2. Datentypen und Variablen I

Prof. Dr. Markus Gross

Informatik I für D-ITET (WS 03/04)

- Elemente eines C++ Programmes
- Präprozessor `#include`
- `main()`-Funktion
- Ein- und Ausgabe mit `cin` und `cout`-Objekten
- Deklaration und Verwendung von Variablen
- Einfache C++-Funktionen

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Vorbemerkungen

- C++ ist *case-sensitive*, d.h. Gross- und Kleinschreibung haben unterschiedliche Bedeutung
  - `cout` versus `COUT`
- *Kommentarzeilen* beginnen mit `//` und beinhalten Anmerkungen des Programmierers
- Sind von höchster Bedeutung für die Lesbarkeit von Code
- Unterschied zu C:
  - Kommentare in `/*..........*/`
- Wir verwenden ausschliesslich C++ Syntax
- C++ Quelldateien (source files) enden üblicherweise mit `*.cpp`

Beispiel_1: myfirst.cpp

```
// myfirst.cpp--displays a message

#include <iostream>              // a PREPROCESSOR directive
using namespace std;             // make standard defs visible
int main()                       // heading summarizes
                                 // function
{                                // start of function body
   cout << "Come up and C++ me some time."; // message
   cout << "\n";                 // start a new line
   return 0;                     // terminate main()
}                                 // end of function body
```

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Elemente von myfirst.cpp

- Präprozessor-Anweisung **`#include`** zur Einbindung weiterer *Header-Files*
  - Header files enthalten zusätzliche Deklarationen
  - Enden oft mit **`*.h`**
  - Nicht-standard, veraltet: **`#include <iostream.h>`**
- **`using namespace`** macht bestimmte Definitionen für Progamm sichtbar (später)
- **`cout`** ist C++ Objekt zur Ausgabe
  - entspricht **`printf()`** bei C-Programmen
- Funktionskopf **`int main()`**
- Funktionskörper begrenzt durch Klammern **`{}`**
- Klammern **`{}`** bilden einen sogenannten *Block*
- **`return`** Anweisung zum Beenden der Funktion

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Die `main()`-Funktion

- **`main()`** hat die generelle Form

```
int main()
{
   statements;
   return 0;
}
```

- Obiges Beispiel ist eine *Funktions-Definition*
  - Besteht aus *Funktionskopf* (*function heading*) und *Funktionskörper* (*function body*)
  - Funktionskörper besteht aus einem Block von *Anweisungen* (*statements*)
- Eine C++-Anweisung ist eine abgeschlossene Instruktion
  - Muss mit einem *Semikolon* (`;`) enden
  - **`return`** Anweisung beendet die Funktion

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Funktionsaufruf

- In C++ werden Funktionen *aufgerufen* (*function call*)
- Der Funktionskopf bildet das *Interface* zur aufrufenden Funktion
- Teil vor dem Funktionsnamen gibt den *Rückgabetyp* (*return type*) an
- Heading von **`main()`** beschreibt Interface mit dem Betriebssystem
- **`return 0`** gibt Wert 0 an die aufrufende Funktion zurück - *Rückgabewert*

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Funktionsaufruf

- `main()` heisst, dass die Funktion keine Werte von der aufrufenden Funktion übernimmt
- `main()` hat also keine *Argumente*
- Argumente dienen also zur Uebergabe von Information an die aufgerufene Funktion
- `int main(void)` heisst, dass **`main`** einen Rückgabetyp **`int`** hat und keine Argumente besitzt
  - `void` sagt explizit aus, dass keine Argumente verwendet werden
  - dient der Lesbarkeit - Stilfrage
- `return 0` wird für `main()` angenommen, falls nicht explizit programmiert

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Besonderheit von `main()`

- Main-Funktion muss in jedem C++ Programm vorliegen
- Programmausführung beginnt immer in der Main-Funktion
- Ausnahme z.B. sind dll-Libraries unter Windows

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Zeichenausgabe

- `cout << "This is Info I";` zur Ausgabe der Zeichen in Anführungszeichen
- Zeichensatz in `" "` ist ein *String*
  - Ein C++ String ist eine wohldefinierte Folge von Zeichen
  - Mehr später
- `cout` ist ein vordefiniertes *Objekt*
  - Es kann viele verschiedene Datentypen darstellen
  - Hat ein einfaches Interface
- `cout` gibt also einen *Zeichenstrom* (*stream*) aus
- `<<` Operator übergibt Daten an `cout`
  - Kann auch Bedeutung des bitweisen Links-Shift haben
- `<<` ist also *überladen* (*overloaded*) (mehrfache Bedeutung)

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

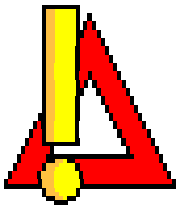
\n Zeichen und Formate

- `\n` ist ein Steuerzeichen zum Beginn einer neuen Zeile (*newline character*)
  - Kann beliebig innerhalb einer Ausgabe verwendet werden
  - `<< endl;` als Alternative
- Semikolon am Ende jeder Anweisung erfoderlich – entspricht nicht dem Zeilenende!
- Formate sind generell frei und vom individuellen Programmierstil abhängig
- Sogenannte *white space characters* (space, tab, cr) zwischen Tokens
  - Tokens sind untrennbare Elemente einer Anweisung (z.B. Funktionsname
  - `int ma in()` im Gegensatz zu `int main ( )`

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Guter Stil ist...

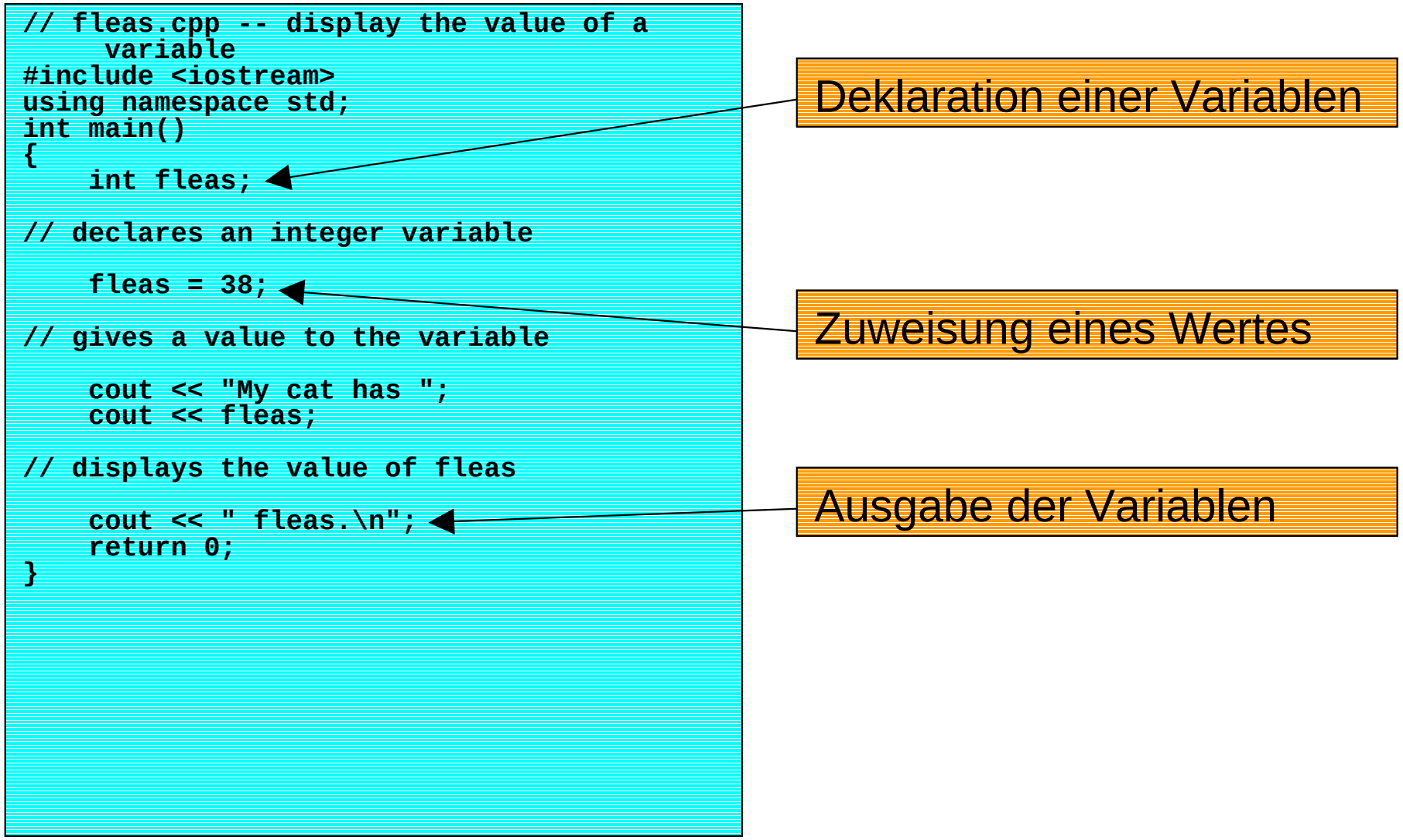
- Nur eine Anweisung pro Zeile
- Klammern als Blockgrenzen immer in eigene Zeilen
- Anweisungen im Block immer eingerückt durch Tab
- Kein space zwischen Funktionsname und Argumentenliste
- Mehr Stilhinweise folgen...



ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Beispiel_2: Variablen

```
// fleas.cpp -- display the value of a
      variable
#include <iostream>
using namespace std;
int main()
{
    int fleas;

// declares an integer variable

    fleas = 38;

// gives a value to the variable

    cout << "My cat has ";
    cout << fleas;

// displays the value of fleas

    cout << " fleas.\n";
    return 0;
}
```



ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Deklaration und Definition

- `int fleas;` ist eine *Deklaration* (s-Anweisung)
  - Deklarationen machen einen Namen für ein C++ Programm kenntlich
- `int fleas;` ist auch eine *Definition*
  - Definitionen kreieren auch eine Entität, die für den Namen steht
  - Beispiel: Speicherplatz dazu reservieren
- Bei einfachen C++ Variablen ist Deklaration = Definition
- Bei komplexeren Datentypen und Funktionen kann Deklaration und Definition getrennt erfolgen

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Deklaration und Definition

- `int fleas;` deklariert den Namen fleas vom Typ Integer und ordnet entsprechend Speicher zu
- Speichergrösse hängt von der jeweiligen Implementation ab
- Einfache Variablendefinition besteht aus 3 Teilen:
- `<type> <name>;`
- C++ erlaubt die Variablendefinition (mit wenigen Restriktionen) an beliebigen Stellen im Programm
  - Guter Stil: So nah wie möglich an der Stelle, wo die Variable zum ersten Mal verwendet wird
- Gültigkeitsbereich (scope) muss beachtet werden

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Zuweisungen

- `=` ist der *Zuweisungsoperator* (*assignment*)
- Kann in Serie angewandt werden

```
int a;
int b;
int c;
a = b = c = 1;
```

- Zuweisungsoperatoren können beliebig überladen werden (später)

Beispiel: Eingabe

```
// yourcat.cpp -- input and output

#include <iostream>
using namespace std;

int main()
{
    int fleas;

    cout << "How many fleas does your
     cat have?\n";
    cin >> fleas;

// C++ input
// next line concatenates output

    cout << "Well, that's " << fleas <<
     " fleas too many!\n";

    return 0;
}
```

- **cin** – Objekt zur Dateneingabe an das Programm
- **cin** liest Zeichenstrom von der Eingabe und weist Werte der Variablen zu
- Typenumwandlung erfolgt implizit
- **<<** – kann Ausgabe konkatenieren
- **<<** ist also überladen (overloaded) (mehrfache Bedeutung)
- **cin** ist ein Objekt der Klasse **istream**
- **cout** ist ein Objekt der Klasse **ostream**
- Beide sind in **iostream** definiert

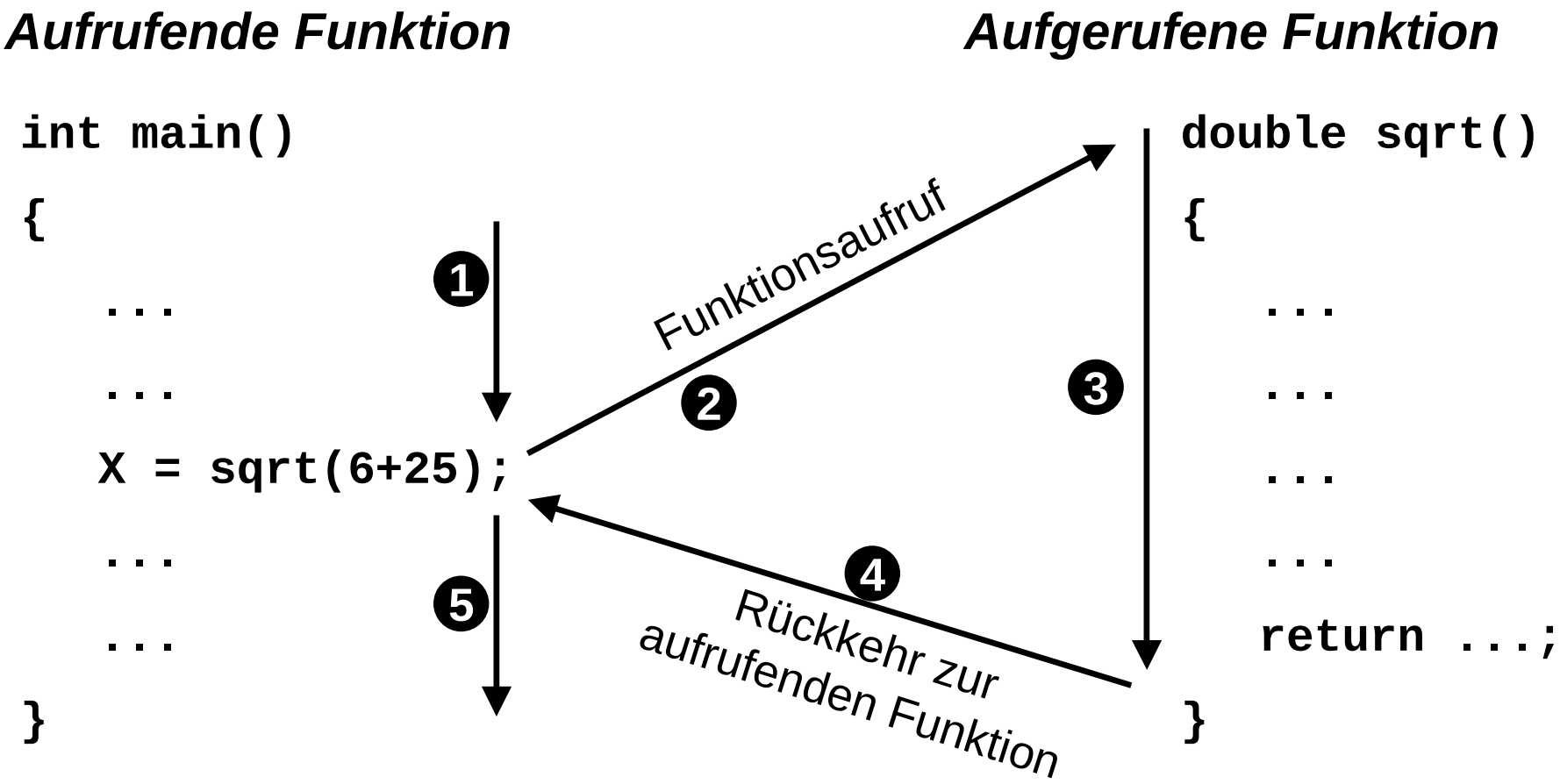
ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Funktionen

- Zwei Arten von Funktionen in C++
  - Funktionen mit Rückgabewert
  - Funktionen ohne Rückgabewert
- Bei einem Funktionsaufruf springt das Programm zur Adresse der aufgerufenen Funktion und führt die dortigen Instruktionen aus
- Programmfluss wird durch einen Sprung unterbrochen
- Nach Beenden der Funktion wird zurückgesprungen
- `X = sqrt(6.35);` beschreibt einen *Funktionsaufruf* (*function call*)
- Uebergabewerte werden auch *Argumente* oder *formale Parameter* der Funktion genannt

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

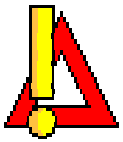
Funktionsaufruf



ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Funktionsprototypen

- Zur korrekten Typenprüfung muss der Compiler folgende Information haben
  - Anzahl und Typen der Argumente
  - Rückgabetyp
  - Beispiel: `double sqrt(double)`
- Zu diesem Zweck dienen *Funktionsprototypen*
- Für jede Funktion in einem C++ Programm sollte auch ein Prototyp vorhanden sein
- Funktionsprototypen können
  - Explizit im Sourcecode eingefügt werden
  - Aus vorhandenen Headerfiles entnommen werden
- `double sqrt(double)` finden wir in `<cmath>` oder `<math.h>`



Beispiel_4: Funktionsprototypen

```
// sqrt.cpp -- use a square root function

#include <iostream>
using namespace std;

#include <cmath>      // or <math.h>

int main()
{
    double cover;

// use double for real numbers

    cout << "How many square feet of
     sheets do you have?\n";
    cin >> cover;
    double side;

// create another variable

    side = sqrt(cover);

// call function, assign return value

    cout << "You can cover a square with
     sides of " << side;
    cout << " feet\nwith your sheets.\n";
    return 0;
}
```

Präprozessor: include header file

Funktionsaufruf

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Funktionsprototypen

- Prototypen sind *Funktionsdeklarationen*
- *Funktionsdefinition* erfolgt an anderer Stelle
  - Typischerweise in einem anderen File
  - Vorcompilierte Bibliotheken für mathematische Funktionen
  - Werden bei Compilation explizit eingebunden:
  - `gcc –o sqrt sqrt.cpp -lm`
- Funktionen können zur *Initialisierung* von Variablen verwendet werden
- Beispiel: `double side = sqrt(power);`
- Weglassen der Argumente bedeutet void
- Beispiel: `int foo();` entspricht `int foo(void);`

Beispiel_5: Eigene Funktionen

```
// ourfunc.cpp -- defining your own
      function
#include <iostream>
using namespace std;

void simon(int);

// function prototype for simon()

int main()
{
    simon(3);

// call the simon() function

    cout << "Pick an integer: ";
    int count;
    cin >> count;
    simon(count);   // call it again
    return 0;
}

void simon(int n)

  // define the simon() function

{
    cout << "Simon says touch your toes
      " << n << " times.\n";
}
// void functions don't need return
     statement
```

- Explizite Deklaration der Funktion **`simon()`** durch Prototypen ausserhalb von **`main()`**
- Definition der Funktion an getrennter Stelle ebenfalls ausserhalb von **`main()`**
- Aufruf der Funktion innerhalb von **`main()`**
- **`main()`** kennt die Funktion **`simon()`** aufgrund der Deklaration
- Gültigkeitsbereich der Funktion ist global

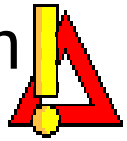
ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Funktionsdefinitionen

- Allgemeine Form einer Funktionsdefinition

```
Type Funktionsname(Argumentenliste)
{
   Statements;
}
```

- Funktionsdefinitionen erfolgen hintereinander in einer Quelldatei
- Bis auf `main()` können alle Funktionen im Programm aufgerufen werden



- `main()` wird vom Betriebssystem aufgerufen
- Return-Wert von `main()` wird auch Exit-Wert genannt
  - `0` bedeutet, dass das Programm korrekt beendet wurde