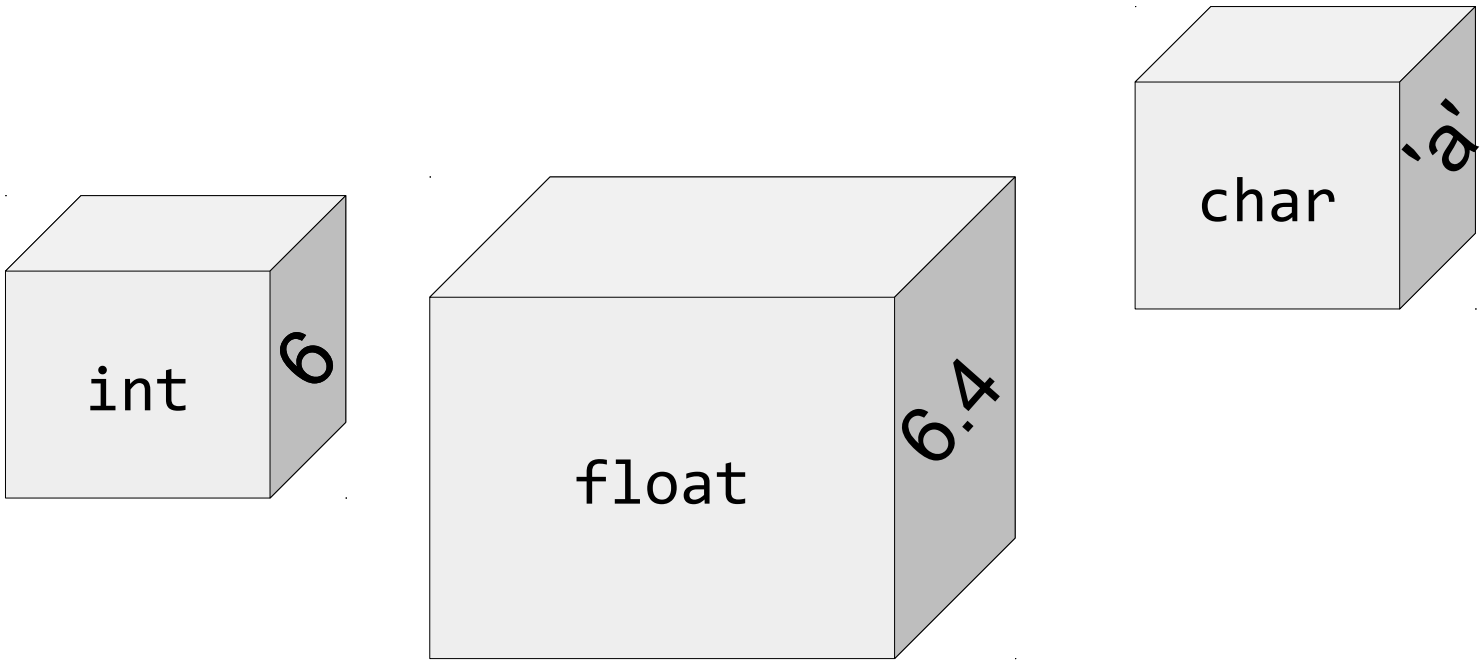
C++ - Einführung in die Programmiersprache

„Deklaration von Variablen“



Anweisungen

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

Beispiele

```
#include <iostream>
```

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

    std::cout << lZahl << '+' << rZahl;
    std::cout << " = " << (lZahl + rZahl);
    std::cout << std::endl;
    return 0;
}
```

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.
- Positionierung am Anfang einer Quelldatei.
- Die Programmiersprachen C und C++ nutzen die Befehle gemeinsam.

Möglichkeiten

- Während der Kompilierung wird die Anweisung `#include` durch den entsprechenden Quelltext ersetzt.
- Die Anweisung `#define` ersetzt definierte Makros im Quelltext.
- Die Befehl können in Abhängigkeit einer Bedingung (`#if`) übersetzt werden.

Befehle in der Programmiersprache C++

- Formulierung einer Aktion entsprechend der Syntax der gewählten Programmiersprache.
- Beendigung mit einem Semikolon.
- Zusammenfassung mit Hilfe der geschweiften Klammern.

Beispiel

```
{
    int lZahl = 5;
    int rZahl = 2;

    std::cout << lZahl << '+' << rZahl;
    std::cout << " = " << (lZahl + rZahl);
    std::cout << std::endl;
    return 0;
}
```

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

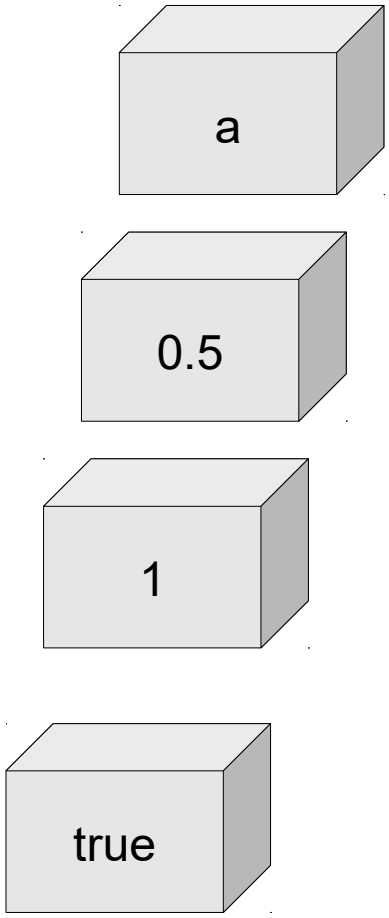
Deklarationsanweisungen

```
const int EXPONENT = 2;
int ergebnis;
int lZahl = 5;
int rZahl = 2;
```

- Deklaration von Platzhaltern für konstante und variable Werte.
- Deklarationsanweisungen befinden sich häufig am Anfang eines Codeblockes. Sie können aber auch an jeder anderen beliebigen Stelle in einem C++-Programm platziert werden.

Variable Werte

- Werte, die sich im Programmablauf ändern.
- Zum Beispiel: Berechnungen der Mehrwertsteuer für einen Einkauf, Messwerte in Abhängigkeit der Tageszeit, Anzahl der Waren im Lager.
- Variable Werte können Gleitkommazahlen, Ganzzahlen, einzelne Buchstaben etc. sein.



Beispiele in C++

```
int main () {

    bool licht_an = true;
    double dezimalzahl;
    int ganzzahl = 1;
    char zeichen;

    dezimalzahl = ganzzahl / 2;
    zeichen = 'A';

    return 0;
}
```

Variablen in C++

`int`	Datentyp. Wie viel Speicherplatz wird benötigt? Wie groß muss die „Box“ sein?
`ganzzahl`	Name. Wie wird die „Box“ (der Speicherplatz) bezeichnet?
`= 1`	Wert. Was wird gespeichert?

Deklaration

`float`	`dezimalzahl`	`;`
`Datentyp`	`name`	`;`

- Jede Variablendeklaration beginnt mit der Angabe des Datentyps.
- Dem Datentyp folgt ein Bezeichner. Der Bezeichner ist frei wählbar.

Bezeichner (Variablennamen)

- Platzhalter für variable und konstante Werte.
- Die Namen sind in ihrem Codeblock eindeutig.
- 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. Einzelne Buchstaben werden für Zähler oder Indizes bei Feldern genutzt.

Konventionen für Variablen

- Variablen beginnen mit einem Buchstaben.
- Variablen nutzen Kleinbuchstaben und Zahlen.
- Als Bezeichner wird häufig ein Substantiv genutzt.
- Jedes Wort in einem Variablennamen beginnt mit einem Großbuchstaben. Es wird die Kamel-Notation genutzt. Beispiel: `durchschnittstemperaturProMonat`, `preisProStueck`.

Informationen in einem Variablennamen

n	c	menge
Basistyp	Präfix	Name

- Basistyp: Welcher Datentyp wird genutzt? Art des Wertes. Hier: Numerische Ganzzahl.
- Präfix: Variablenkategorie. Hier: Die Variable wird als Zähler genutzt.
- Bezeichner: Bezeichnung des dargestellten Objekts aus der realen Welt.

Beispiele für den Basistyp

`n`	Numerischer ganzzahliger Wert.
`u`	Numerischer ganzzahliger Wert ohne Vorzeichen.
`b`	Boolsche Werte.
`f`	Float. Gleitkommazahl.
`ch`	Char. Ein einzelnes Zeichen.

Beispiele für ein Präfix

a	Array. Feld.
c	Zähler.
p	Pointer. Zeiger.
g_	Globale Variable.

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`	`;`	dezimal
`ganzzahl`	`=`	`052`	`;`	oktal
`ganzzahl`	`=`	`0x2a`	`;`	hexadezimal

Standard-Datentypen

`short`	`ganzzahl`	`;`
`int`	`ganzzahl`	`;`
`long`	`ganzzahl`	`;`
`long long`	`ganzzahl`	`;`

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

- `a = 15`. Interpretation als Wert vom Datentyp `int`.
- `a = 15L`. Interpretation als Wert vom Datentyp `long`.
- `a = 15LL`. Interpretation als Wert vom Datentyp `long long`.

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

`float`	`wert`	`;`
`double`	`wert`	`;`
`long double`	`wert`	`;`

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

- `a = 0.45`. Interpretation als Wert vom Datentyp `double`.
- `a = 0.45F`. Interpretation als Wert vom Datentyp `float`.
- `a = 0.45L`. Interpretation als Wert vom Datentyp `long double`.

Wert einer deklarierten Variablen

`float`	`dezimalzahl`	`;`
`Datentyp`	`name`	`;`

- Undefinierter Wert entsprechend des Datentyps.
- Die Variable wird mit einem beliebigen Wert initialisiert.

1. Möglichkeit

```
char zeichen;
zeichen = 'A';
```

- Im ersten Schritt wird die Variable deklariert.
- Im zweiten Schritt wird der Variablen ein definierter Wert entsprechend des Datentyps zugewiesen. Der Zuweisungsoperator wird durch das Gleichheitszeichen dargestellt.

Zuweisungsoperator

`variable`	`=`	`wert`	`;`
`variable`	`=`	`ausdruck`	`;`

- Platzhalter für einen variablen Wert.
- Muss deklariert sein.

- Nutzung eines Literals.
- Berechnung eines Wertes
- Speicherung auf dem Stack.

2. Möglichkeit

```
char zeichen  = 'A';
```

- Variablen können wie Konstanten gleichzeitig deklariert und initialisiert werden.

Deklaration und Initialisierung

`float`	`dezimalzahl`	`=`	`0.5`	`;`
`Datentyp`	`name`	`=`	`wert`	`;`

Deklaration

Initialisierung

Beispiele für Initialisierungswerte

```
bool licht = false;
double wert = 0.0;
int zahl = 0;
```

… seit C++11

float	dezimalzahl	=	{	0.5	}	;
Datentyp	name	=	{	wert	}	;

float	dezimalzahl	{	0.5	}	;
Datentyp	name	{	wert	}	;

Nutzung einer Initialisierungsliste

- Die Initialisierungsliste beginnt und endet mit den geschweiften Klammern.
- In den geschweiften Klammern werden die Initialisierungswerte aufgelistet. Für die Initialisierung von Variablen wird immer nur ein Wert angegeben.
- Der Zuweisungsoperator zwischen dem Platzhalter und der Initialisierungsliste kann gesetzt werden, muss aber nicht.

Beispiele für eine Initialisierungsliste

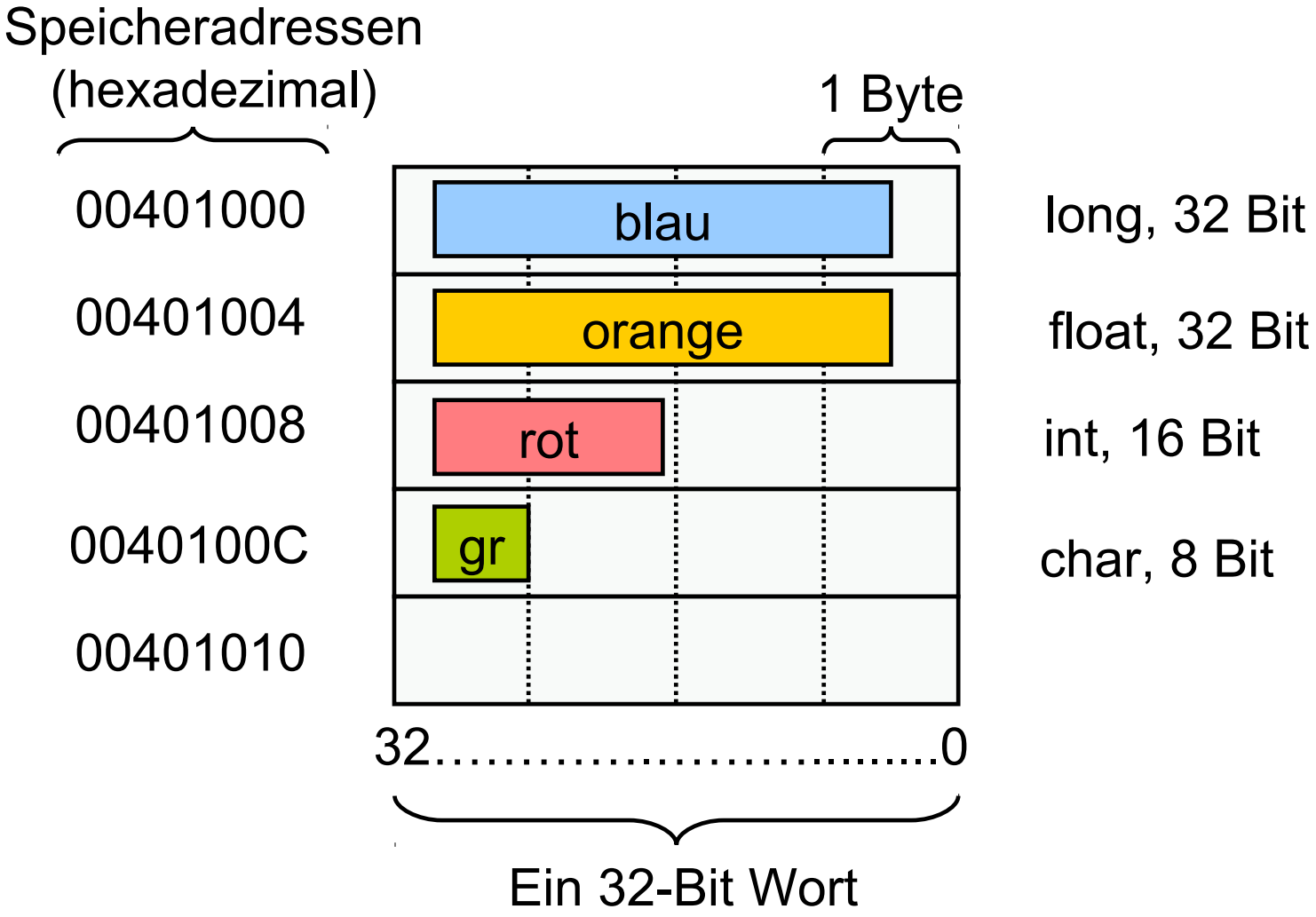
```
bool licht = {false};
double wert = {0.0};
int zahl = {0};
```

- Die Initialisierungsliste beginnt und endet mit den geschweiften Klammern.
- Bei Variablen hat die Initialisierungsliste immer nur ein Wert.
- Zwischen dem Bezeichner und der Initialisierungsliste kann ein Gleichheitszeichen stehen, muss aber nicht.

Speicherbedarf einer Variablen

- Festlegung durch den Datentyp.
- Der Speicherbedarf ist abhängig von dem implementierten Betriebssystems.

Beispiel: Speicherbedarf bei 32-Bit-System



Ermittlung des Speicherbedarfs

```
std::cout << "Speicherbedarf der Datentypen (Ganzzahlen):";
std::cout << "\n short: " << sizeof (short) << " Bytes";
std::cout << "\n int: " << sizeof (int) << " Bytes";
std::cout << "\n long: " << sizeof (long) << " Bytes";
std::cout << "\n long long: " << sizeof (long long) << " Bytes";
std::cout << std::endl;

std::cout << "\nSpeicherbedarf der Datentypen (Gleitkommazahlen):";
std::cout << "\n float: " << sizeof (float) << " Bytes";
std::cout << "\n double: " << sizeof (double) << " Bytes";
std::cout << "\n long double: " << sizeof (long double) << " Bytes";
std::cout << std::endl;
```

Nutzung von sizeof()

`sizeof`	`(`	`int`	`)`
`sizeof`	`(`	`Parameter`	`)`

- Mit Hilfe der Funktion `sizeof()` wird die Größe einer Variablen oder eines bestimmten Datentyps in Bytes ermittelt.

Bytes

- 1 Byte besteht aus 8 Bits.
- Darstellung von bis 256 (2^8) verschiedenen Zustände.
- Codierung von Buchstaben und Zahlen.

Bit

- Kleinste Informationseinheit.
- Darstellung eines Zustandes: Strom eingeschaltet oder ausgeschaltet.
- Binärziffer 1 oder 0.

Wertebereich einer Variablen

- Festlegung durch den Datentyp.
- Der Wertebereich wird durch die Größe des genutzten Speichers beeinflusst.
- Es gibt nur Regeln, aber keine festen Wertebereiche.

Wertebereich eines Datentyps

```
std::cout << "Wertebereich der Datentypen (Ganzzahlen):";
std::cout << "\n short: " << std::numeric_limits<short>::min()
                          << " - " << std::numeric_limits<short>::max();
std::cout << "\n int: " << std::numeric_limits<int>::min()
                          << " - " << std::numeric_limits<int>::max();
std::cout << "\n long: " << std::numeric_limits<long>::min()
                          << " - " << std::numeric_limits<long>::max();
std::cout << "\n long long: " << std::numeric_limits<long long>::min()
                          << " - " << std::numeric_limits<long long>::max();
std::cout << std::endl;

std::cout << "\nWertebereich der Datentypen (Gleitkommazahlen):";
std::cout << "\n float: " << std::numeric_limits<float>::min()
                         << " - " << std::numeric_limits<float>::max();
std::cout << "\n double: " << std::numeric_limits<double>::min()
                        << " - " << std::numeric_limits<double>::max();
std::cout << "\n long double: " << std::numeric_limits<long double>::min()
                        << " - " << std::numeric_limits<long double>::max();
std::cout << std::endl;
```

Template „numeric_limits“

`numeric_limits`	`<`	`int`	`>`
`template`	`<`	`datentyp`	`>`

- Die Vorlage `numeric_limits` ist in der Standard-Bibliothek `<limits>` definiert.
- Die Vorlage ermittelt den Wertebereich eines numerischen Datentyps.
- Templates ermöglichen eine generische Programmierung unabhängig vom Datentyp.

… wird angewendet auf

`numeric_limits`	`<`	`int`	`>`
`template`	`<`	`datentyp`	`>`

- In spitzen Klammern wird der gewünschte Datentyp angegeben.
- In diesem Beispiel wird die Vorlage auf den Datentyp `int` angewendet. Für diesen Datentyp werden die, in dem Template vorhandenen Methoden generiert.

Methode eines Templates

`numeric_limits`	`<`	`int`	`>`	`::`	`min`	`(`	`)`
`template`	`<`	`datentyp`	`>`	`::`	`methode`	`(`	`)`

- Methoden beschreiben eine Handlung in einer Programmiersprache. In diesem Beispiel „Gibt den minimalen Wert“ (`min`) zurück.
- Die Methode wird wie eine Funktion mit Hilfe ihres Namens aufgerufen.
- Dem Namen folgt die Parameterliste. Die runden Klammern enthalten keinen Parameter. Der Methode wird kein Parameter übergeben. Die Parameterliste ist leer.

… definiert in der Vorlage

`numeric_limits`	`<`	`int`	`>`	`::`	`min`	`(`	`)`
`template`	`<`	`datentyp`	`>`	`::`	`methode`	`(`	`)`

- Die Methode und die Vorlage werden mit dem Operator :: verbunden.
- Der Qualifizierungsoperator wird aus zwei Doppelpunkten zusammengesetzt.
- Links vom Operator wird der Name des Templates angegeben. Rechts vom Operator die Methode, die in dem angegebenen Template definiert ist.

Nutzung der C-Bibliothek

```
#include <climits>

int main () {
   cout << "\n\nWertebereich der Datentypen:\n";
   cout << "\n... short: " << SHRT_MIN << "..." << SHRT_MAX;
```

- Die Bibliothek `climits` muss am Anfang des Programms eingebunden werden.
- In der Bibliothek sind Konstanten für die Grenzen des Wertebereiches eines Datentyps definiert.
- Siehe: http://en.cppreference.com/w/cpp/header/climits.

Mögliche Fehlermeldung

```
char zeichen01 = 999;
char zeichen02 = {999};
```

- In Abhängigkeit des verwendeten Compilers und der Compiler-Version.
- Die erste Anweisung erzeugt zum Beispiel eine Warnung „warning: overflow in implicit constant conversion“.
- Die zweite Anweisung erzeugt zum Beispiel den Fehler „error: narrowing conversion of '999' from 'int' to 'char'“.
- In beiden Fällen ist der mögliche Bereich eines Zeichens (`char`) überschritten.

Vorzeichenbehaftete Ganzzahlen

```
int main () {
      signed   short nZahlMitVorzeichen;
```

- Zahlen vom Typ `int`, `long`, `short` und `char` können positiv oder negativ sein.
- Standardeinstellung für Ganzzahlen.
- Das Schlüsselwort `signed` wird selten genutzt.

Vorzeichenlose Ganzzahlen

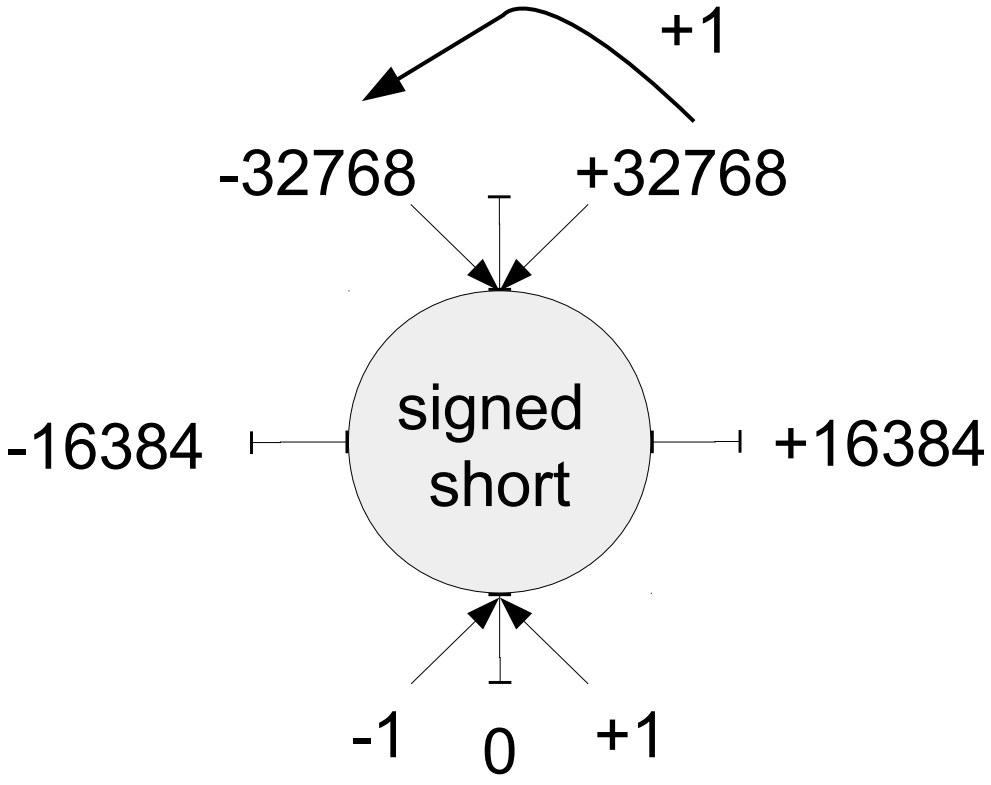
```
int main () {
      unsigned short nWertMitVorzeichen;
```

- Zahlen vom Typ `int`, `long`, `short` und `char` können nur positive Zahlen speichern.
- Kennzeichnung mit dem Schlüsselwort `unsigned`.
- Der Datentyp Typ `char` wird zur Codierung von Zeichen genutzt. Der Datentyp nutzt nur positive Zahlen.

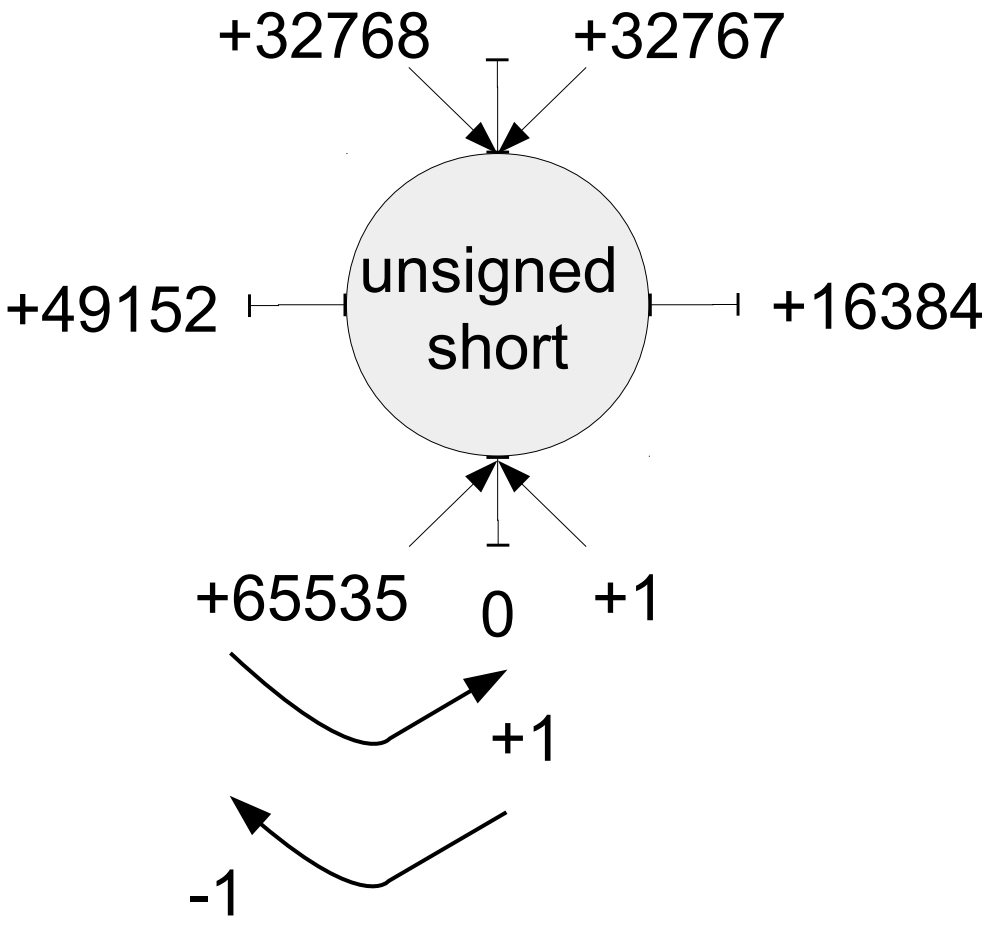
Arithmetischer Überlauf

- Das Ergebnis ist zu groß für den aktuellen Wertebereich.
- Abhängig vom gewählten Datentyp.
- Es wird kein Fehler bei der Kompilierung gemeldet.
- Integer overflow: Umbruch auf den maximalem Wert des Wertebereichs.

Beispiel short signed



Beispiel short unsigned



Genauigkeit einer Gleitkommazahl

```
#include <limits>

int main () {
  cout << "Genauigkeit der Datentypen:\n";
  cout << "\nfloat: " << numeric_limits<float>::digits10;
  cout << "\ndouble: " << numeric_limits<double>::digits10;
  return 0;
}
```

Template „numeric_limits“

`numeric_limits`	`<`	`int`	`>`
`template`	`<`	`datentyp`	`>`

- Die Vorlage `numeric_limits` ist in der Standard-Bibliothek `<limits>` definiert.
- Die Vorlage ermittelt den Wertebereich eines numerischen Datentyps.
- Templates ermöglichen eine generische Programmierung unabhängig vom Datentyp.

… wird angewendet auf

`numeric_limits`	`<`	`int`	`>`
`template`	`<`	`datentyp`	`>`

- In spitzen Klammern wird der gewünschte Datentyp angegeben.
- In diesem Beispiel wird die Vorlage auf den Datentyp `int` angewendet. Für diesen Datentyp werden die, in dem Template vorhandenen Methoden generiert.

Methode eines Templates

`numeric_limits`	`<`	`int`	`>`	`::`	`digits10`	`(`	`)`
`template`	`<`	`datentyp`	`>`	`::`	`methode`	`(`	`)`

- Methoden sind vordefinierte Codeblöcke. In diesem Beispiel werden Anweisungen für „Berechne die Genauigkeit einer Zahl aus dem Dezimalsystem“ (`digits10`) zusammengefasst.
- Die Methode wird wie eine Funktion mit Hilfe ihres Namens aufgerufen. Die Groß- und Kleinschreibung wird beachtet.
- Dem Namen folgt die Parameterliste. Die runden Klammern enthalten keinen Parameter. Es werden keine Parameter als Startwerte übergeben.

… definiert in der Vorlage

`numeric_limits`	`<`	`int`	`>`	`::`	`digits10`	`(`	`)`
`template`	`<`	`datentyp`	`>`	`::`	`methode`	`(`	`)`

- Die Methode und die Vorlage werden mit dem Operator :: verbunden.
- Der Qualifizierungsoperator wird aus zwei Doppelpunkten zusammengesetzt.
- Links vom Operator wird der Name des Templates angegeben. Rechts vom Operator die Methode, die in dem angegebenen Template definiert ist.

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'`	`;`

- Standard-Datentyp.
- 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;
```

- Codierung mit einer bestimmten Zeichentabelle.
- In der Zeichentabelle hat jedes Zeichen einen eindeutigen ganzzahligen Wert.
- Standardzeichentabelle in C++: ASCII-Zeichensatz

ASCII-Zeichensatz

- American Standard Code for Information Interchange.
- Definition von 128 Zeichen.
- Umlaute wie ä, ö, ü oder Sonderzeichen wie zum Beispiel das Euro-Zeichen sind in diesem Zeichensatz nicht vorhanden.
- Siehe http://www.torsten-horn.de/techdocs/ascii.htm

„Maskierung“ von ...

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

- Steuerzeichen für den Drucker etc.
- nicht druckbaren Zeichen eines Zeichensatzes.
- Zeichen, die in C++ in einer besonderen Funktion genutzt werden.
- Zeichen aus dem Unicode-Zeichensatz.

Erläuterung

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

- Maskierte Zeichen werden als Escape-Sequenzen bezeichnet.
- Maskierung mit Hilfe des umgekehrten Schrägstriches (Backslash).

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“.
- Nutzung von `'\0'`: Ende-Zeichen eines Strings (Zeichenkette). Leerer String.
- Nutzung von `' '`: Zwischen den beiden Apostrophe wird ein Leerzeichen eingefügt.

Neue Zeile

```
const char newline = '\n';
std::cout << std::endl;
```

- Die Escape-Sequenz `\n` erzeugt einen Zeilenumbruch.
- In der Bibliothek `iostream` ist die Konstante `endl` im Standard-Namensraum definiert. Die Konstante erzeugt einen Zeilenumbruch und erzwingt eine Ausgabe.

Ausgabe von Variablen

```
std::cout << lZahl << '+' << rZahl;
std::cout << " = " << (lZahl + rZahl);
```

- `cout` gibt Variablen, Konstanten etc. auf der Standardausgabe aus.
- Der Befehl ist in der Bibliothek iostream definiert. Durch die Präprozessor-Anweisung `#include <iostream>` wird die Bibliothek eingebunden.

Definition der Standard-Ausgabe

`std`	`::`	`cout`

- `cout` ist ein Synonym für die Standard-Ausgabe. Bei Konsolenprogrammen wird eine Konsole oder Terminal als Ausgabe genutzt.
- Jedes Element aus einer Bibliothek ist in einem Namensraum abgelegt. Durch die Zusammenfassung von Elementen in Räumen werden Namenskonflikte vermieden. `cout` ist in dem Standard-Namensraum `std` abgelegt.
- Namensraum und Element werden durch den Qualifizierungsoperator `::` verbunden.

Umleitung auf die Standardausgabe

`std`	`::`	cout	`<<`	lZahl

- Umleitungsoperator: <<
- Die Pfeilspitze zeigt die Richtung der Umleitung an.
- Die Kleiner-Zeichen leiten den Wert der Variablen auf die Standardausgabe um.
- Nutzung in einer Anweisung: Beliebig oft.

Hinweise zur MS DOS Eingabeaufforderung

- Der Befehl `dir` listet alle Elemente in einem Ordner auf.
- Mit Hilfe von `cd` kann der Ordner gewechselt werden. `cd codelite` wechselt in den Ordner codelite in dem angegebenen Ordnern. `cd ..` geht einen Ordner nach oben.
- Der Befehl `cmd.exe /k chcp 1252` öffnet die Eingabeaufforderung mit der Codepage 1252
- Der Befehl `chcp 65001` ändert die Codepage der aktuellen Eingabeaufforderung.
- Liste von Codepages: https://docs.microsoft.com/de-de/windows/desktop/Intl/code-page-identifiers

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.