KnowWare Special!

5,20 Deutschland

C++

leicht & verständlich

Hans-Georg Schumann

www.KnowWare.de

Deutschland: 5,20 EUR Österreich: 6,- EUR
Schweiz: 10,- SFR Luxemburg: 6,00 EUR Italien: 7,- EUR

4 195088 005200 05008

Programmieren lernen mit dem Borland C++Builder 6

KnowWare Special!

Bildqualität

Wir versuchen die Dateigröße zu reduzieren, um die Downloadzeit zu verkürzen. Daher ist die Bildqualität in dieser Download-Datei nicht in allen Fällen optimal. Im Druck tritt dieses Problem nicht auf.

Acrobat Reader: Wie komme ich klar?

F5/F6 öffnet/schließt die Ansicht **Lesezeichen**
Strg+F sucht

Im Menü Ansicht stellst du ein, wie die Datei angezeigt wird
STRG+0 = Ganze Seite **STRG+1** = Originalgröße **STRG+2** = Fensterbreite

Im selben Menü kannst du folgendes einstellen: **Einzelne Seite**, **Fortlaufend** oder **Fortlaufend - Doppelseiten** ... Probiere es aus, um die Unterschiede zu sehen.

Navigation
Pfeil Links/Rechts: eine Seite vor/zurück
Alt+ Pfeil Links/Rechts: Wie im Browser: Vorwärts/Zurück
Strg+**+** vergrößert und **Strg**+**–** verkleinert

Bestellung und Vertrieb für den Buchhandel

KnowWare-Vertrieb, Postfach 3920, D-49029 Osnabrück
Tel.: +49 (0)541 33145-20 Fax: +49 (0)541 33145-33
bestellung@knowware.de
www.knowware.de

Autoren gesucht

Der KnowWare-Verlag sucht ständig neue Autoren. Hast du ein Thema, dass dir unter den Fingern brennt? - ein Thema, das du anderen Leuten leicht verständlich erklären kannst?

Schicke uns einfach ein paar Beispielseiten und ein vorläufiges Inhaltsverzeichnis an folgende Adresse:

lektorat@knowware.de

Wir werden uns deinen Vorschlag ansehen und dir so schnell wie möglich eine Antwort senden.

www.knowware.de

Inhaltsverzeichnis

Vorwort

Von C++ zum C++Builder

In diesem Kurs geht es ums Programmieren. Und obwohl es eine ganze Reihe verschiedener Programmiersprachen gibt, zieht es viele doch immer wieder zu C++. Jede Sprache hat ihre Vor- und Nachteile, und du wirst immer jemanden finden, der die eine ganz besonders lobt, während die Übrigen bei ihm nicht allzu gut abschneiden.

C++ gilt nicht zu Unrecht als „Alleskönner“ in Sachen Programmierung. Ursprünglich hatte sie sogar den schlichten Namen C (gesprochen wie der Buchstabe, also „Zeh“).

Diese Programmiersprache verbreitete sich schnell, vor allem im Profibereich. Im Laufe der Jahre kamen eine Menge mächtiger Elemente hinzu, sodass schließlich auch der Name dieser Entwicklung Rechnung tragen musste. Und so wurde aus C eine neue Sprache namens C++.

Auch nach dem Siegeszug von Windows wurde C++ schnell angepasst. Vor allem die Firmen Microsoft und Borland boten Programmiersysteme an, die heute weit verbreitet sind.

Beide haben ihre Vor- und Nachteile und sind in verschiedenen Versionen für gutes Geld zu bekommen.

Dabei handelt es sich jeweils um eine komplette Entwicklungsumgebung: Es gibt einen Editor, in dem du den Programmtext eintippen und bearbeiten kannst. Es gibt einen Compiler, der deinen Text so übersetzt, dass er für den Computer verständlich und als Programm ausführbar wird. Und es gibt eine Reihe von Hilfsprogrammen, die dich bei der Fehlersuche unterstützen – denn Fehler sind beim Programmieren alltäglich.

Schauen wir uns kurz die „Konkurrenz“ von C++ an:

Sehr bekannt ist *Visual Basic*. Dieses Entwicklungssystem ist nicht allzu schwer zu erlernen und zu bedienen. Allerdings reicht die Leistungsfähigkeit nicht an die von C++ heran.

Auch *Java* ist leichter zu erlernen als C++, ist ihm jedoch in seiner Leistungsfähigkeit ebenfalls unterlegen.

Pascal bzw. *Delphi* ist in seinen Leistungen mit C++ vergleichbar, hat aber nie dieselbe Verbreitung erreicht.

Nun wird es Zeit, endlich unser System zu starten und mit dem Programmieren zu beginnen.

Du hast noch kein C++?

Möglicherweise bist du Schüler oder Student oder sonst wie „Lernender“. Dann gibt es bei vielen Software-Anbietern einen Sonderpreis für C++-Software.

Es muss ja nicht unbedingt die neueste Version sein. Nicht selten bekommt man eine ältere zum Schleuderpreis.

Auf der Homepage von Borland gibt es sogar Trial-Versionen von C++Builder 6 oder C++BuilderX für 30 Tage kostenlos zum Ausprobieren, die du dir hier herunterladen kannst:

http://www.borland.com/products/downloads/download_cbuilder.html

oder

http://www.borland.com/products/downloads/download_cbuilderx.html

Allerdings kann das dauern, denn es handelt sich um immerhin mindestens 170 MB!

Sämtliche Beispiele in diesem Heft wurden mit dem Borland C++Builder 6 erstellt, lassen sich also nicht „einfach so“ auf das Konkurrenzprodukt Microsoft Visual C++ übertragen. Es empfiehlt sich also hier auf jeden Fall eine Version des C++Builder (es kann auch eine ältere sein).

Die Beispiele sowie die Lösungen zu den Aufgaben findest du unter der Adresse

http://www.knowware.de/?book=cbuilder.

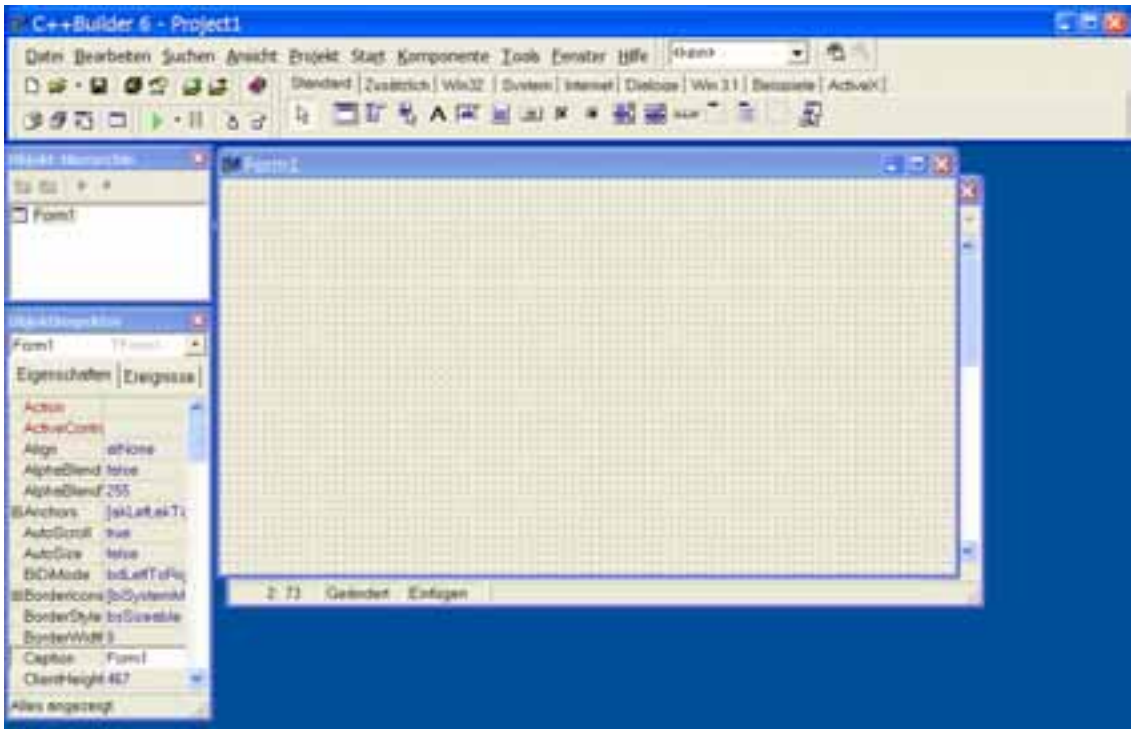
Viel Spaß und Erfolg mit C++ wünscht

Hans-Georg Schumann

Auf zur C++Builder-Tour!

Ich gehe davon aus, dass du weißt, wie man unter Windows ein Programm installiert und wie man es startet. Auch solltest du wissen, wie man bei Windows-Programmen Menüs öffnet und Dialogfelder bedient, um z. B. Dateien zu speichern oder zu laden.

Nachdem der C++Builder in den Startlöchern steht, kann es losgehen – es erscheint dieses oder ein ähnliches Bild:



Ganz oben siehst du die Hauptleiste. Hier tummeln sich Menüeinträge und Symbole – wie bei anderen Windowsprogrammen auch.

Ein Formular

Darunter finden wir eine Art Arbeitsplatte wie in der Küche beim Zubereiten von Speisen; hier heißt diese Platte *Formular* oder *Formfenster*.

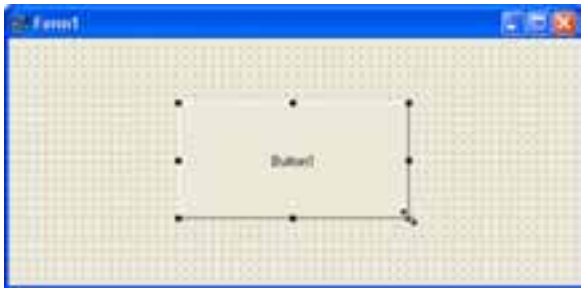
Links davon steht etwas von *Objekt-Hierarchie* und *Objektinspektor*. Mit einem dieser beiden Hilfsfenster befassen wir uns etwas später – das andere benötigen wir hier nicht unbedingt.

Das erste kleine Projekt

Jetzt aber wollen wir gleich unser erstes kleines Projekt erstellen.

- Suche in der Symbolleiste das Symbol für BUTTON und klicke darauf.

- Im Formfenster drückst du die linke Maustaste und ziehst damit einen Rahmen auf, der die Größe deines Buttons festlegt. Seine Lage und Maße lassen sich jederzeit ändern.



Statt Button spricht man im Deutschen auch von Schaltfläche, wie du von deiner Erfahrung mit Windows weißt.

Nun haben wir eine Art Knopf (zum Drücken oder Klicken) in einem Formular – immerhin. Mich stören zunächst die Bezeichnungen: Statt „Form1“ und „Button1“ könnte man da doch etwas anderes hinsetzen?

Objekte

Und nun kommt der *Objektinspektor* zum Zug.

Wie kommt es zu diesem Namen? Unter einem *Objekt* versteht man normalerweise Dinge, die sich in unserer Umgebung befinden: Häuser, Autos, aber auch Menschen, Tiere, Pflanzen. Objekte haben Eigenschaften und Verhaltensweisen.

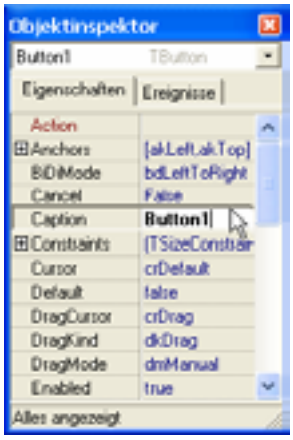
Auch in C++ gibt es Objekte – dort spricht man neben Eigenschaften aber von Methoden.

Der Objektinspektor ist für die Verwaltung von Eigenschaften und Methoden eines Objektes zuständig. Und du vermutest richtig, dass Komponenten wie Formulare und Schaltflächen Objekte sind.

Weil es in C++ – wie im richtigen Leben – mehrere Objekte ein und desselben Typs geben kann, fasst man dort alle Gemeinsamkeiten unter dem Begriff *Klasse* zusammen.

Der Objektinspektor zeigt die Eigenschaften der jeweils markierten Komponente an.

Eine Komponente in Windows – das ist z. B. ein Fenster, ein Menü, eine Schaltfläche usw. Und C++Builder bietet die Möglichkeit, alle diese Komponenten in einem Programm einzusetzen.



Zu unserem Projekt gehören augenblicklich zwei Komponenten: Ein Formular mit dem Namen `Form1` und eine Schaltfläche namens `Button1`.

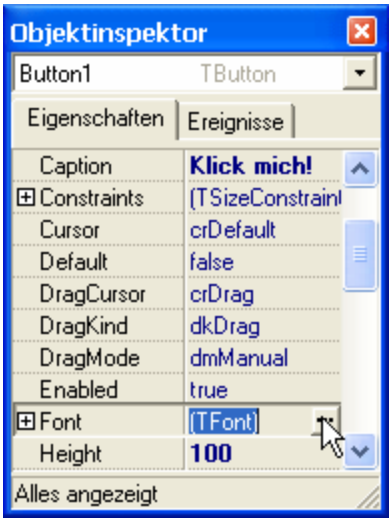
Eine Aufschrift für den Button

Normalerweise müsste die Schaltfläche noch markiert sein – ist sie das nicht, hilft ein Mausklick.

1. Klicke auf den Text hinter CAPTION und lösche ihn.
2. Gib einen neuen Text ein, z.B. „Klick mich!“

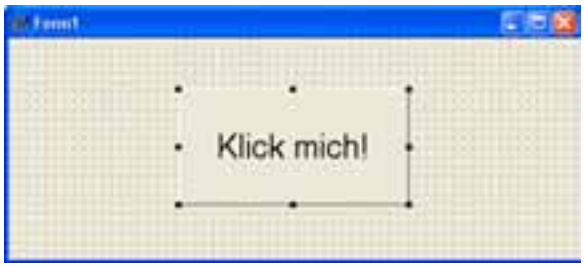
Und siehe da: Der Button hat eine neue Aufschrift, die allerdings noch ein bisschen mickrig aussieht.

3. Deshalb suchst du im Objektinspektor die Eigenschaft FONT, unter der du u.a. Schriftart und Schriftgröße einstellen kannst.



4. Hier wählst du unter SCHRIFTGRAD eine Größe um die 20 aus.

Dann dürfte das Ganze in etwa so aussehen:



Ganz nett – aber wo ist nun das Programm? Und was bringt es, wenn ich auf diesen Button klicken würde?

Ehrlich gesagt, noch gar nichts. Bis jetzt besteht das Projekt auch nur aus einem Formular mit einer Schaltfläche.

Deshalb müssen wir jetzt klären, was passieren soll, wenn auf diesen Button geklickt wird.

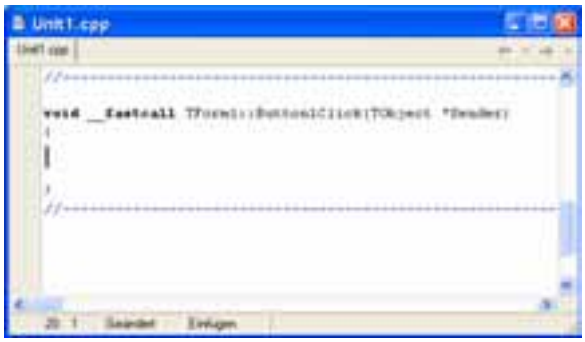
Beginnen wir einfach: Auf Mausklick soll der Button seine Aufschrift ändern, z. B. in „Autsch!“ oder – noch eindrucksvoller – in „AUTSCH!“.

Ereignis und Methode

Und damit landen wir bei den Methoden. C++Builder bietet für den Button eine ganze Reihe von Methoden. Wir benötigen eine, die auf Mausklick reagiert.

- Dazu doppelklickst du einfach auf den Button.

Und schon landest du in einem neuen Fenster, das sich die ganze Zeit hinter dem Formular versteckt hat. In diesem Fenster findest du den *Editor*, in den du den Programmtext eingeben kannst. Ein Profi – der du ja werden willst – sagt übrigens statt Programmtext auch *Quelltext*.



Im Editorfenster steht der Doppelname `TForm1::Button1Click`. Zuerst kommt der Name des Objekttyps bzw. der Klasse, zu der die eigentliche Methode gehört: `TForm1`. Der folgende Methodenname erklärt sich dann selbst: `Button1Click`.

Die Zahlen darin besagen einfach nur, dass C++ natürlich auch mehr als ein Formular und einen Button zu bieten hat – wenn nötig.

Der doppelte Doppelpunkt ist wichtig. Er verknüpft beide Teile, den Klassennamen und den Methodennamen. Weil die Methode für alle Objekte gleich ist, wird sie unter dem Klassennamen vereinbart. Anders sieht es bei den Eigenschaften aus, denn diese können natürlich von Objekt zu Objekt verschieden sein – z. B. die Größe.

Da ist noch etwas – zwei vorangestellte Wörter: Das erste Wort (`void`) erklärt den Typ der Methode (wir kommen darauf zurück, wenn auch recht spät). Beim zweiten Begriff (`__fastcall`) handelt es sich um ein Wort, über das C++Builder diese Methode intern handhabt.

Genug geredet!

- Vollende die Methode, indem du sie so ergänzt:

```
void __fastcall ↵
 TForm1::Button1Click (TObject ↵
 *Sender)
{
  Button1->Caption = "AUTSCH!";
}
```

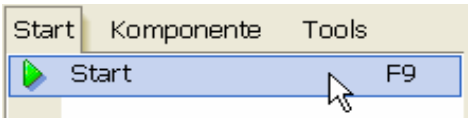
Die geschweiften Klammern markieren in C++ den Anfang und das Ende eines Anweisungsblocks. Darin kann nur eine Anweisung enthalten sein – wie oben –, in der Regel sind es aber mehrere.

Hinweis zu den Quelltexten: Weil's hier ziemlich eng zugeht, passt nicht immer alles in eine Zeile. Das macht aber nichts, denn in C++ wird nicht zeilenorientiert gearbeitet.

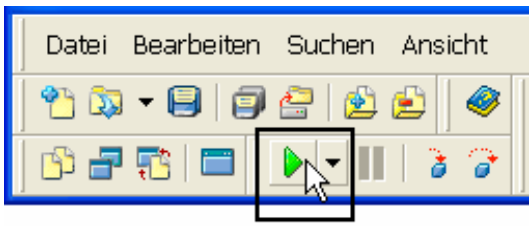
Dieses Zeichen (↵) bedeutet, dass die Zeile weitergeht und nicht umbrochen wird. Drücke also *nicht* auf Enter ! Wenn vor dem Pfeil zusätzlich ein Leerzeichen steht (↵), tippst du das Leerzeichen mit. Falls dort jedoch kein Leerzeichen eingefügt wurde, setzt du die Zeile ohne Leerzeichen fort.

Jetzt soll der Computer das Projekt ausführen.

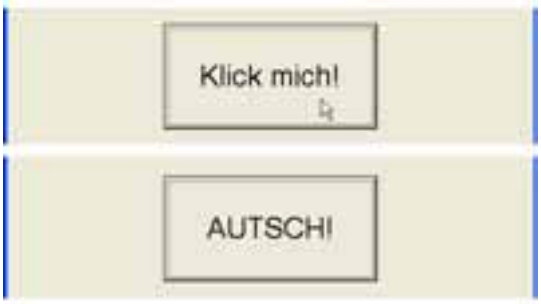
- Dazu klickst du im Menü auf START und nochmals auf START oder drückst F9 .



Eine weitere Möglichkeit ist ein Klick auf das Start-Symbol unter der Menüleiste:



Das Formular hat seine Musterung verloren. Und wenn du jetzt auf unseren Button klickst, ändert sich seine Aufschrift:



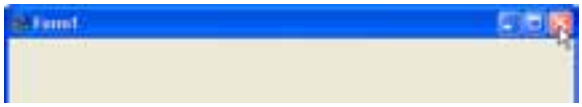
Immerhin: Es ist etwas passiert. Dazu brauchten wir außer einem Button auch nur eine einzige (zusätzliche) Programmzeile.

Wir haben es hier mit einer Kombination von Ereignis und Methode zu tun: Das *Ereignis* heißt Mausklick (bzw. `OnClick`), und es hat die dazugehörige *Methode* `Button1Click` ausgelöst bzw. aktiviert.

Wechselhaft

Bevor es weitergeht, müssen wir dieses Programm beenden, um zu unserem C++-Projekt zurückzukehren.

- Klicke dazu auf die Schaltfläche mit dem kleinen X – wie du es auch von anderen Windowsprogrammen gewöhnt bist.



Nun widmen wir uns der Zeile, um die wir die Methode `Button1Click` erweitert haben:

```
Button1->Caption = "AUTSCH!";
```

Hier taucht ein neuer Verknüpfungsoperator zwischen zwei Namen auf, eine Art Pfeil (dargestellt durch ein Minus- und ein Größerzeichen): Das Objekt, um das es geht, heißt `Button1`, die Eigenschaft heißt `Caption`, und gemeint ist damit so etwas wie „Aufschrift“.

Dieser wird der Text „AUTSCH!“ zugewiesen. Und dafür ist der Zuweisungsoperator = da: ein einfaches Gleichheitszeichen.

Man spricht deshalb auch von einer *Zuweisung*. Weitere Zuweisungen könnten sein:

```
X = 5;
Name = "Erich";
```

Hier bekommt eine Variable namens *X* (die du ja aus dem Matheunterricht kennst) den Wert 5 zugewiesen. Mit Text geht das natürlich auch. Im zweiten Beispiel heißt die Variable *Name* und der zugewiesene Wert ist `"Erich"`.

Es ist übrigens nicht gleichgültig, ob du hier z. B. ein kleines oder ein großes X benutzt. In C++ wird streng zwischen Groß- und Kleinschreibung unterschieden!
Dies gilt natürlich auch für Objekte und Methoden: `Button1Click` ist ein anderer Methodenname als `button1click`.

Mich stört bei dem bisherigen Projekt, dass der Button nach dem Klicken die Aufschrift „Autsch!“ behält. Schöner wäre es, würde nach einer kleinen Weile wieder der alte Text „Klick mich!“ erscheinen.

Damit kommen wir zu unserer nächsten Erweiterung:

```
void __fastcall
 TForm1::Button1Click (TObject
 *Sender)
{
  Button1->Caption = "AUTSCH!";
  Sleep (1000);
  Button1->Caption = "Klick mich!";
}
```

Warum gibt es zum einen den doppelten Doppelpunkt (::), zum anderen den Pfeil (->) als Verknüpfungsoperator? Im ersten Fall wird die Methode eines Objekts vereinbart (hier mit `TForm1::Button1Click`), im zweiten Fall wird eine Anweisung gegeben, die ausgeführt werden soll (hier z. B. mit `Button1->Caption = "AUTSCH!"`)

Nun wird nach einem Mausklick auf die Schaltfläche für eine Sekunde ein klagendes „Autsch!“ angezeigt, um dann wieder der ursprünglichen Aufforderung „Klick mich!“ Platz zu machen.

Dafür, dass der zweite Text eine Sekunde Zeit hat, sorgt die Anweisung `Sleep (1000)`, die die Programmausführung für 1000 Millisekunden „schlafen legt“.

Ein Projekt speichern

Wenn du das Projekt lange genug ausprobiert hast, solltest du es sicher auf der Festplatte unterbringen.

Das geschieht am besten über DATEI|ALLES SPEICHERN.

Da es vermutlich dein erstes Projekt ist, kannst du die vorgegebenen Namen übernehmen:

PROJECT1.BPR	Die Projektdatei mit den Verwaltungsdaten (BPR = Borland PRojekt)
UNIT1.CPP	Die Datei mit dem Quelltext (CPP = C++ bzw. „Zehplusplus“), auch **Unit** genannt.

Hinzu kommen weitere Dateien, die C++Builder automatisch erstellt, wie z. B. UNIT1.DFM mit den Formulardaten (DFM = Delphi ForMular; das Format wurde von Delphi übernommen).

Alle im Heft behandelten Beispiele wie auch die Lösungen zu den Aufgaben kannst du dir von der KnowWare-Homepage herunterladen. Du findest sie unter der Adresse http://www.knowware.de/?book=cbuilder

Reicht dir das fürs Erste, kannst du C++Builder jetzt wie jedes normale Windowsprogramm beenden.

Aufgabe 1

Wenn du es dir zutraust, versuch doch mal den Button „hüpfen“ zu lassen. Dazu müssen sich in `Button1Click` nur statt der oder zusätzlich zur Eigenschaft CAPTION die Werte für LEFT und TOP ändern.

Mathematik

Und schon geht es weiter mit unserem nächsten Projekt (RECHNER1). Diesmal ist etwas Mathe angesagt. Aber keine Bange – nur die Grundrechenarten sind gefragt, Grundschulbildung genügt also – noch.

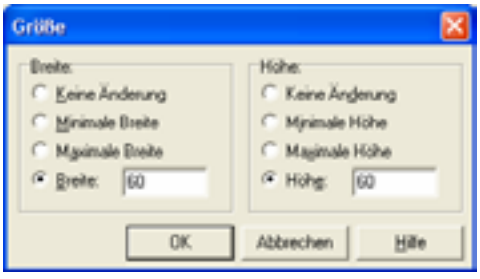
- Klicke auf das Symbol für Button und ziehe vier gleich große Schaltflächen im Formular auf.

Die Maße kannst du jeweils von Hand im Objektinspektor unter den Eigenschaften HEIGHT und WIDTH eintragen (z. B. einen Wert zwischen 60 und 90), die Position wird unter TOP und LEFT festgelegt. Die folgende Tabelle verschafft dir den nötigen Überblick:

WIDTH	Breite der Komponente
HEIGHT	Höhe der Komponente
LEFT	Position der Komponente links
TOP	Position der Komponente oben

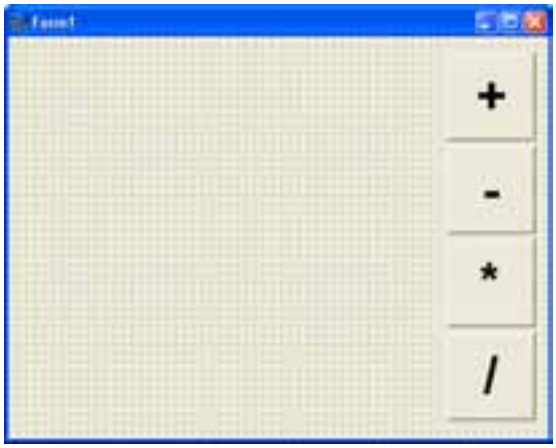
Wenn du mit der rechten Maustaste auf eine Komponente klickst, öffnest du ein Kontextmenü; über die Einträge POSITION|AUSRICHTEN erreichst du ein Dialogfeld, in dem du deine Buttons „in Position“ bringen kannst, wenn du sie zuvor alle zusammen markiert hast.

Über POSITION|GRÖßE im selben Kontextmenü kannst du auch die Maße für alle Buttons gleichzeitig festlegen.



- Bist du mit der Kosmetik fertig, bezeichnest du über CAPTION jeden Button mit einem Symbol für die vier Grundrechenarten (+, -, *, /)

Dann dürfte das ganze Formular etwa so aussehen:



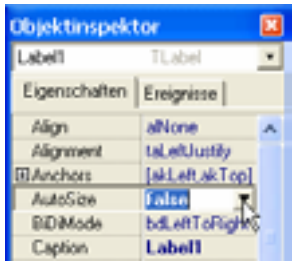
Natürlich habe ich da bei der Eigenschaft FONT noch etwas nachgeholfen, damit alle Symbole auch groß genug sind.

Eine neue Komponente

Könnte eine Art Taschenrechner werden? Aber etwas Wichtiges fehlt: Die Anzeige. Dazu benötigen wir eine neue Komponente, ein so genanntes Label, auch Anzeigefeld genannt:

- Suche in der Symbolleiste das Symbol für LABEL und klicke darauf. Dann ziehe im Formular links neben der Buttongruppe ein Label auf.

Bevor du nun den Eintrag hinter CAPTION änderst, müssen wir uns zuerst um eine andere Eigenschaft kümmern.



- Stelle hinter AUTOSIZE den Wert auf FALSE. So wird verhindert, dass sich die von dir gewählte Größe der Anzeigefläche (automatisch) ändert und dem angezeigten Wert anpasst.

Wir brauchen – wie beim Taschenrechner – immer dieselbe Anzeigefläche.

- Der Anzeigetext „Label1“ ist natürlich blöde – also solltest du ihn ganz löschen, oder du setzt dort z. B. „???“ oder lauter Nullen ein.
- Wichtig ist, dass du über FONT eine ausreichend große Schrift für das Anzeigefeld einstellst.

Weil wir uns nun schon so etwas wie einen Rechner zusammengebaut haben, sollte der Titel „Form1“ verschwinden:

- Dazu klickst du auf eine freie Fläche im Formular und gibst dann unter CAPTION den Text „Kleiner Rechner“ oder „Grundrechenarten“ oder eine Bezeichnung deiner Wahl ein.
- Es kann nicht schaden, ruhig schon mal einen Programmstart zu wagen (z. B. mit Klick auf das Start-Symbol oder F9).

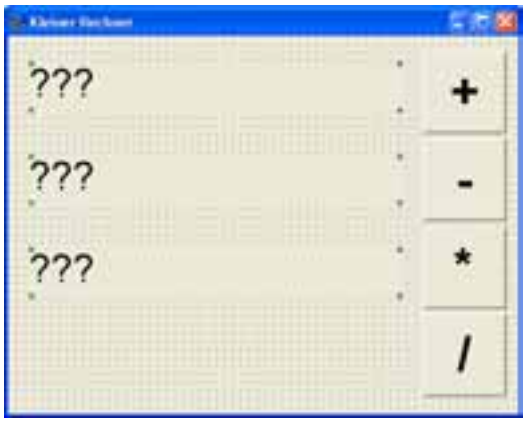
Dann könnte das Formular etwa so aussehen:

Wenn du jetzt auf die verschiedenen Buttons klickst, reagiert das Programm allerdings leider überhaupt noch nicht darauf – es gibt ja auch noch keine passende Methode.

Mehr Anzeigefelder

Außerdem haben wir gar kein Tastenfeld für die Zahlen – wie bei einem echten Taschenrechner. Es gäbe also schon noch einiges zu tun, wenn wir dieses „Gerät“ perfektionieren wollten.

- Zuerst beendest du den Programmtest mit Klick auf das kleine X oben rechts.
- Erweitere jetzt die Anzeige um zwei zusätzliche Felder, die genau so groß sind wie das vorhergehende.
- Stelle auch dort die Eigenschaft AUTOSIZE auf FALSE und über FONT die Schrift entsprechend groß ein.



In den beiden oberen Anzeigefeldern sollen nun zwei zufällige Zahlen erscheinen. Und mit Klick auf einen der Rechenbuttons steht ganz unten das Ergebnis der jeweiligen Rechnung.

Damit das ganze Spielchen starten kann, sollten wir uns noch einen Startknopf zulegen, den wir „Neu“ oder „Start“ benennen.

- Ergänze dein Projekt noch um diese Schaltfläche, die du unter den Labels positionierst.

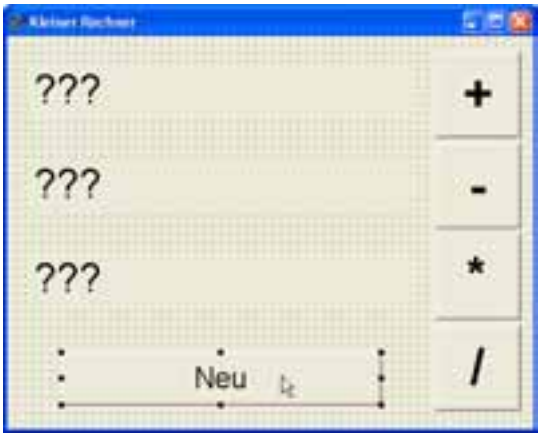
Und nun ist es an der Zeit, das neue Projekt zu speichern.

Das geht nicht nur – wie bei Windowsprogrammen üblich – über das DATEI-Menü, sondern es gibt auch ein praktisches ALLES SPEICHERN-Symbol.

Als Namen könntest du z. B. PROJECT2.BPR und UNIT2.CPP benutzen. Ich habe mich für RECHNER1.BPR und MATHE1.CPP entschieden.

Nachdem unsere bisherige Arbeit erst mal sicher auf Festplatte ruht, kann es weitergehen:

- Doppelklicke auf den Button mit der Aufschrift „Neu“.



Schon landest du wieder im Editor von C++Builder.

Zufallszahlen

Dass die Methode nun `Button5Click` heißt, dürfte für dich keine Überraschung sein, denn es gibt ja außer dem NEU-Knopf vier weitere.

Was soll in dieser Methode zwischen den beiden geschweiften Klammern `{ }` stehen?

Schauen wir's uns an:

```
void __fastcall
 TForm1::Button5Click (TObject
 *Sender)
{
  randomize ();
  Zahl1 = random (1000);
  Zahl2 = random (1000);
  Label1->Caption = IntToStr
   (Zahl1);
  Label2->Caption = IntToStr
   (Zahl2);
}
```

Schälen wir das Innere der Methode heraus und nehmen wir uns dann jede Zeile einzeln vor:

```
  randomize ();
```

Diese Methode – auch Prozedur genannt – startet den *Zufallsgenerator* des C++Builder – ein ausführlicherer Name ist „Zufallszahlengenerator".

Wozu die leeren Klammern?
In C++ werden alle Methoden (Prozeduren und Funktionen) mit Klammern versehen – auch wenn diese mal leer sind.

Nun ist es möglich, nach einer C++-internen Formel zufällige Zahlen zu erzeugen. Wir müssen nur noch die entsprechende Methode einsetzen, die den Bereich bzw. das Intervall angibt, in dem diese Zahlen sich befinden sollen:

```
  Zahl1 = random (1000);
  Zahl2 = random (1000);
```

Der Wert hinter `random` gibt die Obergrenze an, was konkret bedeutet: In Frage kommen alle ganzen Zahlen zwischen 0 und 999 (also nicht die 1000!).

Damit du beide Werte auch zu sehen bekommst, müssen wir sie der Eigenschaft CAPTION der beiden ersten Anzeigefelder (Labels) zuweisen.

Das wird mit diesen Zeilen erledigt:

```
  Label1->Caption = IntToStr
   (Zahl1);
  Label2->Caption = IntToStr
   (Zahl2);
```

Doch halt! Da hat sich doch noch was dazwischengeschmuggelt: Die Methode `IntToStr` – auch Funktion genannt.

Wir können hier nämlich nicht einfach eine Zahl zuweisen, denn CAPTION erwartet immer einen Text.

Der Name „IntToStr" heißt eigentlich „Integer To String", zu Deutsch, frei übersetzt: „Ganze Zahl in Zeichenkette (verwandeln)". In Klammern übernimmt diese Methode also eine Zahl und wandelt sie in eine Zeichenkette um.

Bei der Programmierung spricht man von Zeichenkette oder auch direkt von *String* an Stelle von Text, denn eine solche „Kette" kann außer Buchstaben auch beliebige andere Zeichen beinhalten.

Prozedur oder Funktion? Jede Anweisung oder Methode in einer Programmiersprache ist zunächst mal eine *Prozedur*, Beispiel `randomize`.

Gibt sie einen Wert zurück, dann spricht man auch von *Funktion*, Beispiel `random`. Wird eine Methode nur aufgerufen, ist sie in C++ eine Prozedur; wird sie z. B. zugewiesen, dann handelt es sich um eine Funktion.

Prozeduren und Funktionen können Werte übernehmen, die *Parameter* genannt werden. So ist z. B. bei `random (1000)` die Zahl 1000 ein Parameter.

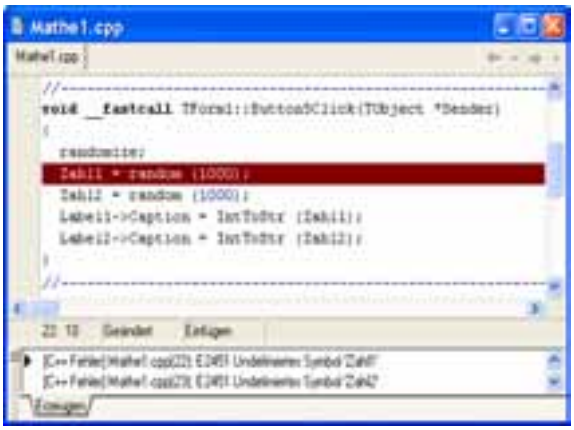
Nebenbei bemerkt übernehmen offenbar alle bisherigen `ButtonClick`-Methoden einen Parameter namens `Sender`, der für eine Art „Nachrichtenübermittlung“ zuständig ist.

Vereinbarungen

Mit `Zahl1` und `Zahl2` verwenden wir zwei Variablen, in denen die Zufallswerte gespeichert werden.

Variablen kennst du aus dem Matheunterricht als so genannte Platzhalter. In C++ müssen Variablen stets vereinbart werden, wie wir gleich sehen.

Wenn du jetzt versuchst, das Projekt in seiner bisherigen Form nochmals laufen zu lassen, gibt es ein paar saftige Fehlermeldungen:



Genau genommen sind es zwei Fehler – sie betreffen die eben genannten Variablen, in C++ auch „Symbol“ oder „Bezeichner“ genannt.

Das Programm kann also nicht starten, weil diese zwei Variablen unbekannt sind.

In C++ (wie in vielen anderen Programmiersprachen) müssen Variablen vor ihrer Verwendung vereinbart werden. Und das geschieht so:

- Blättere im Quelltext nach oben und suche dort diese Zeile
  `TForm1 *Form1;`

Wie du sehen kannst, ist schon etwas mit `Form1` vereinbart. Das ist das Formular, das für C++ offenbar auch eine Variable ist (variabel heißt ja „veränderlich“). Die Variable `Form1` ist offenbar recht umfangreich und besitzt natürlich sehr viel mehr Werte als etwa so „kleine“ Variablen wie `Zahl1` und `Zahl2`. (Auf das Sternchen, wie du es vor `Form1` siehst, kommen wir erst später zu sprechen.)

Eine Variablenvereinbarung geschieht immer in dieser Form:

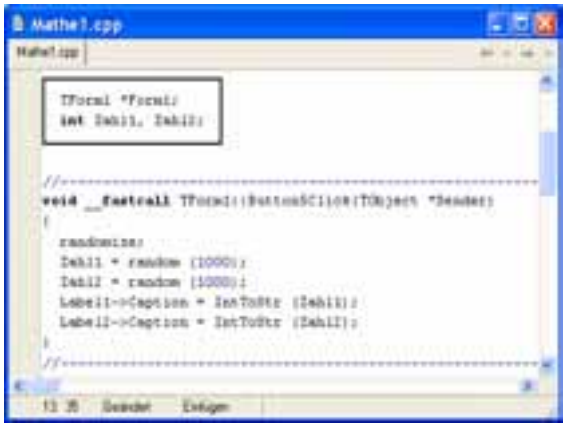
```
Typ Name;
```

Und so vereinbaren wir direkt darunter unsere eigenen Variablen:

```
int Zahl1, Zahl2;
```

`int` bezeichnet Variablen vom Typ *Ganze Zahl*.

- Ergänze deinen Quelltext entsprechend um die Vereinbarung von `Zahl1` und `Zahl2`.



- Dann probiere das Programm aus und starte es.

Jedes Mal, wenn du auf NEU klickst, erscheinen in den beiden oberen Anzeigefeldern zwei zufällige ganze Zahlen (unter 1000).

Die unterste Anzeige behält jedoch ihren ursprünglichen Text bei.

Darum müssen wir uns gleich kümmern, sobald du das Programm wieder beendet hast und zum C++-Formular zurückgekehrt bist.

- Nun doppelklicke auf den obersten Button mit dem Plus.

Das Ergebnis? Wir befinden uns in der (noch leeren) Methode `Button1Click`.

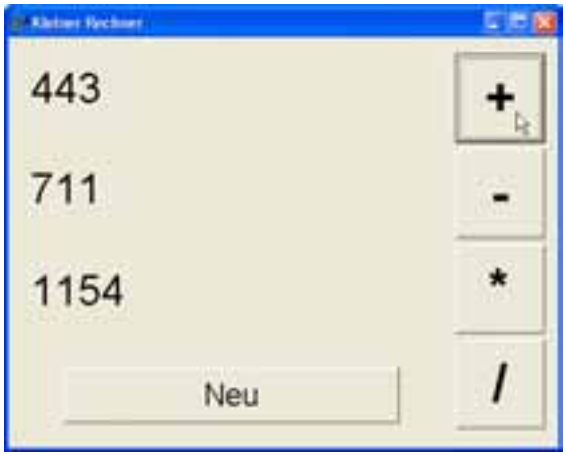
- Gib dort folgende Zeilen ein:

```
void __fastcall
 TForm1::Button1Click (TObject
 *Sender)
{
  Label3->Caption =
   IntToStr (Zahl1 + Zahl2);
}
```

Hier werden zuerst die Werte der Variablen `Zahl1` und `Zahl2` addiert und das Ergebnis in eine Zeichenkette umgewandelt. Anschließend wird dieser String der Eigenschaft CAPTION des dritten Anzeigefeldes zugewiesen.

- Probiere das Programm gleich aus!

Nun klappt es immerhin schon, wenn man nach einem Klick auf NEU hinterher den Plus-Button aktiviert. So könnte es dann aussehen (siehe Nachbarspalte oben):



Plus, Minus, Mal, Durch

Eigentlich dürfte es für dich nun ein Leichtes sein, den anderen Methoden auch ihre Rechnung zu verpassen.

- Doppelklicke auf jeden der anderen drei Rechenbuttons und ergänze die zugehörige Methode um die entsprechende Zuweisung.

Für alle Fälle findest du hier den kompletten Quelltext für alle Methoden:

```
void __fastcall TForm1::Button1Click (TObject *Sender)
{
  Label3->Caption = IntToStr (Zahl1 + Zahl2);
}
//---------------------------------------------------------------------
void __fastcall TForm1::Button2Click (TObject *Sender)
{
  Label3->Caption = IntToStr (Zahl1 - Zahl2);
}
//---------------------------------------------------------------------
void __fastcall TForm1::Button3Click (TObject *Sender)
{
  Label3->Caption = IntToStr (Zahl1 * Zahl2);
}
//---------------------------------------------------------------------
void __fastcall TForm1::Button4Click (TObject *Sender)
{
  Label3->Caption = IntToStr (Zahl1 / Zahl2);
}
```

Die gestrichelten Linien zwischen den einzelnen Methoden dienen der optischen Abgrenzung (und werden hier von C++Builder automatisch erzeugt). Sie gehören nicht zum eigentlichen Programmtext. Damit auch der Computer das weiß, werden am Anfang so genannte Kommentarzeichen (///) gesetzt.

Statt der Linien kannst du also dort auch beschreiben, was gerade im Programm hier passiert. Entdeckt der C++-Compiler zwei solche Schrägstriche, dann weiß er: Das geht mich nichts an, da hat der Programmierer seine eigenen Notizen gemacht. **Kommentare** erhöhen die Übersichtlichkeit und Lesbarkeit eines Quelltextes.

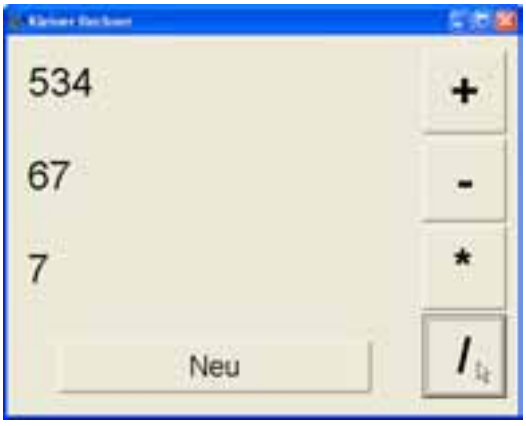
Bevor es weitergeht, muss dieses Projekt unbedingt noch einmal gespeichert werden – immerhin sind wir schon recht weit fortgeschritten.

Und nun geht es wieder mal ans Ausprobieren. Alles scheint zu funktionieren – jedenfalls läuft das Programm.

Allerdings bin ich nicht ganz zufrieden: Die Funktion `IntToStr` sorgt dafür, dass wir als Ergebnis immer eine ganze Zahl erhalten. Das ist nicht unbedingt ein Problem, denn die Addition, Subtraktion und Multiplikation ganzer Zahlen ergeben auch immer ganze Zahlen.

Aber wie ist es bei der Division von z. B. 9 / 4? Da kommt doch entweder 2 Rest 1 heraus oder 2,25. Und damit hätten wir ganz gewiss keine ganze Zahl mehr.

- Probieren wir das gleich noch mal aus.



Damit würde der ganzzahlige Teil des Ergebnisses behalten, der Rest fällt unter den Tisch.

- Anschließend solltest du die Ergebnisse mit einem richtigen Taschenrechner überprüfen.

Versuchen wir's jetzt mal mit dieser Alternative, die richtige Dezimalzahlen liefert?

- Ändere die Methode `Button4Click` so um:

```
void __fastcall
 TForm1::Button4Click (TObject
 *Sender)
{
  Label3->Caption =
   FloatToStr (Zahl1 / Zahl2);
}
```

Neu ist die Funktion `FloatToStr` statt `IntToStr`. Frei übersetzt heißt es nun „Fließkommazahl in Zeichenkette (umwandeln)". Eine Fließkommazahl (oder auch Fließpunktzahl – siehe Taschenrechner) ist nichts anderes als eine *Dezimalzahl*.

Zum vorläufigen Abschluss dieses Projekts hier noch einmal alle Operatoren in einer Tabelle zusammengefasst.

+	Zwei beliebige Zahlen addieren
-	Zwei beliebige Zahlen subtrahieren
*	Zwei beliebige Zahlen multiplizieren
/	Zwei beliebige Zahlen dividieren
%	Rest einer Ganzzahldivision
=	Etwas (Zahl oder String) zuweisen
::	Klasse und Methode/Eigenschaft verknüpfen (Vereinbarung)
->	Objekt und Methode/Eigenschaft verknüpfen (Aufruf)

Zur Vervollständigung habe ich mit `%` noch einen Operator hinzugefügt, den man auch immer mal brauchen kann: Er liefert dir den Rest, der bei einer Ganzzahldivision auftreten kann. Kleines Beispiel zur Erläuterung:

`9 / 4` ergibt 2 (4 passt 2mal in 9)

`9 % 4` ergibt 1 (1 bleibt als Rest)

(In der Schule wird dieser Operator z. B. gern für Primzahlprogramme benutzt, um ganze Zahlen auf ihre Teilbarkeit zu testen.)

Aufgabe 2

Ergänze das Projekt um einen weiteren Button, um auf Knopfdruck den Divisionsrest anzuzeigen. (Mach dazu den NEU-Knopf etwas kleiner und setze den neuen Button rechts daneben.)