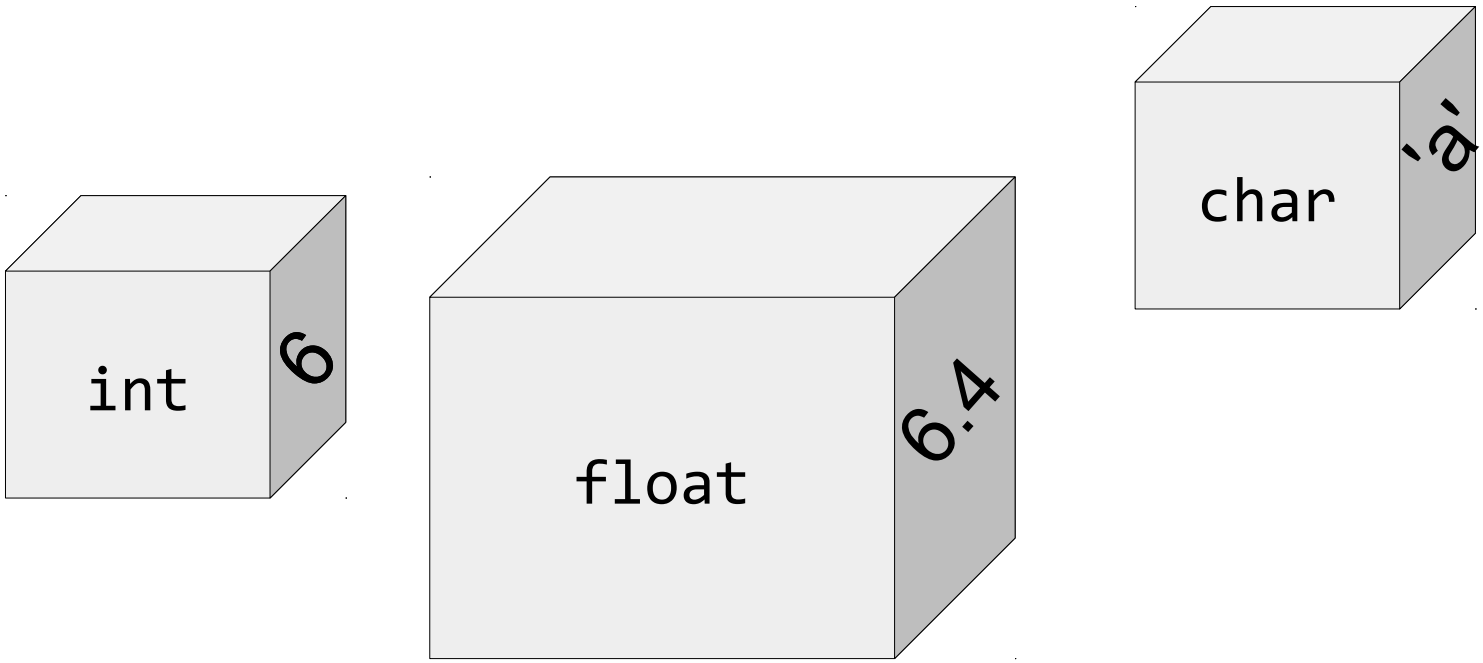
C++ - Einführung in die Programmiersprache

„Deklaration von Konstanten“



Kommentare

- Hilfe für den Entwickler.
- Wer hat wann welche Änderung vorgenommen?
- Warum werden diese Anweisungen hier ausgeführt?
- Kommentare müssen bei Änderungen im Code angepasst werden.
- Kommentare werden bei der Kompilierung überlesen.

Beispiel: Kommentare im Code

```
#include <iostream>

int main()
{
    /*
     * Autor: C++-Kurs
     * Erstellt am 04.01.2009
     * Addition von 5  + 5
     */
    int ergebnis;
    // Addition
    ergebnis = 5 + 5;
    std::cout << ergebnis << std::endl;
    return 0;
}
```

Einzeiliger Kommentar ...

```
// Addition
```

- Beginn: Zwei Schrägstriche ohne Leerzeichen.
- Positionierung rechts von einer Zeile: Der Kommentar bezieht sich auf den Code in der Zeile.
- Positionierung am Anfang einer Methode oder oberhalb einer Zeile: Der Kommentar erläutert den nachfolgenden Abschnitt.

Mehrzeiliger Kommentar ...

```
/*
  * Autor: C++-Kurs
  * Erstellt am 04.01.2009
  * Addition von 5  + 5
  */
```

- Beginn: Schrägstrich und Sternchen.
- Ende: Sternchen und Schrägstrich.
- Positionierung am Beginn eines Programms, vor einer Methode oder oberhalb einer Zeile: Der Kommentar erläutert den nachfolgenden Abschnitt.
- Hinweis: Mehrzeilige Kommentare sind nicht verschachtelbar.

Anweisungen

- Befehle für den Computer in einer Programmiersprache.
- Beschreibung eines Arbeitsschrittes entsprechend der Syntax einer Sprache.
- Anweisungen werden von oben nach unten in einer Quelldatei ausgeführt.
- Anweisungen werden vom Compiler auf Syntaxfehler überprüft.

… in C++

```
#include <iostream>
```

Präprozessor-Anweisungen

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

Anweisungen in Codeblöcken

Präprozessor-Anweisungen

```
#include <iostream> // Ein- und Ausgabe
#include <cmath>   // Mathematische Funktionen
```

- Beginn mit dem Hash-Zeichen.
- Pro Zeile eine Anweisung. Das Zeilenende markiert auch das Ende der Anweisung.
- Am Anfang der Quelldatei werden alle Präprozessor-Anweisungen aufgelistet.
- Der Präprozessor ersetzt die Anweisung durch den entsprechenden Textabschnitt.

Anweisungen in Codeblöcken

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

- Anweisungen beschreiben eine Aktivität.
- Das Semikolon beendet Anweisungen im Code.
- Pro Zeile sollte eine Anweisung stehen.

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

Zusammenfassung von Anweisungen

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

- Mit Hilfe der geschweiften Klammern werden Anweisungen zu Blöcken zusammengefasst.

Zusammenfassung von Anweisungen

- Mit Hilfe von Klassen werden Anweisungen zusammengefasst, die ein Objekt beschreiben.
- Mit Hilfe von Prozeduren, Funktionen und Methoden werden Anweisungen zu einer Aktivität zusammengefasst.
- Ausführung von Anweisungen in Codeblöcken von oben nach unten.
- In Abhängigkeit einer Bedingung werden Anweisungen ausgeführt oder nicht.
- Durch Schleifen werden Codeblöcke x Mal wiederholt.

Deklarationsanweisungen

```
{
  const int EXPONENT = 2;
  int ergebnis;
  int basis;
}
```

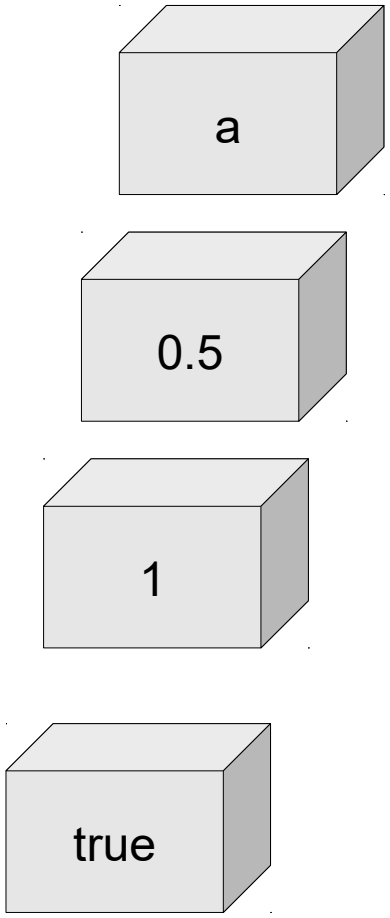
- Deklaration von Platzhaltern für einen Wert von einem bestimmten Typ.
- Bekanntmachung von Platzhaltern für alle nachfolgenden Anweisung.
- Deklaration stehen häufig am Anfang eines Codeblockes.

Platzhalter

- Name für einen variablen oder konstanten Wert.
- Vor der Nutzung müssen Platzhalter deklariert werden.
- Der Anfangswert des Platzhalters ist nach der Deklaration nicht definiert. Die Initialisierung übergibt einen definierten Wert an den Platzhalter.

Konstante Werte

- Werte, die sich im Programmablauf nicht verändern.
- Zum Beispiel die Zahl PI, der niedrigste oder höchste Wert eines Zahlenbereichs, der Mehrwertsteuersatz.
- Konstante Werte können Gleitkommazahlen, Ganzzahlen, einzelne Buchstaben etc. sein.



Beispiele in C++

```
int main () {

    const float DEZIMALZAHL = 0.5;
    const int   GANZZAHL = 1;
    const bool  AN = true;
    const char  ZEICHEN = 'a';

    return 0;
}
```

Deklarationsanweisung

`const`	`int`	`GANZZAHL`	`=`	`1`	`;`

- Platzhalter für konstante Werte müssen gleichzeitig deklariert und initialisiert werden. Andernfalls wird die Fehlermeldung „uninitialized const ...“
- Die Deklarationsanweisung `const int GANZZAHL` legt den Datentyp und den Namen des Platzhalters fest.
- Die Initialisierung `GANZZAHL = 1` bestimmt den Wert des Platzhalters.

Deklaration von Konstanten

`const`	`int`	`GANZZAHL`		`;`
`const`	`Datentyp`	`Name`		`;`

- Die Deklaration beginnt mit dem Schlüsselwort `const`.
- Dem Schlüsselwort folgt der Datentyp der Konstanten. In diesem Beispiel wird eine Ganzzahl (`int`) als Konstante genutzt.
- Dem Datentyp folgt der Name der Konstanten. Der Bezeichner einer Konstanten ist frei wählbar.

… in Codeblöcken

- Jede Deklaration ist einmalig. Jeder Name kommt nur exakt einmal in einem Codeblock vor. Falls der Name mehrmals genutzt wird, wird die Fehlermeldung: „conflicting declaration ...“ angezeigt.
- Der Name ist nur in dem Codeblock bekannt, in dem er deklariert wurde. Bei einer Nutzung außerhalb wird die Fehlermeldung: „… was not declared in this scope“ angezeigt.

… und Initialisierung

	GANZZAHL	=	1	;
	Name	=	Wert	;

- Der Konstanten wird mit Hilfe des Gleichheitszeichen ein Wert zugewiesen.
- Links vom Gleichheitszeichen steht der Name der Konstanten. Unter dem „Namen“ wird der Wert im Speicher abgelegt.
- Der Wert der Konstanten wird rechts vom Gleichheitszeichen angegeben. Der Wert entspricht dem Datentyp der Konstanten.

Bezeichner

- Benutzerdefinierte Namen für Konstanten, Variablen, Funktionen etc.
- Die Namen sind in ihrem Codeblock eindeutig. Ein Codeblock beginnt und endet mit den geschweiften Klammern.
- Schlüsselwörter der Programmiersprache sind als benutzerdefinierte Namen nicht erlaubt.
- Unterscheidung zwischen Groß- und Kleinschreibung. Die Namen „MINZAHL“ und „MINzAHL“ sind unterschiedliche Platzhalter in C++.

Erlaubte Zeichen

- Buchstaben A...Z und a...z.
- Zahlen 0...9.
- Der Unterstrich.

Beispiele

```
radius
```

```
PI
```

```
menge_pro_artikel
```

```
BESTELLWERT_MIN
```

```
farbe_rgb
```

```
EXPONENT
```

Geeignete Bezeichner

- Der Name spiegelt die Nutzung des Bezeichners in dem Code wieder.
- Der Bezeichner nutzt den Namen des abgebildeten Objektes aus der Realität.
- Die Namen der Platzhalter nutzen eine Sprache. Die Sprachen englisch und deutsch, zum Beispiel, sollten nicht gemischt werden.
- Keine kryptischen Bezeichner wie a1, b etc.
- Bezeichner aus einem Buchstaben werden häufig nur Zähler oder Indizes genutzt.

Konventionen für Konstanten

- Konstanten beginnen mit einem Buchstaben.
- Für Konstanten werden Großbuchstaben genutzt.
- Leerzeichen werden durch ein Unterstrich ersetzt.

Datentypen

- Baupläne für Platzhalter.
- Regeln für die Interpretation und Verwendung eines Wertes.
- Speicherbedarf / Größe eines Wertes.

… in C++

`int`	`1`	Ganzzahl.
`float`	`0.5`	Dezimal- oder Fließkommazahl.
`bool`	`true`	Boolsche Werte.
`char`	`'a'`	Ein einzelnes Zeichen.

Datentypen für boolsche Werte

`const`	`bool`	`LICHT_AN`	`=`	`true`	`;`
`const`	`bool`	`LICHT_AUS`	`=`	`false`	`;`

- Wahrheitswerte. Annahme von nur zwei Zuständen.
- `true`. Wahr. Strom / Licht ist eingeschaltet. Der Wert ist ungleich 0.
- `false`. Falsch. Strom / Licht ist ausgeschaltet. Der Wert ist 0.
- Häufig werden Adjektive als Bezeichner genutzt.

Ganzzahlen

- Zahlen ohne Nachkommastellen.
- Der Zahlenraum ist endlich.
- Berechnungen mit ganzen Zahlen sind exakt.

Zahlensysteme

- Das Dezimalsystem basiert auf der Zahl 10. Es werden die Ziffern 0 bis 9 genutzt.
- Das Oktalsystem basiert auf der Zahl 8. Es werden die Ziffern 0 bis 7 genutzt. Zahlen im Oktalsystem haben das Präfix „Null“.
- Das Hexadezimalsystem basiert auf der Zahl 16. Es werden die Ziffern 0 bis 9 und die Buchstaben A bis F genutzt. Zahlen im Hexadezimalsystem werden durch 0x gekennzeichnet. Farbcodierungen werden häufig im Hexadezimalsystem dargestellt.

Schreibweise

`ganzzahl`	`=`	`42`	`;`
`ganzzahl`	`=`	`052`	`;`
`ganzzahl`	`=`	`0x2a`	`;`

dezimal

oktal

hexadezimal

Datentypen

const	short	ganzzahl	=	1	;
const	int	ganzzahl	=	123	;
const	long	ganzzahl	=	12345L	;
const	long long	ganzzahl	=	123456LL	;

Regeln für die Größe

- Der Datentyp `short` ist mindestens 16 Bits groß.
- Der Datentyp `int` ist mindestens genauso groß wie `short`.
- Der Datentyp `long` ist mindestens genauso groß wie `int`. `long` hat mindestens 32 Bits.
- Der Datentyp `long long` ist mindestens genauso groß wie `long`. `long long` hat mindestens 64 Bits. Einführung mit C++11.

Hinweise zu Literale

- Literale werden standardmäßig als Werte vom Datentyp `int` interpretiert.
- Mit Hilfe des Suffix `L` oder `LL` wird der gewünschte Datentyp angegeben.

Gleitkommazahlen

- Darstellung einer reellen Zahl.
- Repräsentation einer Zahl durch Vorzeichen, Mantisse und Exponent.
- Nährung einer Zahl. Der Datentyp gibt die Genauigkeit der Nährung an.
- Als Dezimaltrennzeichen wird der Punkt genutzt.

… in der Dezimalschreibweise

+	5	•	8
-	5	•	8
		•	8

… in der Exponentialschreibweise

+	5	•	8	e+	16
+	5	•	8	E+	16
-	5	•	8	e-	16
-	5	•	8	E-	16

- Nutzung für extrem kleine oder große Zahlen.
- Zahlen in der Exponentialschreibweise werden immer als Gleitkommazahl gespeichert.

Hinweise

- 5.8e+16 (= 5.8 * 10^{16}). Das Dezimaltrennzeichen wird um n Stellen nach rechts verschoben.
- 5.8e-16 (= 5.8 / 10^{16}). Das Dezimaltrennzeichen wird um n Stellen nach links verschoben.

Datentypen

`const`	`float`	`wert`	`=`	`0.5F`	`;`
`const`	`double`	`wert`	`=`	`0.5`	`;`
`const`	`long double`	`wert`	`=`	`0.5L`	`;`

Regeln

- `float` ist mindestens 32 Bit groß. Der Datentyp hat eine Genauigkeit von ca. 6 Stellen.
- `double` ist mindestens genauso groß wie `float`, aber mindestens 48 Bits. Der Datentyp hat eine Genauigkeit von ca. 15 Stellen.
- `long double` ist mindestens genauso groß wie `double`.

Hinweise zu Literale

- Literale werden standardmäßig als Werte vom Datentyp `double` interpretiert.
- Mit Hilfe des Suffix `L` und `F` wird der gewünschte Datentyp angegeben.

Alphanumerische und numerische Zeichen

- Die kleinste Einheit eines Textes.
- Speicherung eines Zeichens entsprechend seiner Kodierung. Zum Beispiel kann das alphanumerische Zeichen a gespeichert werden.
- Escape-Sequenzen wie '\n' für einen Zeilenumbruch. Escape-Zeichen maskieren mit Hilfe des Backslash ein Zeichen aus dem Alphabet.

Datentyp char

`const`	`char`	`zeichen`	`=`	`'a'`	`;`
`const`	`bool`	`sonder`	`=`	`'\n'`	`;`

- Beginn und Ende immer mit dem Apostroph.
- Numerische Zeichen vom Datentyp `char` können nicht in Berechnungen genutzt werden.
- Ein Apostroph muss maskiert werden (`'\''`).

Codierung von Zeichen

```
char zeichen;

zeichen = 'A';
zeichen = 65;
```

- Jedes Zeichen vom Typ `char` wird mit Hilfe einer Zeichentabelle codiert.
- Jedes Zeichen wird durch einen eindeutigen Integer-Wert in der Zeichentabelle identifiziert.
- Standardmäßig werden die Zeichen mit Hilfe des ASCII-Zeichensatz codiert.

ASCII-Zeichensatz

- American Standard Code for Information Interchange.
- Definition von 128 Zeichen.
- Siehe http://www.torsten-horn.de/techdocs/ascii.htm

„Maskierung“ von Zeichen

```
char unicode = = '\u0041';
char newLine = '\n';
```

- Escape-Sequenzen.
- Mit Hilfe des Schrägstrichs wird ein Zeichen maskiert.
- Zeichen, die mit einem Schrägstrich beginnen, haben eine besondere Bedeutung für den Compiler.

Escape-Sequenzen

```
char newLine = '\n';
char apostroph = '\'';
```

- Steuerzeichen für den Drucker etc.
- Nicht druckbare Zeichen eines Zeichensatzes.
- Maskierung von Zeichen, die in C++ in einer besonderen Funktion genutzt werden.

Möglichkeiten

Escape-Sequenz	Beschreibung
\b	Rückschritt (Backspace)
\t	Tabulator
\v	Vertikaler Tabulator
\n	Zeilenumbruch (Newline)
\r	Wagenrücklauf. (Carriage Return)
\a	Alarm.
\?	Fragezeichen
\"	Anführungszeichen
\'	Apostroph
\\	Backslash

Leeres Zeichen

```
const char null = '\0';
const char space = ' ';
```

- Zwei direkt aufeinanderfolgende Apostrophe erzeugen die Fehlermeldung „empty character constant“.
- Um ein leeres Zeichen abzubilden, wird häufig die maskierte 0 oder das Leerzeichen genutzt.
- Das Zeichen \0 terminiert einen String.

Unicode-Zeichen

```
char unicode = = '\u0041';
```

- Maskierung mit \u.
- Das erste Zeichen im Unicode-Zeichensatz wird folgendermaßen kodiert: `'\u0000'`.

Unicode-Zeichensatz

- Das erste Zeichen im Zeichensatz wird mit Hilfe von `'\u0000'` angegeben.
- Die ersten 127 Zeichen des UTF-8-Zeichensatzes sind mit dem ASCII-Zeichensatz identisch.
- Die ersten 256 Zeichen des UTF-8-Zeichensatzes entsprechen dem ISO 8859-1 (Latin 1)-Zeichensatz.
- Siehe https://unicode-table.com/de/.

Nutzung eines Strings

```
char zeichen;

zeichen = 'A';
cout << "\n ASCII-Zeichen: " << zeichen;
```

- Strings sind Zeichenketten aus beliebig vielen numerischen und alphanumerischen Zeichen.
- Beginn und Ende mit Hilfe des Anführungszeichens.
- Strings werden nicht über einen Standard-Datentyp definiert.