

# C++ Teil 1

Sven Groß



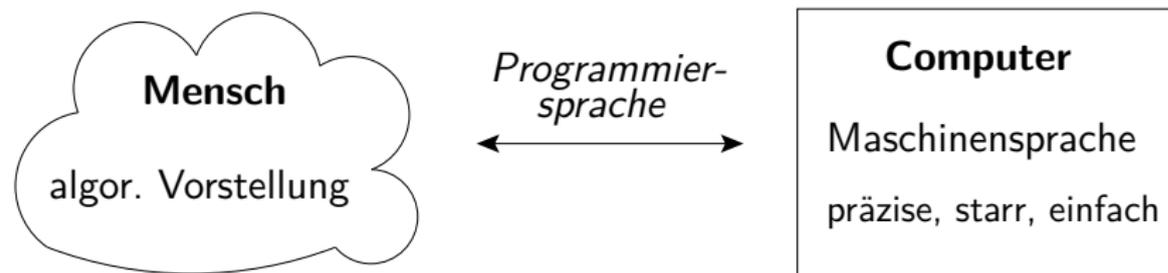
10. Okt 2017

- Anmeldung zum Praktikum (LV+Prüfung) über RWTHOnline **heute!**  
↪ Zugang zu L2P-Lernraum, Einteilung in Gruppen A1–B2
- Vorlesung: Folien einige Tage vorher im L2P  
↪ ausdrucken für eigene Notizen
- Übung/Praktikum: Bearbeiten von Programmieraufgaben  
↪ nur hier lernt ihr Programmieren!
- Prüfungsmodalitäten:
  - Anwesenheitspflicht beim Praktikum
  - Bestehen von 3 Minitests
  - Regelmäßige Online-Tests (Bonuspunkte)
- Entwicklungsumgebung: `Code::Blocks`, auf eigenem Rechner installieren  
↪ auch zuhause Programmiererfahrung sammeln
- Infos auf der Webseite und im L2P: [www.igpm.rwth-aachen.de/cpp](http://www.igpm.rwth-aachen.de/cpp)

# Heutige Themen

- 1 Unser erstes C++-Programm
- 2 Kompilieren, Programmstart, Debuggen
- 3 Elementare Datentypen
- 4 Ein-/Ausgabe
- 5 Operatoren
- 6 Kontrollstrukturen
  - Verzweigungen

# Wozu Programmiersprachen?



- Ziel: Algorithmus (z.B. Gauß-Algorithmus) auf Computer umsetzen
- *Programmiersprache* (FORTRAN, C, C++, Java...) als verständliche, unterstützende, abstrahierende Darstellungsform
- Übersetzung von Programmiersprache in Maschinensprache durch entsprechenden *Compiler*

notwendig:

- Daten (Variablen)
- Ein-/Ausgabe
- Operatoren (Arithmetik, Vergleich, ...)
- Kontrollstrukturen (Verzweigungen, Schleifen, ...)

hilfreich:

- Kommentare
- Präprozessor (`#include <iostream>`)
- Strukturierung durch Funktionen, Klassen, ...

# Unser erstes C++-Programm: Hello World

Quellcode in Datei `hello_world.cpp`:

```
#include <iostream>
using std::cout;
using std::endl;

int main()
{
    // unser erstes kleines Programm

    /* cout fuer Bildschirmausgabe
       endl fuer Zeilenumbruch
    */

    cout << "Hello_world!" << endl; // Bildschirmausgabe

    return 0;                        // Fehlercode 0 = alles ok
}
```

- Jedes C++-Programm beginnt in der Funktion `main`.
- Jede **Anweisung** wird mit `;` abgeschlossen.
- **Blöcke** von Anweisungen werden durch `{ }` eingeschlossen.
- **Kommentare:**
  - wichtig für Programmierer (Code verstehen, Dokumentation), irrelevant für den Compiler
  - `//` leitet eine Kommentarzeile ein (alles danach wird ignoriert)
  - mit `/* ... */` wird ein Kommentarblock markiert (alles dazwischen wird ignoriert)
- `#include<iostream>` fügt die Header-Datei `iostream` ein (*input/output stream*), definiert z.B. `std::cin` und `std::cout` zur Ein- und Ausgabe.
- **Namensraum** (*namespace*) `std` enthält zahlreiche Elemente der Standard-C++-Bibliothek.

# Programm übersetzen, aufrufen und debuggen

- Quellcode im Editor schreiben und abspeichern → `prog.cpp`
  - wesentliche Schritte beim Übersetzen von C++-Code:
    - **Präprozessor**: reine Textersetzung (z.B. `#include <iostream>`, Kommentare, Makros)
    - **Kompilieren**: erzeugt *Objektdateien*  
`g++ -c prog.cpp` → `prog.o`
    - **Linken**: führt Objektdateien und Bibliotheken zu *ausführbarem Programm* zusammen  
`g++ prog.o unit1.o -o myProg` → `myProg`
  - Programm ausführen: `./myProg`
  - Fehlersuche mit Debugger, z.B. `gdb myProg`  
↔ Programm mit Option `-g` übersetzen
- ⇒ integrierte Entwicklungsumgebung (IDE) unterstützt all dies, z.B. `Code::Blocks`

notwendig:

- Daten (Variablen)
- Operatoren (Arithmetik, Ein-/Ausgabe, ...)
- Kontrollstrukturen (Verzweigungen, Schleifen, ...)

hilfreich:

- Kommentare
- Präprozessor (`#include <iostream>`)
- Strukturierung durch Funktionen, Klassen, ...

## Elementare Datentypen in C++

ganze Zahlen	<code>int</code>	<code>int j = -3;</code>
reelle Zahlen	<code>double</code>	<code>double z = 3.1415;</code>
Zeichenketten	<code>std::string</code>	<code>std::string s = "Hello_World!";</code>
	<code>char[]</code>	<code>char s[] = "Hello_World!";</code>
logische Werte	<code>bool</code>	<code>bool bestanden = true;</code>
...		

- weitere Datentypen in der Kurzanleitung
- Jede Variable muß vor Benutzung mit *Typ Name*; **deklariert** werden:

```
int k;
```

↪ **Problem:** `k` kann beliebigen Wert enthalten (Datenmüll)

- *besser:* **Initialisierung** mit *Typ Name=Wert*; bzw. *Typ Name(Wert)*;

```
int k= 7;
```

# Beispiele zur Variablendeklaration

```
int zahl= 1, Zahl= 100;    // unterschiedl. Variablen!  
  
double x, y= -1.3E-2,     // x nicht init., y = -0.013,  
       z= 2e3;            // z = 2000  
  
std::string Name1= "Donald_Duck", // Zeichenkette  
            Name2= "Micky_Mouse";  
  
bool hat_bestanden= true, // Wahrheitswerte  
    istSchlau= false;
```

## Merke:

- aussagekräftige Variablennamen verwenden!
- Variablen bei Deklaration auch initialisieren!
- Groß-/Kleinschreibung beachten!

# Gültigkeitsbereich (*scope*) von Variablen

- Jede Variable ist nur bis zum Ende des Blockes {...} gültig, in dem sie erzeugt wurde (**scope**).

```
{
    int k= 7;  // ab hier ist k definiert
    ...
    {
        ...
        double x= 1.23;  // ab hier ist x definiert
        ...
        std::cout << k;
    }           // scope von x endet hier
    ...
    std::cout << x;    // FEHLER: x nicht definiert
    ...
}              // scope von k endet hier
```

- Verlässt eine Variable ihren Scope, wird sie automatisch zerstört.

- benötigt Header `iostream`: `#include<iostream>`
- `using std::cout`; etc., um `cout` statt `std::cout` schreiben zu dürfen
- **Ausgabe** mittels `cout` (oder `cerr`)

```
int alter= 7;
cout << "Ich bin " << alter << " Jahre alt." << endl;
cerr << "Bald werde ich " << alter+1 << ".\n";
```

- besondere Steuerzeichen: z.B. `\n` neue Zeile, `\t` Tabulator
- **Eingabe** mittels `cin`

```
cout << "Bitte Gewicht in kg eingeben: ";
double gewicht;
cin >> gewicht;
```

- Eselsbrücke: `>>`, `<<` zeigen in Richtung des Informationsflusses

```
cout << "Hallo Leute!\n"; // auf den Bildschirm
int k;
cin >> k; // von der Tastatur
```

## Operatoren in C++

Arithmetik	+ - * / %	3 + 5 * 2
Zuweisung	=	j = 3
Vergleich	== != > >= < <=	j == 5
Logik	&&    !	(x >= 0.0) && (x <= 1.0)
Ein-/Ausgabe	<< >>	cout << "Hello World!";
...		

- weitere Operatoren in der Kurzanleitung
- Priorität der Operatoren, z.B. \* vor +. Im Zweifel Klammern setzen!
- implizite Typumwandlung (*Cast*):  $1/4.0$  liefert  $0.25$   
(Umwandlung `int`  $\rightarrow$  `double`)
- **Vorsicht** bei Integer-Division:  $1/4$  liefert  $0$  !

# Beispiele zu Operatoren

```
double x= 3, y= 10/x;    // y = 10.0/3.0 = 3.333...

int a= 3, b= 10/a,      // b = 3      (ganzzahlige Div.)
    rest= 10%a;        // rest = 1   (Modulo-Operator)

string s1= "Blumento", s2= "pferde",
        s3= s1 + s2;    // "Blumentopferde"

bool istNeg= b < 0,     // false
     istPos= !istNeg && (b != 0); // Negierung und ungleich

a= b= 77;              // Zuweisungsoperator gibt etwas zurueck

bool isTen= (a == 10),  // false
     wrong= (a = 10);   // 10 -> true  (nur 0 -> false)
```

## Merke:

- Vergleich `==` und Zuweisung `=` nicht verwechseln!
- Vorsicht bei Division von `int`.

# Kontrollstrukturen: bedingte Verzweigung

- **if-else-Verzweigung:**

```
if (i < 0)
{ // Bedingung erfuehlt
    cout << "i_ist_negativ" << endl;
}
else
{ // Bedingung nicht erfuehlt
    cout << "i_ist_nicht_negativ" << endl;
}
```

- kein ; nach `if` oder `else` !
- `else`-Teil kann auch entfallen
- nur eine Anweisung im Block: `{ }` dürfen wegfallen

```
if (i < 0) // Bedingung erfuehlt
    cout << "i_ist_negativ" << endl;
else
{ // Bedingung nicht erfuehlt
    cout << "i_ist_nicht_negativ" << endl;
}
```

Dreimal derselbe Code:

- 1 gut lesbar durch Einrückung, Kommentare und genug Leerzeichen:

```
if ( a*a - 2*a == 0 ) // nur erfuehlt, falls a 0 oder 2
{
    cout << "Null_oder_Zwei:_ " << a << endl;
}
else
    cout << "weder_Null_noch_Zwei:_ " << a << endl;

cout << "Bye!\n";      // Abschied
```

- 2 falsche Einrückung kann auch verwirren:

```
if ( a*a - 2*a == 0 ) { // kompakter, aber ok
    cout << "Null_oder_Zwei:_ " << a << endl;
}
else
    cout << "weder_Null_noch_Zwei:_ " << a << endl;
    cout << "Bye!\n";      // missverstaendlich!!!
```

# Lesbarkeit von Code (2)

Dreimal derselbe Code:

- ③ Der Compiler versteht auch das, du auch?

```
if (a*a-2*a==0) {cout << "Null oder Zwei: " << a << endl;} else  
cout << "weder Null noch Zwei: " << a << endl; cout << "Bye!\n";
```

**Lesbarkeit extrem wichtig**, insbesondere für Fehlersuche!

- Code **einrücken** (Empfehlung: 4 Leerzeichen)
- öffnende und schließende **Blockklammern** untereinander
- genug **Leerzeichen** als Trenner zwischen Operatoren, z.B. `a = 17 + 2*3;` statt `a=17+2*3;`
- **Kommentare** (lieber zuviel als zuwenig), um eigenen Code auch noch in 2 Wochen zu verstehen

## Bedingte Verzweigung (2)

- Fallunterscheidung mit `if-else`-Verzweigung:

```
if (i==1)
    cout << "Eins\n";
else if (i==2)
    cout << "Zwei\n";
else if (i==3)
    cout << "Drei\n";
else
    cout << "Diese_Zahl_kenne_ich_nicht!\n";
```

- Alternativ mit `switch-case`-Verzweigung (`break` nicht vergessen!):

```
switch (i)
{
    case 1:  cout << "Eins\n";  break;
    case 2:  cout << "Zwei\n";  break;
    case 3:  cout << "Drei\n";  break;
    default: cout << "Diese_Zahl_kenne_ich_nicht!\n";
}
```