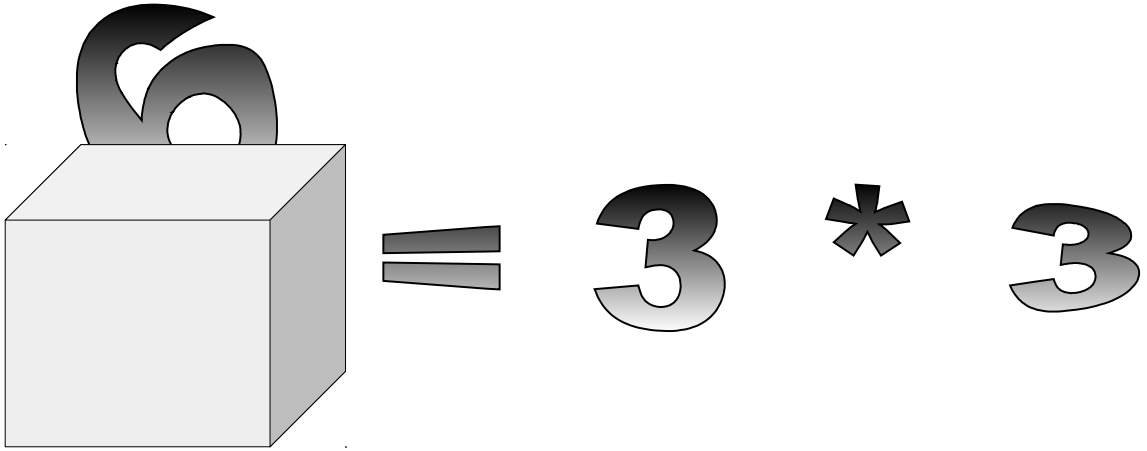
C++ - Einführung in die Programmiersprache

„Ausdrucksanweisungen“



Anweisungen

- Befehle für den Präprozessor. Bearbeitung des Quellcodes durch den Präprozessor.
- Befehle in der Programmiersprache C++. Abbildung von Arbeitsschritten in einem Prozess.

Anweisungen in C++

```
#include <iostream>
```

Präprozessor-Anweisungen

```
int ergebnis;
ergebnis = 5 + 5;
std::cout << ergebnis << std::endl;
return 0;
```

Interpretation durch den Compiler

Befehle in der Programmiersprache C++

- Formulierung einer Aktion entsprechend der Syntax der gewählten Programmiersprache.
- Befehle, die von einem Compiler interpretiert werden. Falls ein Fehler in der Syntax ist, wird die Interpretation abgebrochen.
- Beendigung mit einem Semikolon.
- Abarbeitung des Quellcodes von oben nach unten.
- Zusammenfassung mit Hilfe der geschweiften Klammern.

Möglichkeiten

`const int EXPONENT = 2;`	Deklarations-anweisungen
`int ergebnis;`	
`ergebnis = lZahl + rZahl;`	Ausdrucks-anweisungen
`ergebnis = pow(basis, EXPONENT);`	
`std::cout << "3^2 = " << ergebnis;`	
`return 0;`	Sprung-anweisungen

Leere Anweisung

```
;
{}
```

- Nur das Semikolon ohne eine Anweisung.
- Besser: Leere geschweifte Klammern als Kennzeichnung für einen leeren Block.
- Leere Anweisung führen an den falschen Stellen zu Fehlermeldungen!

Deklarationsanweisungen

```
int main () {
      int lZahl = 5;
      int rZahl = 2;
      int ergebnis = 0;
}
```

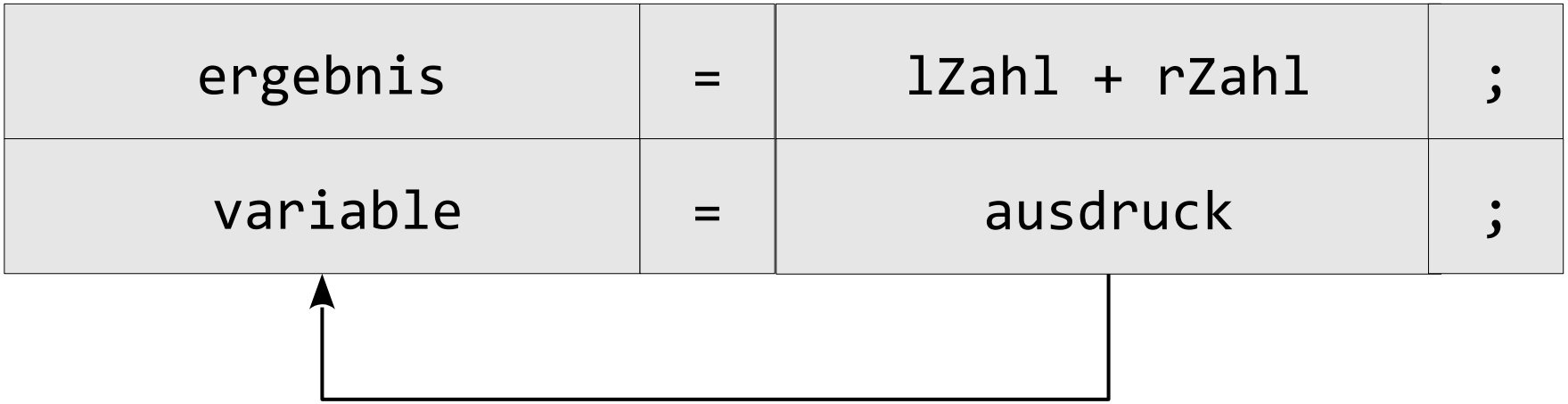
- Deklaration von Platzhaltern für konstante oder variable Werte.
- Vor der Nutzung muss ein Platzhalter deklariert.
- Die Initialisierung von Konstanten ist zwingend. Jede Konstante bekommt bei der Deklaration einen definierten Wert übergeben.
- Die Initialisierung von Variablen ist nicht erforderlich. Nach der Deklaration haben Variablen einen undefinierten Wert.

Ausdrucksanweisungen

```
int main () {
      // Deklaration der Variablen

      ergebnis = lZahl + rZahl;
      ergebnis = lZahl - rZahl;
      ergebnis = lZahl * rZahl;
      ergebnis = lZahl / rZahl;
      ergebnis = lZahl % rZahl;
}
```

Erläuterung



- In einer Ausdrucksanweisung weist der Zuweisungsoperator einer Variablen einen Wert zu.
- Dieser Wert wird mit Hilfe eines Ausdrucks berechnet.

Ausdruck

- Kombination aus Operatoren und Operatoren.
- Rückgabe eines Wertes.

Beispiele

- 2 + 3. Arithmetischer Ausdruck. Berechnung eines Wertes.
- 2 > 3. Aussagenlogischer Ausdruck. Die Aussage ist wahr oder falsch.

Operanden

- Variablen. Platzhalter für variable Werte. Der Wert kann sich durch eine Ausdrucksanweisung ändern.
- Konstanten. Platzhalter für einen definierten Wert. Der Wert kann nicht durch eine Ausdrucksanweisung verändert werden.
- Literale wie zum Beispiel 2, 3.5, 'A' . Die Werte werden direkt in in die Anweisung geschrieben. Literale werden einmalig an einer bestimmten Stelle im Code benötigt.
- Werte, die durch einen Funktionsaufruf ermittelt werden. Beispiel: `sin(wert)` liefert den Sinus-Wert.

Operatoren

- Zuweisungsoperator. Mit Hilfe des Gleichheitszeichens wird einer Variablen ein Wert zugewiesen.
- Arithmetische Operatoren berechnen einen Wert aus ein oder mehreren Operanden.
- Vergleichsoperatoren vergleichen zwei Werte.
- Logische Operatoren verknüpfen verschiedene Ausdrücke.
- Bit-Operatoren nutzen die binäre Darstellung von Zahlen.

… im Web

- https://de.wikibooks.org/wiki/C%2B%2B-Programmierung:_Operatoren
- http://www.cplusplus.com/doc/tutorial/operators/
- http://de.cppreference.com/w/cpp/language/operator_precedence

Leerzeichen in Ausdrucksanweisungen

```
bool vergleich;
int lZahl = 4;
int rZahl = 5;
int ergebnis=0;

ergebnis = lZahl + rZahl;
vergleich=lZahl>rZahl;
```

Erläuterung

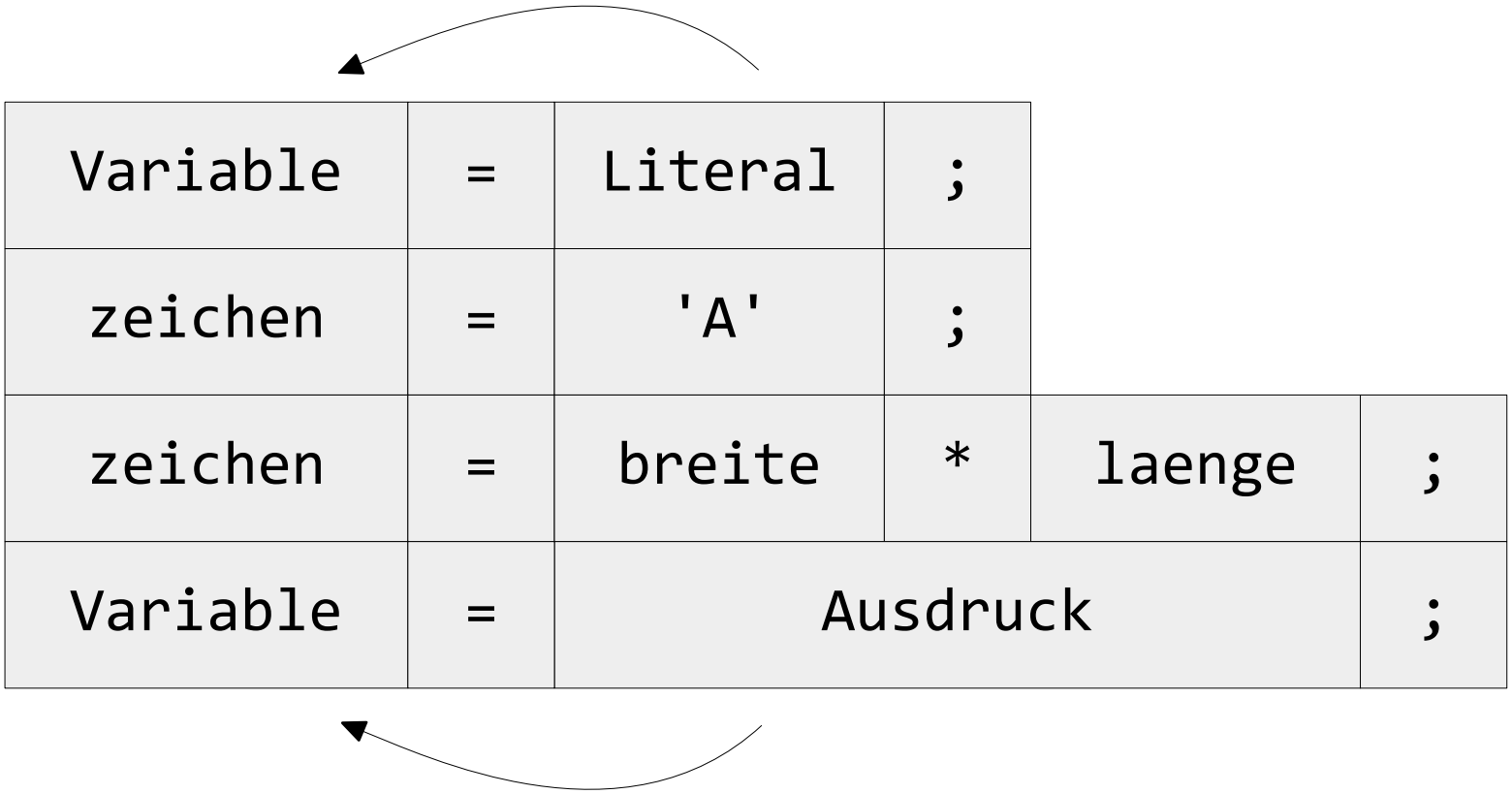
- Leerzeichen müssen zwischen Bezeichnern für Variablen, Konstanten etc. und Schlüsselwörtern gesetzt werden.
- Leerzeichen zwischen Operanden und Operatoren können gesetzt werden. Mit Hilfe von Leerzeichen wird aber die Lesbarkeit erhöht.

Ergebnis eines Ausdruckes

`Integer`	`=`	`Integer`	`/`	`Integer`	`;`
`Float`	`=`	`Float`	`/`	`Float`	`;`
`Float`	`=`	`Float`	`/`	`Integer`	`;`

- Jeder Ausdruck gibt einen Wert zurück.
- Dieser Wert hat einen bestimmten Datentyp.
- Wenn mindestens einer der Operanden vom Typ „Gleitkommazahl“ ist, ist auch der Wert des Ausdrucks vom Typ „Gleitkommazahl“.

Zuweisungsoperator



`Variable`	`=`	`Literal`	`;`		
`zeichen`	`=`	`'A'`	`;`		
`zeichen`	`=`	`breite`	`*`	`laenge`	`;`
`Variable`	`=`	`Ausdruck`			`;`

Hinweise

`long`	`=`	`double`	`*`	`int`	
`zeichen`	`=`	`breite`	`*`	`laenge`	`;`
`Variable`	`=`	`Ausdruck`			`;`
`long`	`=`	`double`			

Falls möglich,
automatische
Konvertierung.

Arithmetische Operatoren

*	Multiplikation
/	Division
%	Modula
+	Addition
-	Subtraktion

Arithmetische Operatoren und Ganzzahlen

		5		2
10	=	lWert	*	rWert
2	=	lWert	/	rWert
1	=	lWert	%	rWert
7	=	lWert	+	rWert
3	=	lWert	-	rWert

Hinweise zur Division

```
int divisor = 5;
int divident = 0;
int ergebnis;

ergebnis = divisor / divident;
```

- Eine Division durch 0 ist nicht erlaubt.
- Das Programm stürzt ab.

Hinweise zu Modulo

```
int divisor = 5;
int divident = 2;
int ergebnis;

ergebnis = divisor % divident;
```

- Ganzzahlige Division.
- Division mit Rest. 5 / 2 = 2, Rest 1 → 5 = 2 * 2 + 1. Der Rest der Division wird als Ergebnis zurückgeliefert.

Arithmetische Operatoren und Gleitkommazahlen

		5.0		2
10.0	=	lWert	*	rWert
2.5	=	lWert	/	rWert
Invalid operands	=	lWert	%	rWert
7.0	=	lWert	+	rWert
3.0	=	lWert	-	rWert

Gleitkommazahl: Division durch Null

```
float divisor = 5.0;
float divident = 0;
float ergebnis;

ergebnis = divisor / divident;
```

- Das Ergebnis wird als Gleitkommazahl gespeichert.
- Die Variable hat den Wert INF (Infinity).
- Ein Wert, der nicht in dem Datentyp darstellbar ist.

Auswertung von Ausdrücken

			20.0		2.0			2.0
11.0	=	(	lWert	+	rWert	)	/	rWert
21.0	=		lWert	+	rWert		/	rWert

- In C++ gilt die Punkt- vor Strichrechnung.
- Mit Hilfe von runden Klammern werden Ausdrücke zusammengefasst. Die Rangfolge von Operatoren kann verändert werden.

Rangfolge der Operatoren

(	)	
- (Vorzeichen)		
+ (Vorzeichen)		
*	/	%
+	-	

Inkrement und Dekrement

```
ergebnis = zahl++;
ergebnis = ++zahl;

ergebnis = zahl--;
ergebnis = --zahl;
```

- Inkrement: `variable = variable + 1`.
- Dekrement: `variable = variable - 1`.

Operatoren

`wert`	`=`	`++`	`var`
`wert`	`=`	`--`	`var`

- Der Inkrementoperator besteht aus zwei Pluszeichen.
- Der Dekrementoperator besteht aus zwei Minuszeichen.
- Hinweis: Zusammengesetzte Operatoren dürfen nicht durch ein Leerzeichen getrennt werden!

Präfix

var	=		5
wert	=	++	var

var	=	var	+	1
wert	=	var		

var	=		5
wert	=	--	var

var	=	var	-	1
wert	=	var		

Postfix

var	=	5	
wert	=	var	++

wert	=	var		
var	=	var	+	1

var	=	5	
wert	=	var	--

wert	=	var		
var	=	var	-	1

Typ eines Ausdrucks

```
typeid(3 / 2).name();
typeid(3 / 2.0).name();
typeid(3.0 / 2.0).name();
typeid('a' - 32).name();
```

- Der Operator `typeid` ermittelt den Datentyp einer Variablen oder Ausdrucks zur Laufzeit eines Programms.
- Die Definition des Operators ist in der Bibliothek `typeinfo` hinterlegt.

Aufbau

typeid	(	ausdruck	)	.	name	(	)
typeid	(	3 / 2	)	.	name	(	)

- Beginn mit dem Schlüsselwort `typeid`.
- Dem Schlüsselwort folgen runde Klammern. In den runden Klammern wird dem Operator ein Ausdruck übergeben.
- Der Operator hat eine Methode `.name()`. Die Methode gibt Namen des Datentyps zurück. Der Operator und die Methode werden durch den Punkt verbunden.

Beispiele für Rückgabewerte

`i`	Integer. Ganzzahl vom Typ int
`d`	Double. Gleitkommazahl vom Typ double.
`b`	Boolsche Werte.
`c`	Char. Ein einzelnes Zeichen.

Konvertierung von Datentypen

- Unterschiedliche Datentypen in Ausdrücken werden entsprechend der Variablen, links vom Zuweisungsoperator, konvertiert.
- Implizite, automatische Umwandlungen.
- Explizite, vom Programmierer gesteuerte Umwandlungen.
- Beide Arten von Konvertierungen können unterschiedliche Ergebnisse liefern.

Implizite Konvertierung

```
int code = 97;
char buchstabe = code;
double wert = code;

bool an = true;
int wahr = an;
```

- Der Wert eines Ausdruckes wird in einer Variablen von einem anderen Typ gespeichert.

Überschreitung des Wertebereiches

- Bei einer impliziten Konvertierung wird ein beliebiger passender Wert entsprechend des Datentyps genutzt.
- Bei einer Konvertierung in einem Datentyp `bool` wird immer `true` genutzt.
- Bei einer Konvertierung in einem Datentyp `char` wird ein undefiniertes Zeichen zurückgegeben.

… in C++11

```
int code = 97;
char buchstabe = {code};
double wert = {code};

bool an = true;
int wahr = {an};
```

- Der Wert eines Ausdruckes wird in einer Variablen von einem anderen Typ gespeichert.
- Bei einem Überschreiten des Wertebereichs wird die Warnung „narrowing conversion“ angezeigt.

Probleme

`double` → `float`	Verlust der Genauigkeit. Undefiniertes Ergebnis.

`double` → `int`
`double` → `long`
`float` → `int`
`float` → `long`
	Streichung der Nachkommastellen ohne Rundung.
	
	Ein zu kleiner Wertebereich liefert ein undefiniertes Ergebnis.

`int` → `short` `long` → `int`	Konvertierung der niedrigsten Bits Verlust von Informationen.

Explizite Konvertierung

```
int intRWert = 2;
int intLWert = 5;
double ergebnis;

ergebnis = double(intLWert / intRWert);
ergebnis = double(intLWert) / intRWert;
```

… in C++

`Datentyp`	`(`	`ausdruck`	`)`

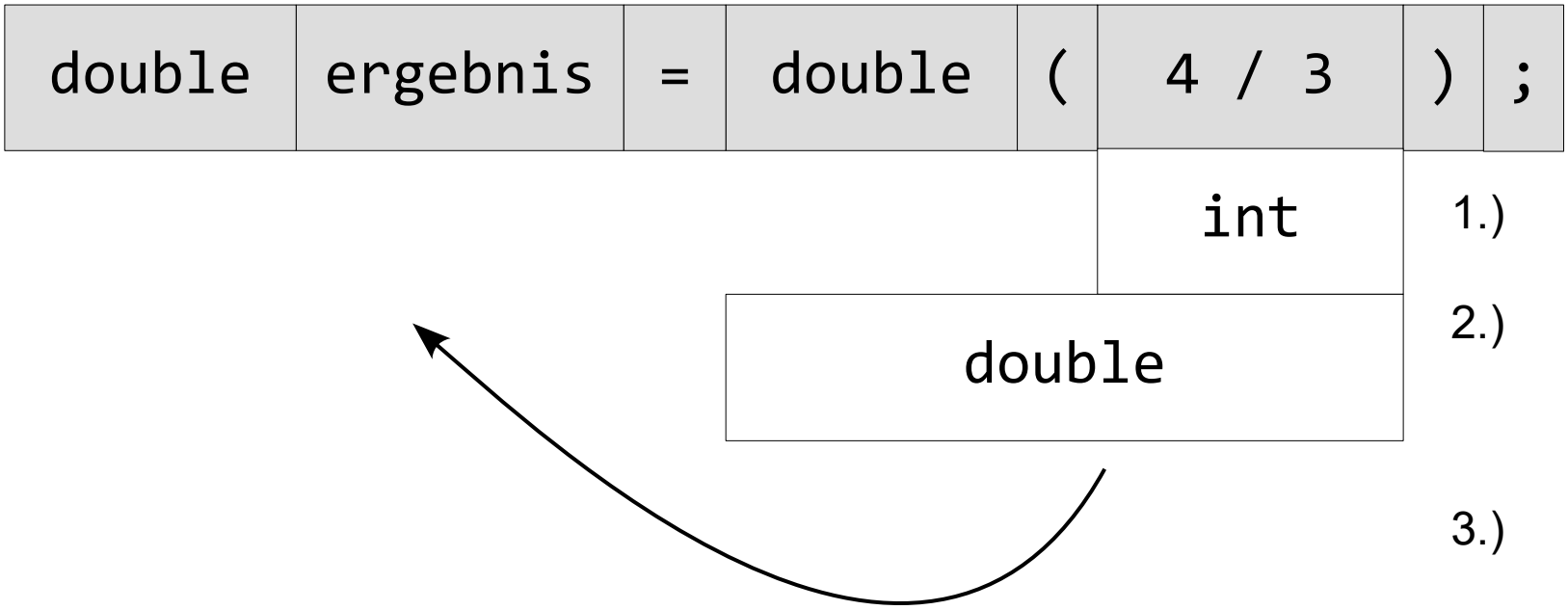
- In runden Klammern: Der zu konvertierende Wert.
- Links von der öffnenden runden Klammern: Ein beliebiger Standard-Datentyp. Der Wert des Ausdrucks wird in diesen konvertiert.

Beispiel

`ergebnis`	`=`	`double`	`(`	`int`	`)`	`;`
`variable`	`=`	`Datentyp`	`(`	`ausdruck`	`)`	`;`
`variable`	`=`	`ausdruck`				`;`

Der Datentyp des Ausdrucks sollte mit dem Datentyp der Variablen übereinstimmen.

Arbeitsschritte



Beispiel

`ergebnis`	`=`	`Datentyp`	`(`	`ausdruck`	`)`	`;`
`1`	`=`	`double`	`(`	`3 / 2`	`)`	`;`
`1.5`	`=`	`double`	`(`	`3.0 / 2`	`)`	`;`

Restriktive Typumwandlung

```
int intRWert = 2;
int intLWert = 5;
double ergebnis;

ergebnis = static_cast<double>(intLWert) / intRWert;
```

- Konvertierung zwischen Ganzzahlen und Gleitkommazahlen.
- Konvertierung in Abhängigkeit von Regeln.
- Entspricht der impliziten Konvertierung.

Erläuterung

`static_cast`	`<`	`Datentyp`	`>`	`(`	`ausdruck`	`)`

- Die Konvertierung beginnt mit dem Schlüsselwort static_cast.
- Dem Schlüsselwort folgen spitze Klammer. Die Konvertierung soll in diesen Datentyp erfolgen.
- Den spitzen Klammern folgen runde Klammern. In den runden Klammern wird der zu konvertierende Ausdruck angegeben.

Ableitung von Datentypen

```
decltype(5.0 / 2) ergebnis01  = 5 / 2;
auto ergebnis03  = 5 / 2;
```

- Einführung mit C++11.
- Mit Hilfe der Funktion `decltype()` wird ein Datentyp definiert.
- Durch die Kennzeichnung `auto` wird ein Datentyp entsprechend des Wertes eines Ausdrucks gewählt.

Funktion „decltype“

```
decltype(5.0 / 2) ergebnis01  = 5 / 2;
```

- Einführung mit C++11.
- Die Funktion `decltype()` wird mit ihren Namen aufgerufen.
- Dem Namen folgt eine Parameterliste in runden Klammern. In den runden Klammern wird ein Ausdruck definiert.
- Der übergebene Parameter legt den Datentyp der Variablen fest.

Nutzung von auto

```
auto ergebnis03  = 5 / 2;
```

- Berechnung des Ausdrucks. In diesem Beispiel hat der berechnete Wert den Typ `int`.
- Zuweisung des Wertes an die Variable. Die Variable hat den Typ `int` wie der Wert des Ausdruckes.