch für die Umwandlung einer Ziffer in ein Wort verwenden wir eine Funktion (Listing 32).

- **typename** = **kleinsterwert** .. **groessterwert**; definiert einen Bereichstyp.
- **function** **funktionsname** (**parameter**) : **rückgabety**; definiert eine Prozedur welche einen Wert zurückgibt.
- **case** verzweigt aufgrund eines Schlüsselwertes.

4.6. Umwandlung der gesamten Zahl

Die Umwandlung der Zahl in Text führen wir im Ereignis `eingabe.onsetvalue` aus. Alle Dateneingabekomponenten haben die Ereignisseigenschaft `onsetvalue`. `onsetvalue` wird aufgerufen nachdem die Anwenderin nach einer Dateneingabe Enter gedrückt hat, aber bevor der Wert in der `value`-Eigenschaft gespeichert ist. In `onsetvalue` besteht die Möglichkeit den Eingabewert zu überprüfen, zu verändern oder zurück zu weisen. Falls der Wert nicht zurückgewiesen wird, wird er in `value` gespeichert und anschliessend wird `ondataentered` aufgerufen. Die `onsetvalue`-Prozedur ist in Listing 33.

```
function zifferzuwort(ziffer: zifferty): mstring;
begin
  case ziffer of
    0: begin
      result:= 'null';
    end;
    1: begin
      result:= 'eins';
    end;
    2: begin
      result:= 'zwei';
    end;
    3: begin
      result:= 'drei';
    end;
    4: begin
      result:= 'vier';
    end;
    5: begin
      result:= 'fünf';
    end;
    6: begin
      result:= 'sechs';
    end;
    7: begin
      result:= 'sieben';
    end;
    8: begin
      result:= 'acht';
    end;
    9: begin
      result:= 'neun';
    end;
  end;
end;
```

Listing 32: Ziffer-Wort-Umwandlungsfunktion.

```

procedure tmainfo.setvalueexe(const sender: TObject; var avalue: Integer;
                                var accept: Boolean);
var
  ziffern: mstring;
  ziffernformatiert: mstring;
  worte: mstring;
  wert: integer;
  zehnerwert: zifferty;
  i1: integer;
begin
  if avalue <= 0 then begin
    showmessage('Ungültiger Betrag!');
    accept:= false;
  end
  else begin
    wert:= avalue;
    ziffern:= ""; //initialisieren
    ziffernformatiert:= "";
    worte:= "";
    repeat
      zehnerwert:= wert mod 10;
      ziffern:= ziffern + zifferzuzeichen(zehnerwert); //reihenfolge umgekehrt
      worte:= zifferzuwort(zehnerwert) + '-' + worte; //reihenfolge richtig
      wert:= wert div 10;
    until wert = 0;
    setlength(worte,length(worte)-1); //letztes '-' abtrennen
    for i1:= length(ziffern) downto 1 do begin
      if (i1 mod 3 = 0) and (i1 <> length(ziffern)) then begin
        ziffernformatiert:= ziffernformatiert + ' ';
        //jede dritte ziffer ein leerzeichen einfuegen
      end;
      ziffernformatiert:= ziffernformatiert + ziffern[i1];
    end;
    zifferdisp.value:= ziffern;
    ziffernformatdisp.value:= ziffernformatiert;
    wortedisp.value:= worte;
  end;
end;

```

Listing 33: eingabe.onsetvalue.

Zuerst wird geprüft, ob der eingegebene Wert kleiner oder gleich Null ist. In diesem Fall wird eine Fehlermeldung angezeigt und der Wert nicht akzeptiert. Die Prozedur `showmessage()` (zeige Meldung) erzeugt ein Meldungsfenster mit dem angegebenen Text.

Falls der Eingabewert in Ordnung ist, werden die Variablen initialisiert. " entspricht einem leeren String. Danach werden die einzelnen Ziffern in einer **repeat** ... **until** (wiederholen - bis) - Schleife abgearbeitet (**Listing 34**). Die Zeichen in der `ziffern`-Variable stehen in umgekehrter Reihenfolge, die Zeichenreihenfolge wird später beim Einfügen der Leereichen für jede dritte Stelle umgedreht. Nach der Auswertung einer Zehnerstelle wird `wert` durch 10 geteilt, dadurch rutschen die Zehnerstellen um eine Position nach rechts. Ist `wert` nach der Division Null, bedeutet dies, dass alle Stellen "gegessen" sind und die Schleife verlassen werden kann.

```
setlength(worte, length(worte)-1);
```

trennt das letzte unerwünschte "-" ab. Beim Einfügen der 3-er-Stellen-Leezeichen muss beachtet werden, dass Vergleichsoperationen weniger binden als Rechen- und Logikoperationen, darum die Klammern in

```
if (i1 mod 3 = 0) and
    (i1 <> length(ziffern)) then begin
```

Zur Anzeige der Werte verwenden wir drei `tstringdisp`, die mit `abc` bezeichnete Komponente aus dem Bereich `Widget`, `name = zifferdisp`, `ziffernformatdisp` und `wortedisp`. `ziffernformatdisp` machen wir breiter, damit die Worte besser Platz finden (**Bild 64**).

Damit sich die Breite von `ziffernformatdisp` der Fensterbreite anpasst, kann in `anchors` (Anker) `an_right` auf `true` gesetzt werden. `an_left`, `an_top`, `an_right` und

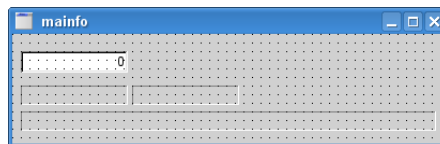


Bild 64: Anordnung der Komponenten.

`an_bottom` bestimmen die Ränder, welche "am Boden festgeklebt" sind. Normalerweise sind es `an_left` und `an_top`, das heisst, Position und Grösse werden durch die Fenstergrösse nicht beeinflusst. Möchten wir die Breite mit der Fensterbreite vergrössern und verkleinern, müssen `an_left` und `an_right` auf `true` stehen, wie wir es bei `ziffernformatdisp` gemacht haben. Soll sich die Komponente mit dem rechten Fensterrand bewegen, setzen wir `an_right` auf `true` und `an_left` auf `false`. Nicht umgekehrt, da für `an_left = false` und `an_right = false` die gesamte zur Verfügung stehende Fensterbreite ausgenützt wird!

Für die Komponentenhöhe verwenden wir entsprechend `an_top` und `an_bottom`.

```

repeat
  zehnerwert:= wert mod 10;
  ziffern:= ziffern + zifferzuzeichen(zehnerwert); //reihenfolge umgekehrt
  worte:= zifferzuwort(zehnerwert) + '-' + worte; //reihenfolge richtig
  wert:= wert div 10;
until wert = 0;

```

Listing 34: Schleife zur Abarbeitung der Ziffern.

- Nach onsetvalue wird der eingegebene Wert in value abgespeichert und ondataentered aufgerufen.
- **repeat** ... **until** booleanausdruck ; bezeichnet eine Programmschleife, welche abgebrochen wird, wenn booleanausdruck **true** ist.
- **while** booleanausdruck **do** **begin** ... **end** ; bezeichnet eine Programmschleife, welche solange ausgeführt wird, als booleanausdruck **true** ist.
- **length(stringausdruck)** bringt die Anzahl der Elemente.
- **setlength(stringvariable,laenge)** setzt die Anzahl Element in stringvariable. Die Inhalte der zusätzlichen Stellen sind undefiniert und enthalten zufällige Werte.
- Logische Operationen (**and**, **or**, **xor**) und algebraisch Operationen (**+**, **-**, *****, **/**, **div**, **mod**) binden stärker als Vergleichsoperationen (**=**, **<**, **>**, **<=**, **>=**, **<>**).

5. Die Aufzugssteuerung



Bild 65: Aufzugssimulation.

Dies wird eine sehr anspruchsvolle Aufgabe. Ziel ist die Programmierung einer Aufzugsteuerung und die Simulation und Darstellung des Betriebs.

5.1. Darstellung

Zur Darstellung des Aufzugschachts und der Stockwerke verwenden wir eine tpaintbox. Die Kabine ist eine zweite tpaintbox, welche im Schacht bewegt wird. Eine weitere tpaintbox stellt die Bedientafel in der Kabine dar. Auf jedem Stockwerk gibt es einen Pfeil-aufwärts- und einen Pfeil-abwärts-Knopf zur Anforderung der Kabine. Auf der Kabinenensteuertafel ist für jedes Stockwerk einen Ziel-Knopf vorhanden.

Ziehe das Formular in die Länge, setze eine tpaintbox an den linken Rand, Name = schacht, frame.levelo = -1, bounds_cx = 114, bounds_cy = 510.

Setze eine weitere tpaintbox in die schacht-Komponente. Name = kabine, frame.levelo = 1, color = cl_dkgray. kabine ist ein "Kind" von schacht, wie schacht ein Kind von container des Formulars ist. Wird schacht verschoben, wird auch kabine verschoben. Bitte kontrolliere, ob dies der Fall ist. Falls nicht, hast du kabine nicht in schacht gesetzt, sondern daneben. Falls notwendig lösche kabine und probiere es nochmals.

Setze eine weitere tpaintbox rechts neben schacht, Name = bedienfeld, bounds_cx = 154, bounds_cy = 327, frame.levelo = -1, frame.leveli = 1, frame.caption = Bedienfeld in Kabine.

Platziere drei trealdisp (Abteilung Widget, mit 1.21 bezeichnet, real entspricht float) in bedienfeld, Name = posdisp, geschwdisp, beschdisp, frame.caption = Position, Geschwindigkeit, Beschleunigung (siehe Bild 66).

Für jedes Stockwerk benötigen wir folgende Informationen:

- Stockwerksbezeichnung.
- Türe geöffnet.
- Aufzug-auf Anforderung gedrückt.
- Aufzug-ab Anforderung gedrückt.
- Stockwerk als Ziel auf Kabinensteuertafel gewählt.



Bild 66: Aufzugskomponenten.

Wir gruppieren die Informationen in einen **record** Typ und bauen ein Array davon in die Aufzugsinformation ein (**Listing 35**).

Weiter benötigen wir die Abmessungen und Positionen der Kabinen-Anforderungstasten und deren Pfeile. Zum Zeichnen der Tasten benutzen wir die `tcanvas`-Prozeduren `fillrect()` (`fill` = füllen, `rectangle` = Rechteck) und `fillpolygon()` (Polygon = Vieleck) für die Pfeile. `fillpolygon()` erwartet ein array of `pointty`, worin die Eckpunkte der zu füllenden Fläche aufgeführt sind. Für die Darstellung der Pfeile verwenden wir ein einfaches Dreieck, dessen Definition für `fillpolygon()` ist ein `array[0..2]` of `pointty`.

Ein Rechteck wird durch den **record**-Typen `rectty` bestimmt. `rectty` ist definiert als

```
rectty = record
  case integer of
    0: (x,y,cx,cy: integer);
    1: (pos: pointty;
```

```
      size: sizety);
  end;
```

Wir erinnern uns, `pointty` ist definiert als

```
pointty = record
  x,y: integer;
end;
```

`sizety` (`size` = Abmessung) ist definiert als

```
sizety = record
  cx,cy: integer;
end;
```

case in `rectty` zeigt verschieden Sichten auf die gleichen Daten. Wir können `rectty` als Zusammenfassung von x-Position, y-Position, Breite und Höhe oder als Zusammenfassung von Position und Grösse auffassen. Es kann aber auch sein, dass die einzelnen **case** Teile in einer **record** Definition nichts miteinander zu tun haben, sondern lediglich den gleichen Speicherplatz teilen.

Die Umrisse der Tasten müssen wir nur einmal für alle Stockwerke gemeinsam bestimmen, beim Zeichnen verschieben wir den Ursprung jeweils um ein Stockwerk. In `oncreate` des Formulars berechnen wir die Abmessungen (**Listing 37**), dabei verwenden wir die Definitionen von **Listing 36**.

Die Höhe von `schacht` wird durch

```
schacht.clientheight:=
  round(massstab * schachthoehe);
```

gesetzt.

`clientheight` bestimmt die Höhe des Rechtecks im Innern des Rahmens (`clientrect`). Wenn mehrere Operationen mit Teilen einer Record-Variable durchgeführt werden, kann der Zugriff durch **with recordvariable do begin ... end ;** vereinfacht werden. Für den Zugriff auf die `aufzug`-Variable verwenden wir eine


```

type
stockwerkinfoty = record
  bezeichnung: mstring;
  tuereauf: boolean;
  anforderungauf: boolean;
  anforderungab: boolean;
  ziel: boolean; //true wenn entsprechende taste auf bedienfeld gedrueckt
end;

aufzuginfoty = record
  stockwerke: array[0..stockwerkzahl-1] of stockwerkinfoty;
end;

```

Listing 35: stockwerkinfoty.

```

implementation

uses
  main_mfm, sysutils;

const
  massstab = 20.0; //pixel pro m
  stockwerkzahl = 11;
  stockwerkhoehe = 2.4; // m
  kabinenhoehe = 2.0; // m
  kabinenhoehepix = round(kabinenhoehe*massstab);
  kabinenbreite = 3.0; // m
  kabinenbreitepix = round(kabinenbreite*massstab);
  schachthoehe = stockwerkzahl * stockwerkhoehe;
  tastenxpix = round((kabinenbreite+0.2)*massstab);
  tastenypix = round(1.5*massstab);
  tastengroessepix = 10;

type
stockwerkinfoty = record
  bezeichnung: mstring;
  tuereauf: boolean;
  anforderungauf: boolean;
  anforderungab: boolean;
  ziel: boolean; //true wenn entsprechende taste auf bedienfeld gedrueckt
end;

aufzuginfoty = record
  stockwerke: array[0..stockwerkzahl-1] of stockwerkinfoty;
end;

var
  tasteaufrect: rectty;
  tasteabrect: rectty;
  tasteaufpfeil: array[0..2] of pointty;
  tasteabpfeil: array[0..2] of pointty;
  bedienfelddtastenstart: integer;
  bedienfelddtastenrect: rectty;
  bedienfelddtastendistanz: integer;
  aufzug: aufzuginfoty;

```

Listing 36: Konstanten-, Typen- und Variablen-Definitionen.

```

procedure tmainfo.createexe(const sender: TObject);
var
  i1: integer;
begin
  tasteaufrect.x:= tastenxpix;
  tasteaufrect.y:= -tastenypix;
  tasteaufrect.cx:= tastengroessepix;
  tasteaufrect.cy:= tastengroessepix;
  tasteabrect:= tasteaufrect;
  tasteabrect.y:= tasteabrect.y + tastengroessepix + 1;
  tasteaufpfeil[0].x:= tasteaufrect.x + 1;
  tasteaufpfeil[0].y:= tasteaufrect.y + tasteaufrect.cy - 1;
  tasteaufpfeil[1].x:= tasteaufrect.x + tasteaufrect.cx div 2;
  tasteaufpfeil[1].y:= tasteaufrect.y + 1;
  tasteaufpfeil[2].x:= tasteaufrect.x + tasteaufrect.cx - 1;
  tasteaufpfeil[2].y:= tasteaufpfeil[0].y;
  tasteabpfeil:= tasteaufpfeil;
  tasteabpfeil[0].y:= tasteabrect.y + 1;
  tasteabpfeil[1].y:= tasteabrect.y + tasteabrect.cy - 1;
  tasteabpfeil[2].y:= tasteabpfeil[0].y;
  bedienfeldtastenstart:= beschdisp.bounds_y +
                           beschdisp.bounds_cy + 10;
  bedienfeldtastendistanz:= tasteaufrect.cy+1;
  bedienfeldtastenrect.x:= beschdisp.bounds_x;
  bedienfeldtastenrect.y:= 0;
  bedienfeldtastenrect.size:= tasteaufrect.size; //gleiche groessen

  schacht.clientheight:= round(massstab * schachthoehe);
  kabine.bounds_x:= 1; //linksbuendig, 1 pixel breiter schacht rahmen
  kabine.bounds_cx:= kabinenbreitepix;
  kabine.bounds_cy:= kabinenhoehepix;
  posdisp.value:= 0.0;
  geschwdisp.value:= 0.0;
  beschdisp.value:= 0.0;

  with aufzug do begin
    for i1:= 0 to high(stockwerke) do begin
      case i1 of
        0: begin
          stockwerke[i1].bezeichnung:= 'UG';
        end;
        1: begin
          stockwerke[i1].bezeichnung:= 'EG';
        end;
        else begin
          stockwerke[i1].bezeichnung:= inttostr(i1-1); //in unit sysutils
        end;
      end;
    end;
  end;
end;

```

Listing 37: Berechnung der Abmessungen der Zeichenelemente.

solche `with`-Konstruktion. `inttostr()` wandelt einen `integer`-Wert in einen `string` um. `inttostr()` ist in der unit `sysutils` (Systemwerkzeuge) definiert, darum muss `sysutils` in `uses` aufgeführt werden (**Listing 36**).

Die Bedienelemente der Kabinenesteuertafel werden in `bedienfeld.onpaint` gezeichnet (**Listing 38**).

Die `move()`-Prozedur einer `tcanvas`-Komponente verschiebt den Zeichnungsnulldpunkt (Ursprung) um die `x`- und `y`-Werte des übergebenen `pointty`. Am Anfang der `onpaint`-Prozedur ist der Nullpunkt die linke, obere Ecke des `clientrect`. `mp()` (make point = mache Punkt) setzt aus den übergebenen `x`- und `y`-Werten den von `move()` erwarteten `pointty`-Wert zusammen. `tcanvas.drawstring()` gibt einen Text an der angegebenen Stelle aus. Der Referenzpunkt ist die Grundlinie des Beginns des ersten Zeichens.

Teste das Programm, es sollte etwa wie **Bild 67** aussehen.



Bild 67: Bedientafel der Kabine.

- `record case typ of wert :`
`(...); ... end;` definiert einen Record-Typen mit Variantenteilen.
- `with recordvariable do`
`begin ... end ;` sucht Bezeichner zuerst in `recordvariable`.

5.2. Mausclick

Wenn mit der Maus in ein Rechteck eines Tastensymbols geklickt wird, soll die `ziel`-Variable in `stockwerkinfoty` des entsprechenden Stockwerkes gesetzt werden. Auf Mausereignisse innerhalb des `clientrect` können wir in `onclientmouseevent` reagieren. Trage in `bedienfeld.onclientmouseevent` `bedienfeld.mouseevent` ein, die Ereignis-Behandlungsprozedur ist in **Listing 39**.

Im Prozedur-Kopf kann über `sender` auf die Komponente zugegriffen werden von der das Ereignis stammt, das heisst, das Widget innerhalb dessen `clientrect` sich der Mauszeiger befindet. `ainfo` stellt Informationen über das Mausereignis zur Verfügung. `ainfo` ist definiert als

```
mouseeventinfoty = record
  eventkind: eventkindty;
  shiftstate: shiftstatesty;
  pos: pointty;
  eventstate: eventstatesty;
  timestamp: longword;
  button: mousebuttonsty;
end;
```

`eventkind` ist definiert in **Listing 40** und beschreibt die Art des Ereignisses als Aufzählung. Ein Aufzählungstyp ist eine Nummerierung mit Namen statt Zahlen. Intern werden trotzdem Zahlen verwendet, die Namen erleichtern das Verständnis des Programmes. Eine Aufzählungstyp zur Nummerierung der Wochentage könnte zum

```

procedure tmainfo.bedienfeldpaint(const sender: twidget;
                                   const acanvas: tcanvas);

procedure zeichneetagenknopf(const stockwerkinfo: stockwerkinfoty);
var
  col: colorty;
begin
  if stockwerkinfo.ziel then begin
    col:= cl_ltyellow;
  end
  else begin
    col:= cl_dkgray;
  end;
  acanvas.fillrect(bedienfeldtastenrect,col);
  with bedienfeldtastenrect do begin
    acanvas.drawstring(stockwerkinfo.bezeichnung,mp(x+cx+5,y+cy));
  end;
end; //zeichneetagenknopf

var
  i1: integer;
begin
  acanvas.move(mp(0,bedienfeldtastenstart));
  with aufzug do begin
    for i1:= high(stockwerke) downto 0 do begin
      zeichneetagenknopf(stockwerke[i1]);
      acanvas.move(mp(0,bedienfeldtastendistanz));
    end;
  end;
end;

```

Listing 38: Kabinen-Bedienelemente zeichnen.

Beispiel folgendermassen aussehen:

```
type
wochentagety = ( so , mo , di , mi ,
                do , fr , sa ) ;
```

Der erste Namen entspricht 0, der zweite entspricht 1, usw.

Mit `ek_buttonpress` reagieren wir nur auf Maustastendrücke. In `ainfo.pos` ist die Position des Mauspfels bezogen auf die linke obere Ecke des `clientrect` von `sender` - in unserem Fall `bedienfeld` - abgelegt.

Die Funktion `pointinrect()` liefert `true`, wenn sich der geprüfte Punkt innerhalb des Rechtecks befindet. Falls in eine Taste geklickt wurde, wird die `ziel` `boolean` Variable des entsprechenden Stockwerkes gesetzt. Da dann die Bedientafel möglicherweise zum Anzeigen der beleuchteten Taste neu gezeichnet werden muss, wird `bedienfeld.invalidate()` aufgerufen.

Teste das Programm, geklickte Tastenfelder sollten beleuchtet dargestellt werden (Bild 68).

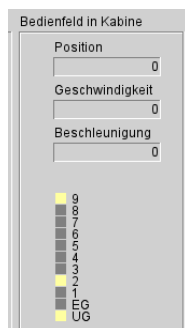


Bild 68: Tastenfeld mit Maus-Auswertung.

Falls es nicht funktioniert, gibt es verschiedene mögliche Ursachen. Entweder wurde durch das Klicken die `ziel`-Variable nicht gesetzt oder die Taste wird trotz gesetzter `ziel`-Variable nicht beleuchtet dargestellt. Im ersten Fall liegt der Fehler vermutlich in der Prozedur

`bedienfeldmouseevent()`, im zweiten Fall eher in der Prozedur `bedienfeldpaint()`.

Ein laufendes Programm wird in der Kommandozeile durch `Target-Interrupt` (Ziel unterbrechen) oder durch Klicken auf die beiden blauen senkrechten Striche rechts neben dem Reset-Knopf unterbrochen (Bild 69).

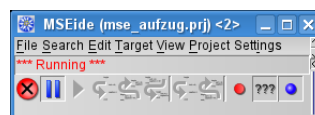


Bild 69: Programm unterbrechen.

Ist ein Programm unterbrochen, können die Werte der globalen Variablen kontrolliert werden. `View-Watches` zeigt ein Fenster mit einer Tabelle. In der Spalte `Expression` (Ausdruck) wird die Variable bestimmt deren Inhalt in `Result` (Ergebnis) angezeigt wird (Bild 70).

	Expression	Result
1	<code>aufzug.stockwerke[0].ziel</code>	true
2	<code>aufzug.stockwerke[1].ziel</code>	false
3	<code>aufzug.stockwerke[2].ziel</code>	false
4	<code>aufzug.stockwerke[3].ziel</code>	true
5	<code>aufzug.stockwerke[4].ziel</code>	false

Bild 70: Variableninhalte ansehen.

Falls beispielsweise die UG-Taste trotz Klicken nicht erhellt dargestellt wird, kontrollieren wir den Wert von `aufzug.stockwerke[0].ziel`. Ist er `true`, wurde der Mausklick erkannt und wir setzen einen Haltepunkt in `tmainfo.bedienfeldpaint()`, um zu verfolgen, was beim Zeichnen passiert. Ist der Wert `false`, setzen wir einen Haltepunkt in `tmainfo.bedienfeldmouseevent()`, um zu erkunden, warum der Klick nicht erkannt wird. Im gezeigten Beispiel ist alles in Ordnung. Mit Druck von F9 läuft das

```

procedure tmainfo.bedienfeldmouseevent(const sender: twidget;
                                         var ainfo: mouseeventinfoty);
var
  pt1: pointty;
  i1,i2: integer;
begin
  if ainfo.eventkind = ek_buttonpress then begin
    pt1:= ainfo.pos;
    pt1.y:= pt1.y - bedienfeldtastenstart;
                                     //oberste taste entspricht null
    i1:= pt1.y div bedienfeldtastendistanz; //anzahl tasten von oben
    i2:= high(aufzug.stockwerke) - i1;      //stockwerknummer
    if (i2 >= 0) and (i2 <= high(aufzug.stockwerke)) then begin //gueltig?
                                     //ja
      pt1.y:= pt1.y - i1 * bedienfeldtastendistanz;
                                     //taste des aktuellen stockwerkes entspricht null
      if pointinrect(pt1,bedienfeldtastenrect) then begin
                                     //wurde auf taste geklickt?
        aufzug.stockwerke[i2].ziel:= true; //ja
        bedienfeld.invalidate(); //muss eventuell neu gezeichnet werden
      end;
    end;
  end;
end;

```

Listing 39: Maus Auswertung für Kabinen-Bedienfeld.

```

eventkindty = (ek_none,ek_focusin,ek_focusout,ek_checkapplicationactive,
               ek_enterwindow,ek_leavewindow,
               ek_buttonpress,ek_buttonrelease,ek_mousewheel,
               ek_mousemove,ek_mousepark,
               ek_mouseenter,ek_mouseleave,ek_mousecaptureend,
               ek_clientmouseenter,ek_clientmouseleave,
               ek_expose,ek_configure,
               ek_terminate,ek_abort,ek_destroy,ek_show,ek_hide,ek_close,
               ek_activate,ek_loaded,
               ek_keypress,ek_keyrelease,ek_timer,ek_wakeup,
               ek_release,{ek_releasedefer,}ek_closeform,ek_checkscreenrange,
               ek_childscaled,ek_resize,
               ek_dropdown,ek_async,ek_execute,ek_object,ek_component,
               ek_asyncexec,ek_releaseobject,
               ek_connect,
               ek_dbedit,ek_dbupdaterowdata,ek_data,ek_objectdata,ek_childproc,
               ek_dbinsert, //for tdscontroller
               ek_sysdnd,ek_sysdndstatus,
               ek_mse,
               ek_user);

```

Listing 40: eventkindty Aufzählung.

Programm weiter.

- `type`
`aufzaehlungsname =`
`(name0 , name1 , name2 , ...) ;`
 definiert einen Aufzählungstyp.

5.3. Stockwerke zeichnen

Die Stockwerke werden über die Kabine gezeichnet, darum verwenden wir `schacht.onafterpaint` (= nach Zeichnen) statt `onpaint` (Listing 41). In `onpaint` würde die Stockwerkzeichnung von der Kabine verdeckt.

Ein einzelnes Stockwerk wird in der Unterprozedur `zeichnestockwerk()` gezeichnet, dabei wird mitgegeben, ob es sich um das erste oder das letzte Stockwerk handelt, damit die Auf- respektive die Ab-Taste weggelassen werden kann. `tcanvas.drawLine()` zeichnet eine Linie zwischen zwei Punkten. Die etwas umständliche Berechnung der Stockwerksverschiebung mit den lokalen Variablen `i2` und `schiebung` stellt sicher, dass sich Rundungsfehler nicht summieren können. Der Zeichnungsursprung in `zeichnestockwerk()` ist der Stockwerksboden, senkrechte Höhen (y-Werte) müssen daher als negative Zahlen aufgefasst werden, `tcanvas` y-Werte nehmen ja von oben nach unten zu.

Teste das Programm, der Aufzugsschacht müsste etwa wie **Bild 71** aussehen.

5.4. Mausbehandlung für die Stockwerke

Die Mausklickauswertung der Stockwerk-Tasten funktioniert ähnlich wie bei der Bedientafel. Die Prozedur für



Bild 71: Gezeichnete Stockwerke.

`schacht.onclientmouseevent` ist in **Listing 42** ausgeführt.

Bitte prüfe, ob die Fahrstuhl-Anforderungstasten beim Aufzugsschacht nach Klicken beleuchtet dargestellt werden.

Ein weiteres Werkzeug zur Kontrolle des Programmablaufes ist `View-Stack` (`stack = Stapel`). Setze einen Haltepunkt auf

```
if stockwerkinfo.ziel then begin
```

```
  in zeichneetagenknopf().
```

Starte das Programm, es bleibt beim Haltepunkt stehen (**Bild 72**). Im `Stack`-Fenster sehen wir wie die “aufeinandergestapelten” Prozeduraufrufe. Die unterste (= erste) Prozedur ist `main()` (= haupt) welche `tcustomapplication.run()` aufruft. Wenn wir auf die Zeile 10 doppelklicken, wird die entsprechende Stelle angezeigt (**Bild 73**).

```
Erwartungsgemäss wird
tmainfo.bedienfeldpaint() als
zweite Zeile aufgeführt, den Aufruf
von zeichneetagenknopf() in
tmainfo.bedienfeldpaint() haben wir
ja selbst programmiert.
```

```

procedure tmaininfo.schachtafterpaint(const sender: twidget;
                                     const acanvas: tcanvas);

procedure zeichnestockwerk(const stockwerkinfo: stockwerkinfoty;
                          erstes: boolean; letztes: boolean);

var
  col: colorty;
begin
  acanvas.drawline(mp(0,0),mp(schacht.bounds_cx,0)); //boden
  acanvas.drawline(mp(0,-kabinenhoehepix),
                  mp(schacht.bounds_cx,-kabinenhoehepix)); //decke
  acanvas.drawline(mp(kabinenbreitepix,-kabinenhoehepix),
                  mp(kabinenbreitepix,0),cl_dkgray); //rechts
  if stockwerkinfo.tuereauf then begin
    acanvas.drawline(mp(2,-kabinenhoehepix),mp(2,0)); //links
    acanvas.drawline(mp(kabinenbreitepix-2,-kabinenhoehepix),
                    mp(kabinenbreitepix-2,0)); //rechts
  end
  else begin
    acanvas.drawline(mp(kabinenbreitepix div 2,-kabinenhoehepix),
                    mp(kabinenbreitepix div 2,0)); //mitte
  end;
  if not letztes then begin
    acanvas.fillrect(tasteaufrect,cl_gray);
    if stockwerkinfo.anforderungauf then begin
      col:= cl_ltyellow;
    end
    else begin
      col:= cl_black;
    end;
    acanvas.fillpolygon(tasteaufpfeil,col);
  end;
  if not erstes then begin
    acanvas.fillrect(tasteabrect,cl_gray);
    if stockwerkinfo.anforderungab then begin
      col:= cl_ltyellow;
    end
    else begin
      col:= cl_black;
    end;
    acanvas.fillpolygon(tasteabpfeil,col);
  end;
  acanvas.drawstring(stockwerkinfo.bezeichnung,
                    mp(tasteabrect.x+tasteabrect.cx+5,tasteabrect.y));
  //etagen beschriftung
end;

var
  i1,i2: integer;
  schiebung: integer;
begin
  //y richtung = von oben nach unten
  acanvas.move(mp(0,round(stockwerkhoehe*stockwerkzahl*massstab)));
  //ursprung = erdgeschossboden
  schiebung:= 0; //summe der ursprungsverschiebung
  for i1:= 0 to stockwerkzahl - 1 do begin
    i2:= round(i1 * stockwerkhoehe * massstab) - schiebung;
    //notwendige ursprungsverschiebung
    acanvas.move(mp(0,-i2)); //aufwaertsverschiebung
    schiebung:= schiebung + i2; //summe nachfuehren
    zeichnestockwerk(aufzug.stockwerke[i1],i1=0,i1=stockwerkzahl-1);
  end;
end;

```

Listing 41: Stockwerke zeichnen.


```

procedure tmaininfo.schachtmouseevent(const sender: twidget;
                                         var ainfo: mouseeventinfoty);
var
  stockwerk: integer;
  pt1: pointty;
begin
  if ainfo.eventkind = ek_buttonpress then begin
    pt1:= ainfo.pos;
    stockwerk:= trunc(((maininfo.schacht.clientheight-pt1.y) / massstab) /
                      stockwerkhoehe);

    if (stockwerk >= 0) and (stockwerk < stockwerkzahl) then begin
      pt1.y:= pt1.y - round((stockwerkzahl-stockwerk)*stockwerkhoehe*massstab);
      if pointinrect(pt1,tasteaufrect) then begin
        aufzug.stockwerke[stockwerk].anforderungauf:= true;
      end;
      if pointinrect(pt1,tasteabrect) then begin
        aufzug.stockwerke[stockwerk].anforderungab:= true;
      end;
      schacht.invalidate(); //koennte veraendert sein
    end;
  end;
end;

```

Listing 42: Mausklick-Auswertung für die Anforderungstasten.

Die dazwischenliegenden Prozeduren sind Teile der MSEgui-Programmbibliothek, welche Aufgaben erledigen, die in allen Programmen bewältigt werden müssen und uns viel Arbeit abnehmen. Die Entwicklung einer solchen Entwicklungsumgebung ist sehr aufwändig, die Entwicklung von MSEide+MSEgui dauerte z.B. mehr als zehn Jahre.

Es ist möglich, die verschiedenen MSEide Fenster zu gruppieren. Es wäre zum Beispiel praktisch, die Fenster **Stack** und **Watches** zusammen zu packen. Führe in der Kommandozentrale **View-Panels-New Panel** aus. Es erscheint ein neues leeres Fenster (**Bild 74**).

Nun kann das **Stack**-Fenster an der senkrechten Leiste am rechten Rand mit der Maus "gepackt" und in **Panel** gezogen werden. Maustaste in **Panel** loslassen und **Stack** wird in **Panel** verschoben (**Bild 75**).

Machen wir dasselbe mit **Watches** und auch **Watches** wird in **Panel** verschoben (**Bild 76**).

Sowohl **Stack** als auch **Watches** haben ei-



Bild 74: Leeres Panel.

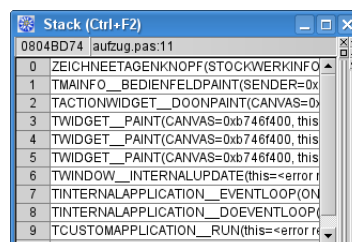


Bild 75: Stack in Panel.

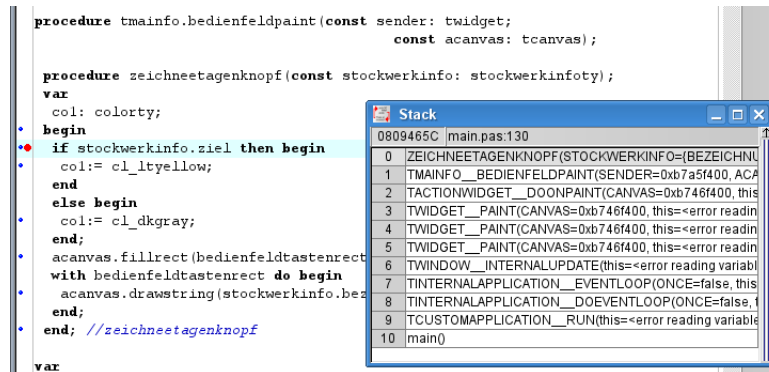


Bild 72: Aufruf-Stack-Fenster.

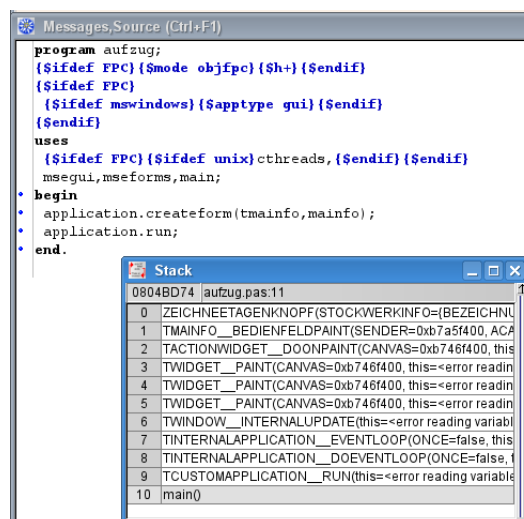


Bild 73: main() Prozedur.

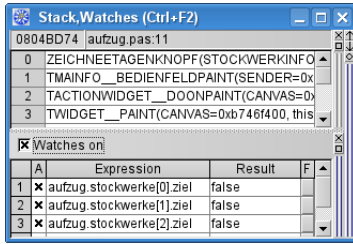


Bild 76: Stack und Watches in Panel.

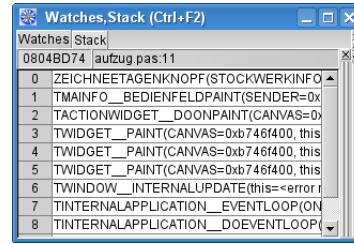


Bild 79: Register-Darstellung.

ne seitliche Fensterleiste. Damit kann die Reihenfolge der Fensterbereiche bestimmt werden (Bild 77).

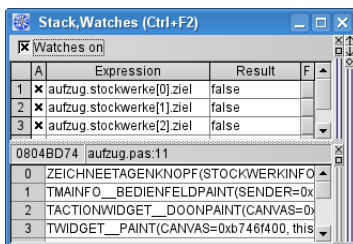


Bild 77: Geänderte Reihenfolge von Stack und Watches.

Ziehen der Leiste eines Fensterbereichs auf die gegenüberliegende Seite ändert die Teilungs-Ausrichtung von waagrecht zu senkrecht (Bild 78) oder umgekehrt.

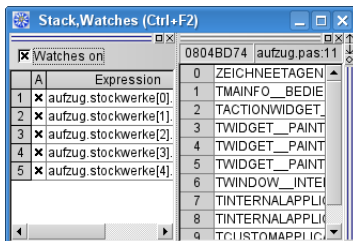


Bild 78: Vertikale Teilung.

Ziehen der Leiste eines Fensterbereichs in die Mitte schaltet auf registerartige Darstellung um (Bild 79).

Durch Klicken auf das kleine Kreissymbol der Fensterleisten werden die Leisten der innen liegenden Fensterbereiche verborgen (Bild 80).

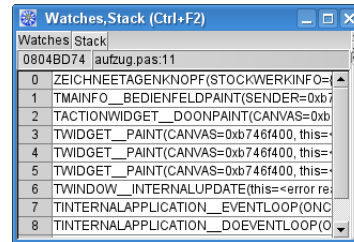


Bild 80: Verborgene Bereichs-Fensterleisten.

5.5. Aufzugsmotor

Zur Simulation des Verhaltens des Aufzugmotors schreiben wir Listing 43. `motortick()` muss entsprechend dem Ticken einer Uhr in regelmässigen Abständen aufgerufen werden. Damit der Ablauf für das Auge fließend erscheint, muss der Aufruf mindestens 50 mal pro Sekunde erfolgen und zudem dürfen sich die Verhältnisse zwischen den Aufrufen nicht allzu sehr verändert haben, sonst wird die Simulation ungenau.

Wird `motortick()` z.B. nur alle 5 Minuten aufgerufen, ist klar, dass die Simulation kein vernünftiges Ergebnis zeigen wird, da der Fahrstuhl in 5 Minuten viele Stockwerke durchfahren kann und die Motorposition in dieser Zeit nur einmal nachgeführt wird. Dadurch werden wichtige Ereignisse verpasst und die Simulation wird unbrauchbar.

Wir ergänzen `const` mit

```
const
masstab = 20.0; //pixel pro m
```

```

type
  motorinfoty = record
    sollgeschwindigkeit: float; //m/s, negativ -> abwaerts
    endschalteroben: boolean; //true wenn betätigt
    endschalterunten: boolean; //true wenn betätigt
    kabinenposition: float; //m, distanz des kabinenbodens vom schachtboden
  end;

var
  motor: motorinfoty;

procedure motortick(var motorinfo: motorinfoty);
begin
  with motorinfo do begin
    if (sollgeschwindigkeit > 0) and not endschalteroben or
        (sollgeschwindigkeit < 0) and not endschalterunten then begin
      kabinenposition:= kabinenposition + sollgeschwindigkeit / tickprosekunde;
    end;
  end;
end;
end;

```

Listing 43: Simulation des Motors.

tickprosekunde = 50;

Die Verbindung zwischen Simulation und Anzeige wird in der Prozedur `visualisierung()` hergestellt (**Listing 44**).

Der Tick wird wie beim PanoRamato-graph durch `ontimer` einer `ttimer` Komponente aus dem Bereich NoGui erzeugt. Name = tick, `ontimer` = tickexe. In `tickexe()` werden `motortick()` und `visualisierung()` aufgerufen (**Listing 45**). Die `schachtgeaendert`-Variable werden wir erst später benutzen.

In der Initialisierungs-Prozedur `tmainfo.createexe()` ergänzen wir **Listing 46**.

Teste das Programm, die Kabine sollte sich langsam nach oben bewegen und an der Decke des obersten Stockwerkes stehen bleiben (**Bild 81**). Dabei fällt auf, dass die Anzeige der Kabinenposition am Anfang zwei Nachkommastellen hat und dann plötzlich auf `x.xx99999999...` umspringt.

Dies ist ein Effekt, der bei Float-Zahlen immer berücksichtigt werden muss. Dezimalzahlen im Zehnersystem lassen sich nicht immer exakt im Binärsystem des Computers darstellen. Darum muss man

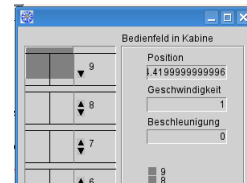


Bild 81: Motor Test.

beim Vergleichen von Float-Zahlen vorsichtig sein, zwei Werte für 17.50m müssen nicht unbedingt exakt übereinstimmen. Da der Vergleichsoperator `=` auf exakte Übereinstimmung testet, muss gegebenenfalls stattdessen ein Toleranzbereich überprüft werden, z.B.

```

if (kabinenposition > 17.49) and
    (kabinenposition < 17.51) then begin
  ...
end;

```

Das Anzeigeformat von `trealdisp` kann in der Eigenschaft `format` eingestellt werden. Für `posdisp` geben wir `0.00m` ein, die Nullen stehen für angezeigte Ziffern. Entsprechend definieren wir für `geschwdisp` `0.00m/s` und für `beschdisp` `0.00m/s^2` (**Bild 82**).

```

procedure visualisierung(schachtgeaendert: boolean);
begin
  mainfo.kabine.bounds_y:= round((schachthoehe - motor.kabinenposition -
                                kabinenhoehe) * massstab) + 1;
                                //1 pixel schacht frame
  motor.endschalterunten:= motor.kabinenposition <= 0.0;
  motor.endschalteroben:= motor.kabinenposition + kabinenhoehe >= schachthoehe;
  mainfo.posdisp.value:= motor.kabinenposition;
  mainfo.beschdisp.value:=
    (motor.sollgeschwindigkeit - mainfo.geschwdisp.value) * tickprosekunde;
  mainfo.geschwdisp.value:= motor.sollgeschwindigkeit;
  if schachtgeaendert then begin
    mainfo.schacht.invalidate(); //muss neu gezeichnet werden
    mainfo.bedienfeld.invalidate();
  end;
end;

```

Listing 44: Visualisierungs-Prozedur.

```

procedure tmainfo.tickexe(const sender: TObject);
var
  schachtgeaendert: boolean;
begin
  schachtgeaendert:= false; //provisorisch
  motortick(motor);
  visualisierung(schachtgeaendert);
end;

```

Listing 45: Tick-Prozedur

```

procedure tmainfo.createexe(const sender: TObject);
var
  i1: integer;
begin
  ...
  motor.kabinenposition:= 0; //zuunterst
  motor.sollgeschwindigkeit:= 0; //aus
  tick.interval:= 1000000 div tickprosekunde;
  tick.enabled:= true;
  motor.sollgeschwindigkeit:= 1.0; //1m/s aufwaerts als test
end;

```

Listing 46: Motor-Initialisierung.



Bild 82: Anzeigeformat.

- float-Variablen können Dezimalzahlen nicht immer exakt darstellen.

5.6. Motorregelung

Für einen komfortablen und sicheren Betrieb des Aufzuges müssen Geschwindigkeit und Beschleunigung begrenzt werden. Die Berechnung der aktuellen Sollgeschwindigkeit findet in **Listing 47** statt.

Zuerst wird berechnet, wie weit die Kabine bei sofortigem Abbremsen noch fährt. Falls die Kabine dabei das Ziel nicht erreicht, wird stattdessen beschleunigt.

In `tickexe()` fügen wir den Aufruf von `regelungtick()` ein (**Listing 48**).

`createexe()` wird geändert zu **Listing 49**.

Teste das Programm, die Kabine sollte sanft auf 4m/s beschleunigen und beim 8. Stockwerk sanft anhalten.

5.7. Steuerung

Die Steuerung der Anlage packen wir in die Prozedur `steuerung()` (**Listing 50**, **Listing 51**).

In `oeffnetuere()` halten wir den Zeitpunkt der Türöffnung in der Variable `tuereaufzeitpunkt` fest. Die Funktion `nowutc()` ("jetzt in UTC", UTC = koordinierte Weltzeit) liefert einen `tdatetime`-Wert. Der entsprechende `float`-Wert ist die Zeit in Tagen, die seit 1899-12-30 verstri-

chen ist. `nowutc()` stammt aus der `unit msedate`, darum muss `msedate` in `uses` aufgeführt werden.

implementation

uses

`main_mfm,sysutils,msedate;`

`nowlocal()` liefert den aktuellen Zeitpunkt in Lokalzeit. Wegen der Sommerzeitschaltung und möglicher Veränderungen der Zeitzone des Systems können wir `nowlocal()` zur Bestimmung der Türöffnungs-dauer nicht verwenden. `guibeep()` erzeugt ein akustisches Signal.

`aufzuginfo` muss zu **Listing 52** ergänzt werden.

Der Aufruf von `steuerung()` geschieht in `tickexe()` (**Listing 53**).

Die variable `schachtgeändert` enthält nun die Information, ob die Steuerung Änderungen vorgenommen hat und der Aufzug neu gezeichnet werden muss.

`createexe()` ändern wir zu **Listing 54**.

Teste ob der Aufzug richtig funktioniert, bevor du ihn zur Benutzung freigibst!

- `tdatetime` ist ein `float`-Wert der die Zeit in Tagen angibt.

5.8. Mängel

Unsere Aufzugssteuerung weist beträchtliche Mängel auf. Beispielsweise gibt es keine Anzeige des aktuellen Stockwerkes und kein Licht in der Kabine. Weiter ist die Auswahl des nächsten anzufahrenden Stockwerkes alles andere als optimal. Ich bin gespannt, ob du die Mängel beheben kannst. :-)

```

const
maximalgeschwindigkeit = 4.0; // m/s
anhaltegeschwindigkeit = 0.1; // m/s
maximalbeschleunigung = 1.0; // m/s^2

type
regelinfoty = record
  sollposition: float; // m
  geschwindigkeit: float; // m/s
end;

var
regelung: regelinfoty;

procedure regelungtick(var regelinfo: regelinfoty; var motorinfo: motorinfoty);
var
distanz: float; // m
anhaltezeit: float; // s
anhalteweg: float; // m
beschleunigung: float; // m/s^2
absgeschwindigkeit: float; // m/s absolutgeschwindigkeit, immer positiv
begin
distanz:= regelinfo.sollposition - motorinfo.kabinenposition;
absgeschwindigkeit:= abs(regelinfo.geschwindigkeit);
if (distanz > 0.01) or (distanz < -0.01) or
  (absgeschwindigkeit > anhaltegeschwindigkeit) then begin
  //position nicht erreicht oder zu schnell ->
  //neue geschwindigkeit berechnen
  anhaltezeit:= absgeschwindigkeit / maximalbeschleunigung;
  anhalteweg:= regelinfo.geschwindigkeit * anhaltezeit / 2.0;
  beschleunigung:= maximalbeschleunigung;
  if (anhalteweg >= distanz) xor (distanz < 0.0) then begin
    //bremsen notwendig?
    beschleunigung:= -maximalbeschleunigung; //ja, bremsen
  end;
  if distanz < 0 then begin
    beschleunigung:= -beschleunigung; //umgekehrte richtung
  end;
  regelinfo.geschwindigkeit:= regelinfo.geschwindigkeit +
    beschleunigung/tickprosekunde; //beschleunigen oder bremsen
  if regelinfo.geschwindigkeit > maximalgeschwindigkeit then begin
    //steiggeschwindigkeit begrenzen
    regelinfo.geschwindigkeit:= maximalgeschwindigkeit;
  end;
  if regelinfo.geschwindigkeit < -maximalgeschwindigkeit then begin
    //sinkgeschwindigkeit begrenzen
    regelinfo.geschwindigkeit:= -maximalgeschwindigkeit;
  end;
end
else begin
  regelinfo.geschwindigkeit:= 0.0; // position erreicht, stop
end;
motorinfo.sollgeschwindigkeit:= regelinfo.geschwindigkeit;
end;

```

Listing 47: Motorregelung.

```

procedure tmaininfo.tickexe(const sender: TObject);
var
  schachtgeaendert: boolean;
begin
  schachtgeaendert:= false; //provisorisch
  regelungtick(regelung,motor);
  motortick(motor);
  visualisierung(schachtgeaendert);
end;

```

Listing 48: Aufruf regelungtick().

```

motor.kabinnenposition:= 0.0; //zuunterst
motor.sollgeschwindigkeit:= 0.0; //aus
regelung.sollposition:= 0.0;
regelung.geschwindigkeit:= 0.0; //aus;
tick.interval:= 1000000 div tickprosekunde;
tick.enabled:= true;
regelung.sollposition:= 9*stockwerkhoehe; //8. stockwerk als test
end;

```

Listing 49: Initialisierung der Motorregelung.

```

const
  tuereaufzeit = 5.0/(24.0*60*60); //5s

function steuerung(var aufzug: aufzuginfoty;
                   var regelinfo: regelinfoty): boolean;
  //true wenn aenderungen vorgenommen wurden

procedure oeffnetuere(); //oeffnet tuere des zielstockwerkes
begin
  with aufzug do begin
    offenetuere:= zielstockwerk;
    tuereaufzeitpunkt:= nowutc();
    result:= not stockwerke[zielstockwerk].tuereauf; //aenderung vorgenommen
    if result then begin
      guibeep(); //akustisches signal
    end;
    stockwerke[zielstockwerk].tuereauf:= true;
  end;
end; //oeffnetuere

```

Listing 50: Anlagensteuerung 1. Teil.


```

var
  i1: integer;
begin
  result:= false; //init
  with aufzug do begin
    if gestartet then begin
      if (regelinfo.geschwindigkeit = 0) then begin //ziel erreicht
        anhaltestockwerk:= zielstockwerk;
        gestartet:= false;
        oeffnetuere();
        with stockwerke[zielstockwerk] do begin
          anforderungauf:= false;
          anforderungab:= false;
          ziel:= false;
        end;
        result:= true; //aenderung vorgenommen
      end;
    end
  else begin //nicht gestartet
    for i1:= 0 to stockwerkzahl-1 do begin
      with stockwerke[i1] do begin
        if anforderungauf or ziel then begin
          if not gestartet and (anhaltestockwerk = i1) then begin
            anforderungauf:= false; //keine anforderung notwendig, sofort ruecksetzen
            ziel:= false;
            oeffnetuere();
          end;
          zielstockwerk:= i1;
        end;
        if anforderungab or ziel then begin
          if not gestartet and (anhaltestockwerk = i1) then begin
            anforderungab:= false; //keine anforderung notwendig, sofort ruecksetzen
            ziel:= false;
            oeffnetuere();
          end;
          zielstockwerk:= i1;
        end;
        if (i1 = offnetuere) and tuereauf and
          (nowutc() - tuereaufzeitpunkt > tuereaufzeit) then begin
          tuereauf:= false;
          result:= true; //aenderungen vorgenommen
        end;
      end;
    end;
    if zielstockwerk <> zielstockwerkvorher then begin
      if not gestartet and not stockwerke[zielstockwerkvorher].tuereauf then begin
        //fahrt kann gestartet werden
        gestartet:= true;
        zielstockwerkvorher:= zielstockwerk;
        regelinfo.sollposition:= zielstockwerk * stockwerkhoehe;
      end;
    end;
  end;
end;
end;
end;
end;

```

Listing 51: Anlagensteuerung 2. Teil.

```

aufzuginfo = record
  stockwerke: array[0..stockwerkzahl-1] of stockwerkinfo;
  zielstockwerk: integer;
  zielstockwerkvorher: integer;
  anhaltstockwerk: integer;
  offenetuere: integer; //stockwerk mit offener tuer, -1 fuer keine
  tuereaufzeitpunkt: tdatetime; //zeitpunkt letzte tueroeffnung UTC
  gestartet: boolean;
end;

```

Listing 52: aufzuginfo

```

procedure tmaininfo.tickexe(const sender: TObject);
var
  schachtgeaendert: boolean;
begin
  schachtgeaendert:= steuerung(aufzug,regelung);
  regelungtick(regelung,motor);
  motortick(motor);
  visualisierung(schachtgeaendert);
end;

```

Listing 53: Aufruf der Steuerungsfunktion.

```

tick.interval:= 1000000 div tickprosekunde;
tick.enabled:= true;
aufzug.offenetuere:= -1; //alle tueren geschlossen
end;

```

Listing 54: Initialisierung der Anlagensteuerung.

Index

64 Bit-Computer, 17

A

anchors, 53
and, 23
application, 14
application.run, 42
application.terminated, 14
array, 34
Assembler, 47
Aufzählung, 59

B

Basis, 17
begin, 18
Bereichstyp, 49
Binärsystem, 17
Bit, 14
boolean, 23
bounds_, 36
Breakpoint, 31
'Button', 14
byte, 17

C

caption, 9, 10
cardinal, 19
case, 50
case, record, 56
Circular unit reference, 45
cl_default, 25
clientrect, 56
Compiler, 42, 47
Component Palette, 6
const, 27, 40
container, 19
CPU, 47

D

Daten, 14
Debugger, 9
div, 34
drawline(), tcanvas, 63
drawlines, 38

E

Einrückung, 18
ek_buttonpress, 61
else, 50
end, 18
Esc, 9
eventkindty, 59

F

F12, 6, 9
F9, 9, 31, 32, 45
face, 20
false, 14
Farbe, 15, 20
Fehlersuche, 31
float, 34
font, 21
for, 35
format, 68
frame, 10
function, 49
Funktion, 49

G

Gleitkommazahl, 33
Globale Variablen, 34
guibeeep(), 33, 70

H

Höhe anpassen, 11

halt(), 36
Haltepunkt, 31
high(), 35

I

implementation, 45
Index, 34
Instanzvariable, 45
integer, 19
interface, 45
interval, 33
invalidate(), 35

K

Kommentar, 14
Konstante, 27

M

main(), 63
mod, 36
mouseeventinfoty, 59
move(),tcanvas, 59
MSEgui, 6
MSEide, 5

N

Negative Zahlen, 19
not, 23
nowlocal(), 70
nowutc(), 70

O

Objektinspektor, 9
onafterpaint, 63
onclientmouseevent, 59
ondataentered, 50
onexecute, 33
onpaint, 36, 59
onsetvalue, 50
or, 23
ord(), 49
out, 40

P

Panel, 65

Parameterdefinition, 40
Pixel, 15
pointinrect(), 61
pointty, 38
program, 42
Programm abbrechen, 9
Programm beenden, 14
Programm starten, 9
Programm unterbrechen, 61
Projekt, 5, 42
properties, 6
Prozessor, 47

R

Rahmenbreite, 15
Rahmenhöhe, 15
random(), 36
rectty, 56
repeat, 53
result, 49
RGB, 20

S

Schriftart, 21
Schriftgrösse, 21
Speicherung, 14
Stack, 63
string, 47

T

tbutton, 14
tcanvas, 36
tdatetime, 70
Template, 5
terminated, 14
Text, 47
tfacecomp, 39
Tick, 67
tintegerdisp, 6
tintegeredit, 11
trealdisp, 55
true, 14
tstringdisp, 53
ttimer, 33
type, 35

U

Unicode, 48
unit, 45
until, 53
Ursprung, 38
uses, 45
UTC, 70

V

var, 18, 27, 40
Variablen, 18

W

Watches, 61
widget, 6
with, 56

X

xor, 24

Z

Zahlen, 16
Zeichenkette, 47
Zeichnungsursprung, 38

A. Installation der Entwicklungsumgebung

Die Installation der aktuellen MSEide+MSEgui Version ist in [4] beschrieben. Für die Beispiele wird mindestens die Version 3.2 benötigt. MSEide+MSEgui ist momentan für Linux 32 und 64 Bit sowie Windows 32 Bit erhältlich. Die Windows 32 Bit Version läuft auch problemlos auf Windows 64 Bit. Der verwendete Compiler ist Free Pascal [1].

1. Installiere Free Pascal Version 2.6.2 oder 2.6.4 [3], für Linux entweder die 32-Bit oder die 64-Bit Version, für Windows die 32-Bit Version.
2. Starte den Computer neu (nur für Windows).
3. Lade die MSEide+MSEgui Entwicklungsumgebung (die Datei mseide_msegui_src_VERSION.zip) herunter [5] .
4. Lade das MSEide Programm herunter, für Windows mseide_i386_win32_VERSION.zip, für 32-Bit Linux mseide_i386_linux_VERSION.tar.gz und für 64-Bit Linux mseide_x86_64_linux_VERSION.tar.gz.
5. Entpacke die Entwicklungsumgebung und MSEide in ein Verzeichnis deiner Wahl (DEINVERZEICHNIS).
6. Auf Linux starte DEINVERZEICHNIS/bin/mseide, auf Windows DEINVERZEICHNIS\bin\mseide.exe.
7. Klicke 'Settings'->'Configure MSEide', wähle in \${MSEDIR} DEINVERZEICHNIS/msegui.

B. Verweise

- [1] Free Pascal
<http://www.freepascal.org/>
- [2] Free Pascal Dokumentation
<http://www.freepascal.org/docs.var>
- [3] Free Pascal download
<http://www.freepascal.org/download.var>
- [4] MSEide+MSEgui
<http://mseide-msegui.sourceforge.net/>
- [5] MSEide+MSEgui download
<http://sourceforge.net/projects/mseide-msegui/files/mseide-msegui/>
- [6] Unicode Consortium
<http://www.unicode.org/>