Software-Technik: Vom Programmierer zur erfolgreichen …

1. Von der Idee zur Software
2. Funktionen und Datenstrukturen
3. Organisation des Quellcodes
4. Werte- und Referenzsemantik
5. Entwurf von Algorithmen
6. Fehlersuche und –behandlung
7. Software-Entwicklung im Team
8. Abstrakte Datentypen: Einheit von Daten und Funktionalität
9. Vielgestaltigkeit (Polymorphie)
10. Entwurfsprinzipien für Software

Anhang A: Die Familie der C-Sprachen

Anhang B: Grundlagen der C++ und der Java-Programmierung

Allgemeine Bemerkungen zu C/C++

- C ist im Gegensatz zu Java keine objektorientierte, sondern eine klassische prozedurale Sprache.
- C++ ist eine Erweiterung von C, sie ist eine hybride Sprache, die sowohl die klassische prozedurale als auch die objektorientierte Programmierung ermöglicht.
- In C gibt es keine Klassen mit Attributen und Methoden sondern stattdessen Variable und Funktionen (Unterprogramme, Prozeduren).
- Im Gegensatz zu Java unterstützt C++ verschiedene Paradigmen (Muster, Methoden) der Programmierung und sie unterstützt verschiedene Abstraktionsebenen. Das Erlernen von C/C++ hat insbesondere für den Informatiker den Vorteil, dass er verschiedene Paradigmen und verschiedene Abstraktionsebenen kennen lernt.

Grundlagen der C++ und der Java-Programmierung

Kompendium, 3. Auflage: 1.4-1.5
Kompendium, 4. Auflage: 1.3
Lehrbuch: 1.2

Trennzeichen (White Spaces)

Trennzeichen der Sprache C/C++ sind:

Zwischenraum (space, blank), horizontaler Tabulator, neue Zeile
vertikaler Tabulator, Seitenvorschub (form feed)

sowie alle anderen Sonderzeichen, die in der Sprache erlaubt sind, wie:

/ & % (= usw.

d.h. alle Zeichen außer *Ziffern*, *Buchstaben* und *Unterstrich* (_).
Leerzeichen und *Zeilenumbrüche* dürfen in C / C++ an beliebiger Stelle außer in Namen stehen:

```
int var_i=17;
int var_i    =                    17;
int
   var_i
=              17;

int var_     i=17;  // ist aber verboten
```

Kommentare

Zur Kommentierung gibt es -- in C++ die Zeichenfolge „//“
(sog. Inline-Kommentare). Alles ab hier, bis zum Ende der Zeile wird vom Compiler als Kommentar betrachtet:

```
int anzahl = 0;    // Zähler für die Anzahl der Häuser,
                   // die ein weißes Dach haben
int i;
```

Bei den Kommentarzeichen “/*” wird alles bis zu den Kommentarende-zeichen “*/” als Kommentar vom Compiler überlesen.

```
int anzahl = 0;     /* Zähler für die Anzahl der
                       Häuser, die ein weißes Dach haben */
int i;
```

Kommentare in Kommentaren

Kommentare der Form „/*...*/“ können nicht geschachtelt werden:

```
/*
int anzahl = 0;  /* Zähler für die Anzahl der
                    Häuser, die ein weißes Dach haben */
int i;
*/
```

falsch !

Inline-Kommentare können dagegen in Kommentaren der Form „/*...*/“ geschachtelt werden:

```
/*
int anzahl = 0;     // Zähler für die Anzahl der Häuser,
                    // die ein weißes Dach haben
int i;
*/
```

richtig !

Kommentare in Kommentaren (2)

Besser ist zur Auskommentierung die Präprozessor-Anweisungen *#ifdef ... #endif* zu verwenden:

```
#ifdef  IRGENDETWAS_UNDEFINIERTES
int anzahl = 0;     // Zähler für die Anzahl der Häuser,
                    // die ein weißes Dach haben
int i;
#endif
```

funktioniert in beiden Fällen

```
#ifdef  IRGENDETWAS_UNDEFINIERTES
int anzahl = 0;     /*Zähler für die Anzahl der Häuser,
                     die ein weißes Dach haben  */
int i;
#endif
```

Grundlagen der C++ und der Java-Programmierung

Kompendium, 3. Auflage: 1.4-1.5
Kompendium, 4. Auflage: 1.3
Lehrbuch: 1.2

Grundelemente der Sprache (Lexikalische Elemente, Symbole, Token)

Schlüsselwörter (reservierte Wörter),
Bezeichner (Namen),
Konstanten,
Zeichenfolgen (String-Literale),
Operatoren,
Interpunktionszeichen (Punktuatoren)

Bezeichner (Namen für Variable, Funktionen, ...) (2)

Namen beginnen mit einem Buchstaben oder einem Unterstrich.
Die weiteren Zeichen sind Buchstaben, Zahlen oder Unterstriche.
Schlüsselworte der Sprache und dürfen nicht als Name verwendet werden:

```
a;
_pointer;
ganz_langer_name_24_undNochLaenger;
name; Name;    // Groß- und Kleinschreibung wird unterschieden
34Name;                  // Fehler Zahl am Anfang nicht erlaubt
Strassen Name;           // Leerzeichen nicht erlaubt: Strassen_Name
Ölinhalt;                // Fehler, Umlaute verboten
C&A;                     // Fehler;  Sonderzeichen verboten
```

Variablen müssen vor ihrer ersten Benutzung auf jeden Fall deklariert worden sein.

Elementare (vordefinierte) Datentypen

Elementare Datentypen in C und C++, Teil I

		Wertebereich	Genauigkeit
char = signed char **unsigned char**	1 Byte 1 Byte	-2^7 ... +2^7-1 0 ... 2^8-1	2 Dez.-stellen
int = signed int **unsigned int**	4 Byte 4 Byte	-2^31 ... +2^31-1 0 ... 2^32-1	9 Dez.-stellen
short int = signed short int **unsigned short int**	2 Byte 2 Byte	-2^15 ... +2^15-1 0 ... 2^16-1	4 Dez.-stellen
long int = signed long int **unsigned long int**	4 Byte 4 Byte	-2^31 ... +2^31-1 0 ... 2^32-1	9 Dez.-stellen
bool	1 Byte	{ false, true }	nur C++
wchar_t	2 Byte		nur C++

Die angegebenen Werte stellen Beispiele für 32-Bit-Umgebungen dar, sie sind implementierungsabhängig.
In C99 gibt es noch long long.

überspringen

Elementare Datentypen in C und C++, Teil II

		Wertebereich	Genauigkeit
enum { *list* } **enum *id* { *list* }** **enum *id***	4 Byte 4 Byte 4 Byte	wie int	
float **double** **long double**	4 Byte 8 Byte 10 Byte	ca. -10^38 ... +10^38 ca. –10^308 ... +10^308 ca. -10^4932 ... +10^4932	7 Dez.-Stellen 15 Dez.-Stellen 19 Dez.-Stellen
*type** **void***	4 Byte 4 Byte	0...2^32-1 0...2^32-1	
type&	4 Byte	0...2^32-1	

Die angegebenen Werte stellen Beispiele für 32-Bit-Umgebungen dar, sie sind implementierungsabhängig.

überspringen

Clicker-“Abstimmung“

Sie wollen die Anzahl der Hörer dieser Vorlesung verwalten.
(Viel mehr werden wir nicht). Was verwenden Sie?

1. unsigned int □
2. int □
3. short □
4. double □

Sie wollen den Umfang eines Kreises berechnen und damit weiter arbeiten. Was verwenden Sie?

1. unsigned int □
2. float □
3. double □
4. long double □

Datentypen

Für ganzzahlige Variablen wird im Allgemeinen int verwendet.
Für Gleitkomma Variablen wird im Allgemeinen double benutzt.

Für Boole‘sche Werte gibt es den Datentyp bool mit den Konstanten true und false.
Zur Verarbeitung von ASCII-Zeichen dient der Datentyp char.
Für Zeichenketten wird der Datentyp string aus der C++-Standardbibliothek verwendet.

```
int i = -64;
double  d = 64.3345;
double  d1 = 1.234e-22;          // ohne Leerzeichen zu schreiben
double x = 5.;                   // 5. entspricht 5.0
bool gefunden = true;
bool ende = false;
char c = 'X';
string worte = "hier bin ich";
```

Konstanten

Die Verwendung von Konstanten erhöht die Lesbarkeit eines Programms,
Statt

```
umfang = 2 * radius * 3.14159;
```

besser

```
const double PI = 3.14159;
. . .
umfang = 2 * radius * PI;
```

Weitere Beispiele:

```
const double UmfangMeinKreis = 2.0 * radius * PI;      // radius muss keine
                                                       // Konstante sein
const double sinPIviertel = sin(PI / 4.0);
```

const heißt nur, dass die Variable nicht mehr verändert werden darf, der Wert muss nicht schon zur Compile-Zeit bestimmt werden können.

Konstanten (2)

Guter Programmierstil ist es außer den Konstanten –1, 0 und 1 keine expliziten Zahlenkonstanten in seinem Programm zu verwenden, sondern diese immer über *const* einen Namen zuzuweisen,
Statt

```
for (int i=0; i < 10; ++i) ...
```

also besser

```
const int ANZAHL_DER_MITSPIELER = 10;
. . .
for (int i=0; i < ANZAHL_DER_MITSPIELER; ++i) ...
```

Zeichenkonstanten:

Bei Zeichenkonstanten wird das Zeichen in einfachen Anführungszeichen angegeben.

```
'a'     '2'     '.'     ' '   // Leerzeichen
```

Spezielle Zeichenkonstanten:

```
  /* Zeilenvorschub:                 */   '\n'
  /* Horizontaler Tabulator:         */   '\t'
  /* Vertikaler Tabulator:           */   '\v'
  /* Backspace:                      */   '\b'
  /* Carriage-Return:                */   '\r'
  /* Form-Feed:                      */   '\f'
  /* Alarm:                          */   '\a'
  /* Backslash:                      */   '\\'
  /* Fragezeichen:                   */   '\?'
  /* Einfaches Anfuehrungszeichen:   */   '\''
  /* Doppeltes Anfuehrungszeichen:   */   '\"'
```

Aufzählungstypen

Manchmal ist es sinnvoll, eine Vielzahl gleichartiger Konstanten zu einer Menge zusammenzufassen. Jedes Element dieser Menge bekommt eine Ordinalzahl Das erste Element erhält z.B. die Ordinalzahl 0, das zweite 1 usw., Beispiel:

```
enum Wochentag { Mon, Die, Mit, Don, Fri, Sam, Son };
Wochentag Tag;
Tag = Mon;
```

Syntax von Aufzählungen:

enum *AufzTyp* **{** *Bezeichner1, Bezeichner2, ...* **}** *Variable*;
enum { *Bezeichner1, Bezeichner2, ...* **}** *Variable*;
enum *AufzTyp* **{** *Bezeichner1* = 2, *Bezeichner2,* ... **};**

Aufzählungstypen (2)

Wird hinter den Bezeichnern keine Initialisierung angegeben, erhalten diese den Wert des Vorgängers vermehrt um eins.

Man kann jedoch die Bezeichner explizit initialisieren, indem man dem Bezeichner ein Gleichheitszeichen und einen Ausdruck anfügt, der einen Integer - Typ zurückgibt.

Es ist auch möglich, mehreren Bezeichnern denselben Wert zuzuordnen.

Man kann gleich eine oder mehrere Variablen des Aufzählungstyps am Ende definieren

überspringen

Aufzählungstypen (3)

```
enum Reihe { Zahl1=30, Zahl2=Zahl1+5,
                       Zahl3=Zahl2*2, Zahl4=1, Zahl5 };
```

Eine alternative (aber umständlichere) Schreibweise hierfür ist

```
const int Zahl1 = 30;
const int Zahl2 = 35;
const int Zahl3 = 70;
const int Zahl4 = 1;
const int Zahl5 = 2;
```

Kompatibilitäten:

```
int i;
i = Zahl1;  /* zulässig, da Zahl1 vom Typ Integer ist */

Zahl1 = i;  /* nicht zulässig, da Zahl1 ein konstanter Wert ist */
```


überspringen

Grundlagen der C++ und der Java-Programmierung

Kompendium, 3. Auflage: 1.9
Kompendium, 4. Auflage: 1.7
Lehrbuch: 1.7

Ein- und Ausgabe, 1. Teil

Die iostream-Bibliothek definiert die Ein- und Ausgabe für jeden eingebauten Typ, wie z.B. char, int, double etc.

```
int i = 10;
cout << "i hat den Wert " << i << "\n";
```

```
i hat den Wert 10
```

Obiges Programm ist äquivalent zu:

```
int i = 10;
cout << "i hat den Wert ";
cout << i ;
cout << "\n";
```

Damit dieses funktioniert, muss <iostream> zuvor eingebunden werden.
#include <iostream>
using namespace std;
Das "c" in cout steht für character.

Formatierte Ausgabe und Ausgabe in eine Datei

```
cout.width(4);      // 4 spaltige Ausgabe für die nächste Ausgabe
cout << anzahl;     // Ausgabe in vier Spalten, wenn möglich
cout << anzahl;     // Ausgabe wieder soviel wie nötig
```

Ausgabe in eine Datei

```
#include <fstream>
using namespace std;
```

Anlegen einer Datei:

```
ofstream  strname("dateiname.txt",ios::out);
strname << "Text" << i << endl;
strname.width(7);
strname << i;
```

strname wird also genau wie cout verwendet.

Eingabe

Für die Eingabe wird anstatt "cout" "cin" verwendet.
Das "c" in "cin" steht für chararcter, also Zeicheneingabe.
Die Eingabe wird erst an das Programm gesendet, nachdem die Returntaste betätigt wurde.

Beispiel:

```
int i;
int j;
double d;

cin >>  i   >>   d >> j;  // Eingabe von einem int, einem double und
                          // einen int-Wert
```

entspricht:

```
cin >> i;
cin >> d;
cin >> j;
```

Grundlagen der C++ und der Java-Programmierung

Kompendium, 3. Auflage: 1.4
Kompendium, 4. Auflage: 1.3
Lehrbuch: 1.2

Binäre Operatoren

`+`
`-`
`*`
`/`
`%`
	Addition
	Subtraktion
	Multiplikation
	Division
	Divisionsrest
		Arithmetik	Zahlen, mit Einschr. Adressen
			
			nur ganze Zahlen

`<`
`<=`
`==`
`!=`
`>=`
`>`
	Vergl. auf *kleiner*
	Vergl. auf *kleiner oder gleich*
	Vergl. auf *gleich*
	Vergl. auf *ungleich*
	Vergl. auf *größer oder gleich*
	Vergl. auf *größer*
		Vergleich	alle Typen

`&`
`|`
`^`
`<<`
`>>`
	bitw. *UND*-Verknüpfung
	bitw. *ODER*-Verknüpfung
	bitw. *Eckl.-Oder*-Verknüpfung
	bitw. Linksverschieben
	bitw. Rechtsverschieben
		Bitoperation	ganzz. Typen

`&&` `||`	log. *UND*-Verknüpfung log. *ODER*-Verknüpfung	Logische Verknüpfung	Boolesche Werte

Unäre Operatoren, Postfix- und Präfix-Operatoren

& *****	Adresse von Inhalt von	Referenzierung Dereferenzier.	alle Typen Zeiger
+ **–**	pos. Vorzeichen neg. Vorzeichen	Arithmetik	Zahlen
~	bitw. Invertieren	Bitoperation	ganzz. Typen
!	log. Invertieren	Log. Verkn.	Bool. Werte
(*type*) **static_cast** **const_cast** **reinterpret_cast**	C-Allzweck-Cast Spezialisierte Konvertierungen in C++	in C und C++ nur in C++	viele Typen spezi- elle Typen
sizeof **sizeof**	sizeof *expr.*: Speicherbedarf sizeof (type): Speicherbedarf		Ausdrücke Typen
++ **––**	Inkrementierung Dekrementierung	Postfix und Präfix	ganzz. Typen und Zeiger

Zuweisungsoperatoren und sonstige Operatoren

`=`	Wertzuweisung	Zuweisung	alle Typen

`+=`
`-=`
`*=`
`/=`
`%=`
	Addition
	Subtraktion
	Multiplikation
	Division
	Divisionsrest
		Arithmetik
		und
		Zuweisung
			Zahlen, mit
			Einschr. Adressen
			
			nur ganze Zahlen

`&=`
`|=`
`^=`
`<<=`
`>>=`
	bitw. *UND*-Verknüpfung
	bitw. *ODER*-Verknüpfung
	bitw. *Eckl.-Oder*-Verknüpfung
	bitw. Linksverschieben
	bitw. Rechtsverschieben
		Bitoperationen
		und
		Zuweisung
			ganzz. Typen

`?:` `,`	Formulierung bed. Ausdrücke Aufzählung in Klammerausdr.		Ausdrücke Ausdrücke

Arithmetische Operatoren

Einfache Zuweisung =

```
summe = 64;
```

Addition + bzw. +=

```
summe = zahl + 61;
summe = summe + 4;
summe += 4;
```

Subtraktion - bzw. -=
Multiplikation * bzw. *=
Division / bzw. /=
Modulo-Operator % bzw. %=

und weitere Operatoren ...

Jede Anweisung wird durch ein Semikolon abgeschlossen.

Initialisierung von Variablen in C und C++

Definition und Initialisierung von Variablen:

```
int i = 7;  /* Speicherplatz wird reserviert, der mit 7 initialisiert wird */
```

Zuweisung an Variablen:

```
i = 8;      /* Vorhandener Speicherplatz wird mit neuem Wert belegt.*/
```

Gleichwertig zur obigen Definition ist in C++:

```
int i(7);   // sollte bei OOP bevorzugt verwendet werden
```

Bei der Zuweisung geht diese Zuweisung dann aber **nicht**

```
i(8);       // ist syntaktisch falsch
```

Vergleichsoperatoren

- gleich ==
- ungleich !=
- kleiner <
- kleiner gleich <=
- größer >
- größer gleich >=

Beispiele:

```
if (zahl < 64) ...
if (zahl == 16) ...
if ((zahl >= 0)  && (zahl <= 64)) ...  // zahl aus dem Bereich [0..64]
bool bo = 5 > 7;                       // bo erhaelt den Wert false
int a = 5 > 7;                         // a erhält den Wert 0
int b = 5 < 7;                         // b erhält den Wert 1
```

Logische Verknüpfungsoperatoren

- Logisches Und &&
- Logisches Oder ||
- Logisches Nicht !

Beispiel:

```
if ( (!(zahl < 0)) && (!(zahl > 64))) ...  // Alle drei Abfragen sind gleich-
if ( (zahl >= 0)  && (zahl <= 64) ) ...  // bedeutend, sie testen, ob zahl
if ( zahl >= 0 && zahl <= 64 ) ...       // in dem Bereich 0..64  liegt
```

Achtung:

```
if (0 <= x <= 99)
```

entspricht **nicht** dem Erwarteten!!!, sondern

```
( (0 <= x)   <= 99 )
( false/true <= 99 )  // d.h. 0 <= x wird false (->0) oder true (->1)
```

Clicker-“Abstimmung“

```
int zahl1 = 4;
int zahl2 = ++zahl1;
cout << zahl1 << " ," << zahl2 <<" , ";

int zahl3 = 14;
int zahl4 = zahl3++;
cout << zahl3 << " ," << zahl4;
```

```
int zahl1 = 4;
++zahl1;
int zahl2 = zahl1;
cout << zahl1 << " ," << zahl2 <<" , ";
int zahl3 = 14;
int zahl4 = zahl3;
++zahl3;
cout << zahl3 << " ," << zahl4;
```

Was ist die Ausgabe des obigen Programms auf dem Bildschirm?

1. 5 ,5 , 15 ,15
2. 5 ,5 , 15 ,14
3. 5 ,4 , 15 ,15
4. 5 ,4 , 15 ,14

Ergebnis:

___ 5,5, 15,15 __ 5,5, 15,14, __ 5,4, 15,15 __ 5,4, 15,14

Inkrement und Dekrement

```
zahl = zahl + 1;  // kann verkürzt werden zu
zahl += 1;        // und nochmals zu
zahl++;
```

Entsprechend gibt es *zahl--;* für das Erniedrigen von zahl um 1.

Beachte:
„zahl++;“ und „*++zahl;“* sind nicht unbedingt das gleiche.

Beispiel:

```
int zahl1 = 64;

int zahl2 = ++zahl1;      // zahl2 und zahl1 sind nun beide 65

int zahl3 = zahl2++;      // zahl3 ist 65 und zahl2 66.
```

Prä-Increment und Post-Increment als Funktionen

```
int zahl2 = ++zahl1;

// entspricht
int zahl2 =PreInc(zahl1);

// mit
int PreInc(int& i) {
   i = i + 1;
   return i;
}
```

int&:
Übergabe eines int-Parametes als Call By Reference

```
int zahl2 = zahl1++;

// entspricht
int zahl2 =PostInc(zahl1);

// mit
int PostInc(int& i) {
   int temp = i;
   i = i + 1;
   return temp;
}
```

Grundlagen der C++ und der Java-Programmierung

Kompendium, 3. Auflage: 1.4
Kompendium, 4. Auflage: 1.3
Lehrbuch: 1.2

Explizite Typkonvertierungen

In einigen Fällen ist eine explizite Typkonvertierung unbedingt erforderlich:

```
int a = 5;
int b = 2;

// In C/C++
double x = ((double) a ) / ((double) b);     // x ist 2.5, sonst 2.0

// In C++ auch erlaubt:
double x = double(a) / double(b)

// In C++ empfehlenswert:
double x = static_cast<double>(a)  /  static_cast<double>(b);
```

Neben static_cast gibt es in C++ noch weitere Konvertierungs-operatoren, auf die an späterer Stelle noch eingegangen wird.

Grundlagen der C++ und der Java-Programmierung

Kompendium, 3. Auflage: 1.4
Kompendium, 4. Auflage: 1.3
Lehrbuch: 1.2

Prioritäten der Operatoren

1	`! ~ ++ -- + - * & sizeof (type)`	unär, postfix, präfix
2 3	`* / %` `+ -`	binäre arithmetische Operatoren
4	`<< >>`	Shift-Operatoren
5 6	`< <= > >=` `== !=`	Vergleichs-operatoren

7
8
9
	`&`
	`^`
	`|`
		Bitoperationen

10 11	`&&` `||`	Logische Verknüpfungen
12	`= += -= *= /= %= &= = |= <<= >>=`	Zuweisungsoperatoren
13	`,`	Kommaoperator

1 ist die höchste, 13 die niedrigste Priorität.

Prioritäten von Operatoren (2)

Beispiel:

```
// ohne zus. Klammern
int a, b, c, d, y1, y2, y3;

y1 = a < b && !(c == d);
y2 = a & ~b | ~(c | d);
y3 = a << b & c >> a;
```

```
// mit zus. Klammern
int a, b, c, d, y1, y2, y3;

y1 = (a < b) && (!(c == d));
y2 = (a & (~b)) | (~(c | d));
y3 = (a << b) & (c >> a);
```

Grundsätzlich sollte man lieber eine (oder 10) Klammern zu viel verwenden als eine zu wenig.

Und das nicht nur im Zweifelsfall, weil derjenige, der das Programm warten muss, vielleicht im Zweifel ist!

Grundlagen der C++ und der Java-Programmierung

Kompendium, 3. Auflage: 1.4
Kompendium, 4. Auflage: 1.3
Lehrbuch: 1.2

Clicker-“Abstimmung“

```
if (0.3 + 0.3  == 0.6)
   {cout << "Then-Teil“;}
else {cout << "Else-Teil“;}
```

Was ist die Ausgabe des obigen Programms auf dem Bildschirm?

1. Then-Teil
2. Else-Teil
3. Compiler und/oder Rechner-abhängig
4. Then-Teil (Leerzeile) Else-Teil

Rechnerdarstellung einer Gleitkommazahl

```
if (0.3 + 0.3  == 0.6)  { /* nein */  }
else {  /* häufig ist der else Zweig dran */  }
```

0.6 ist im Binärsystem:

1001 1001 1001 1001 1001 1001 1001 1001 ...

32 Bits (**Mantisse** genannt)

Dieses bedeutet:

1* 1/2 + 0 * 1/4 + 0 *1/8 + 1 * 1/ 16 + 1* 1/ 32 ... 1*1/ 268435456

Dieses ergibt: 0,59999999776482500000

0.6 ist im Dualsystem somit periodisch und nicht exakt darstellbar, sodass man Gleitkommazahlen nie auf Gleichheit testen sollte!

Wie sollte man dann aber einen Gleitkomma-Vergleich programmieren?

gefährlich

```
double d = 59.4;
… // ganz viel Rechnung mit d
if ( d == 66.3 ) { … }
```

Schon besser

```
double d = 59.4;
… // ganz viel Rechnung mit d
if ( fabs(d - 66.3) < 0.000001 ) { … }
```

Noch besser

```
double d = 59.4;
… // ganz viel Rechnung mit d
const double eps = 0.000001;
if ( fabs(d - 66.3) < eps ) { … }
```

Spitze

```
const double eps = 0.000001;
bool doublesEqual(
   double d1, double d2) {
 return  fabs(d1 – d2) < eps );
}
…
double d = 59.4;
… // ganz viel Rechnung mit d
if ( doublesEqual(d, 66.3 ) { … }
```

Visual Studio 2017 / 2019 erklären

Erstellung eines Projektes vorführen