Programmierkurs
Die ersten Schritte

Manfred Hauswirth | Open Distributed Systems | Einführung ins Programmieren, WS 22/23

Rückblick

VL 0 „Organisation und Inhalt“: Ablauf der Vorlesung, Termine
VL 1 „Hello World“: „Lebenswichtiges“, Programablauf, Programmierablauf, Kompilierung und Ausführung von Programmen
VL 2 „Die ersten Schritte“: Erstes C-Programm, Elementare C-Strukturen, Datentypen, Operatoren, Schleifen
VL 3 „Kontrollstrukturen & Funktionen“: Syntax, Semantik, bedingte Anweisungen, Blöcke, Sichtbarkeit
VL 4 „Rekursive Funktionen & Bibliotheken“: rekursive Funktionsaufrufe, Modularisierung
VL 5 „Typen“: Einfache und strukturierte Datentypen, Wertebereiche, Typendefinition
VL 6 „Speicher und Adressen“: Speicher, Pointer, Funktionsaufrufe „call by value“ vs. „call by reference“
VL 7 „Speicher und Arrays“: Speicher, Arrays, mehrdimensionale Arrays, Arrays und Pointer
VL 8 „Dynamische Speicherverwaltung“: Speicherallokation, Fehlerbehandlung, Rückgabewerte, Arrays/Pointer/Adressen
VL 9 „Strings, Kanäle, Git“: Strings und Arrays, Zeichensätze, Stringlänge, Ein- und Ausgabe, Arbeiten mit git

Algorithmus vs. Programm

- Algorithmen beschreiben was ein Computer ausführen soll (in schematischer Form)
  - „Kochrezept“
  - Konzept

- Programmiersprachen stellen eine Schnittstelle dar, um die Algorithmen auf dem Computer definieren und ausführen zu können
  - Die genauen Schritte eines Algorithmus werden beschrieben

Algorithmus vs. Programm

- Algorithmen fokussieren auf Korrektheit, Vollständigkeit und Komplexität
  - Funktioniert der Algorithmus immer richtig?
  - Deckt der Algorithmus alle möglichen Fälle ab?
  - Ist der Algorithmus effizient (schnell, Ressourcenverbrauch)?

- Programmiersprachen müssen zusätzlich alle Details des Computers berücksichtigen
  - Die exakten Anweisungen für jeden Schritt des Algorithmus

Beispiel: Kuchen backen

- Input: Zutaten

Beispiel: Kuchen backen

- Input: Zutaten
- Softwareumgebung/Datenstrukturen: Werkzeuge

Beispiel: Kuchen backen

- Input: Zutaten
- Softwareumgebung/Datenstrukturen: Werkzeuge
- Algorithmus: Rezept

Beispiel: Kuchen backen

- Input: Zutaten
- Softwareumgebung/Datenstrukturen: Werkzeuge
- Algorithmus: Rezept
- Programm: Rezept ausführen

Beispiel: Kuchen backen

- Input: Zutaten
- Softwareumgebung/Datenstrukturen: Werkzeuge
- Algorithmus: Rezept
- Programm: Rezept ausführen
- Hardware: Kochwerkzeuge, Ofen

Beispiel: Kuchen backen

- Input: Zutaten
- Softwareumgebung/Datenstrukturen: Werkzeuge
- Algorithmus: Rezept
- Programm: Rezept ausführen
- Hardware: Kochwerkzeuge, Ofen
- Output: Kuchen ☺

Beispielalgorithmus: Zweierpotenzen

- Berechne die Zweierpotenzen bis n (also solange “$2^m < n$” gilt):
  - Sei m gleich 0
  - Sei p gleich 1
  - Solange p kleiner n ist, mache:
    - Gib „2 hoch m ist p“ auf der Konsole aus
    - Addiere zu m den Wert 1
    - Multipliziere p mit dem Wert 2

- Der Algorithmus ist natürlich-sprachlich in einer Art Pseudocode beschrieben!
- Warum geben wir 2^m zuerst aus, bevor m erhöht wird?

Beispielalgorithmus: Zweierpotenzen in Pseudocode

- Berechne die Zweierpotenzen bis n:

```
m = 0;
p = 1;
while (p < n)
        Ausgabe von: „2^m ist p“;
        m = m + 1;
        p = p * 2;
```

- Jetzt ist der Algorithmus in Pseudocode beschrieben!
- Kürzer und präziser

Algorithmus

- Ein Algorithmus ist eine Liste von Anweisungen, quasi die Essenz eines Programms, und wird in Pseudocode aufgeschrieben

- Wichtige Aspekte:
  - Korrektheit: Erfüllt der Algorithmus seine Anforderungen?
    D.h.: Gibt der obige Algorithmus wirklich die Zweierpotenzen aus?
  - Effizienz: Wie viel Zeit und wie viel Speicherplatz braucht er?
  - (Terminierung: Hält der Algorithmus immer an?)

Elemente von Pseudocode am Beispiel: Zweierpotenzen

- Algorithmus: Gebe Zweierpotenzen bis n aus:

```
m = 0;
p = 1;
while (p < n)
    Ausgabe von: „2^m ist p“;
    p = p * 2;
    m = m + 1;
```

- Der Algorithmus ist in Pseudocode beschrieben!

Elemente von Pseudocode am Beispiel: Zweierpotenzen

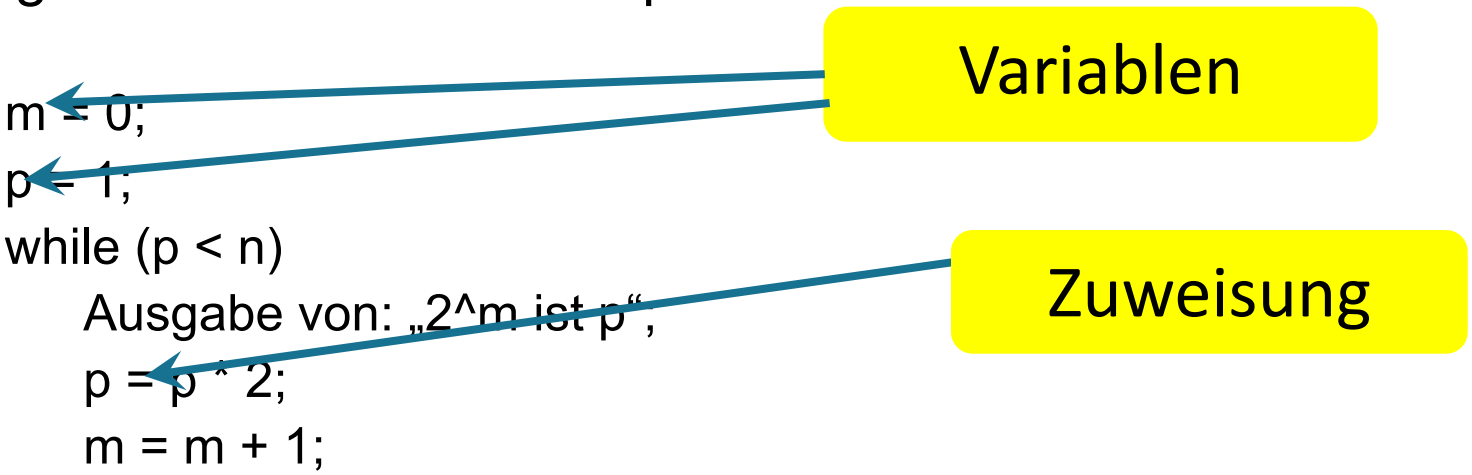
- Algorithmus: Gebe Zweierpotenzen bis n aus:


```
m = 0;
p = 1;
while (p < n)
    Ausgabe von: „2^m ist p“;
    p = p * 2;
    m = m + 1;
```

- Der Algorithmus ist in Pseudocode beschrieben!

Elemente von Pseudocode am Beispiel: Zweierpotenzen

- Algorithmus: Gebe Zweierpotenzen bis n aus:



- Der Algorithmus ist in Pseudocode beschrieben!

Elemente von Pseudocode am Beispiel: Zweierpotenzen

- Algorithmus: Gebe Zweierpotenzen bis n aus:

```
m = 0;
p = 1;
while (p < n)
    Ausgabe von: „2^m ist p“;
    p = p * 2;
    m = m + 1;
```

Variablen

Zuweisung

Berechnung

- Der Algorithmus ist in Pseudocode beschrieben!

Elemente von Pseudocode am Beispiel: Zweierpotenzen

- Algorithmus: Gebe Zweierpotenzen bis n aus:

```
m = 0;
p = 1;
while (p < n)
    Ausgabe von: „2^m ist p“;
    p = p * 2;
    m = m + 1;
```

Variablen

Zuweisung

Berechnung

Wiederholung

- Der Algorithmus ist in Pseudocode beschrieben!

Elemente von Pseudocode am Beispiel: Zweierpotenzen

- Algorithmus: Gebe Zweierpotenzen bis n aus:

```
m = 0;
p = 1;
while (p < n)
    Ausgabe von: „2^m ist p“;
    p = p * 2;
    m = m + 1;
```

Variablen

Zuweisung

Berechnung

- Der Algorithmus ist in Pseudocode beschrieben!

Wiederholung zusätzlich Verzweigung

Programmiersprache C

Wiederholung: Minimales C Programm

```
$ more hello.c
#include <stdio.h>

int main() {
    printf("Hello World.\n");
}
```

Wiederholung: Minimales C Programm

```
$ more hello.c
#include <stdio.h>

int main() {
    printf("Hello World.\n");
}

$ clang -Wall -std=c11 -o hello hello.c
```

Wiederholung: Minimales C Programm

```
$ more hello.c
#include <stdio.h>

int main() {
    printf("Hello World.\n");
}

$ clang -Wall -std=c11 -o hello hello.c

$ ./hello
```

Wiederholung: Minimales C Programm

```
$ more hello.c
#include <stdio.h>

int main() {
    printf("Hello World.\n");
}

$ clang -Wall -std=c11 -o hello hello.c

$ ./hello
Hello World.
```

Erstes „echtes“ C Programm: Zweierpotenzen

Beispielalgorithmus: Zweierpotenzen

Pseudocode

```
m = 0;
p = 1;
while (p < n)
  Ausgabe: "2^m ist p";
  m = m + 1;
  p = p * 2;
```

Beispielalgorithmus: Zweierpotenzen

Pseudocode

```
m = 0;
p = 1;
while (p < n)
  Ausgabe: "2^m ist p";
  m = m + 1;
  p = p * 2;
```

C-Code

Beispielalgorithmus: Zweierpotenzen

Pseudocode

```
m = 0;
p = 1;
while (p < n)
  Ausgabe: "2^m ist p";
  m = m + 1;
  p = p * 2;
```

C-Code

```
#include <stdio.h>
```

Beispielalgorithmus: Zweierpotenzen

Pseudocode

```
m = 0;
p = 1;
while (p < n)
  Ausgabe: "2^m ist p";
  m = m + 1;
  p = p * 2;
```

C-Code

```
#include <stdio.h>

int main() {

}
```

Beispielalgorithmus: Zweierpotenzen

Pseudocode

```
m = 0;
p = 1;
while (p < n)
  Ausgabe: "2^m ist p";
  m = m + 1;
  p = p * 2;
```

C-Code

```
#include <stdio.h>

int main() {
int m = 0;
int p = 1;

}
```

Beispielalgorithmus: Zweierpotenzen

Pseudocode

```
m = 0;
p = 1;
while (p < n)
  Ausgabe: "2^m ist p";
  m = m + 1;
  p = p * 2;
```

C-Code

```
#include <stdio.h>

int main() {
int m = 0;
int p = 1;
int n = 10; // nur als Beispiel

}
```

Beispielalgorithmus: Zweierpotenzen

Pseudocode

```
m = 0;
p = 1;
while (p < n)
  Ausgabe: "2^m ist p";
  m = m + 1;
  p = p * 2;
```

C-Code

```
#include <stdio.h>

int main() {
int m = 0;
int p = 1;
int n = 10; // nur als Beispiel
while (p < n) {

}
}
```

Beispielalgorithmus: Zweierpotenzen

Pseudocode

```
m = 0;
p = 1;
while (p < n)
  Ausgabe: "2^m ist p";
  m = m + 1;
  p = p * 2;
```

C-Code

```
#include <stdio.h>

int main() {
  int m = 0;
  int p = 1;
  int n = 10; // nur als Beispiel
  while (p < n) {
    printf("2^%d ist %d", m, p);
    m = m + 1;
    p = p * 2;
  }
}
```

Elementare C Strukturen

Elementare C Strukturen

- Variablen
- Zuweisung
- Berechnung

- Wiederholung
- Verzweigung

Das einfachste C-Programm Basisstruktur

- Das einfachstes C Programm besteht nur aus einer Funktion: `main` Das ist der Einsprungspunkt (Startpunkt) für das Betriebssystem.
- Das untenstehende Programm ist ein korrektes Programm, das jedoch „leer“ ist, d.h., **es tut nichts**.

```
int main() {

}
```

Typen und Variablen

- Variablen
  - Die Basiselemente eines Programms
  - Erlauben es, Daten strukturiert zu speichern
- Typen
  - Definieren die Art der Daten
  - Beispiele: Zahlen, Zeichen, …

- Beispiele:
  - `int x;` `/* Variable x vom Typ Integer */`
  - `int y, z;` `/* Variablen y und z vom Typ Integer */`

Bezeichner, z.B. für Variablennamen

- Namen sind frei wählbar mit folgenden Einschränkungen:
  - Erstes Zeichen aus: a-z, A-Z, _
  - Weitere Zeichen: 0-9, a-z, A-Z, _
  - Keine Schlüsselwörter
- Groß-/Kleinschreibung wird unterschieden.

- Schlüsselwörter sind C-Sprachelemente

  Beispiele:

```
– int, char, float, void, …      /* Typen */
– =, +, >, <, …                  /* Math. Operationen */
– while, for, if, else, …        /* Kontrollstrukturen */
– /*, */, //                     /* Kommentare */
```

Ausgaben: printf

- Zweck: Texte und Werte am Bildschirm ausgeben
- Beispiele:

```
printf("Hello world\n");
```

Ausgabe: `Hello world`

```
printf("Wert der Variablen i: %d\n", i);
```

Ausgabe (Beispiel): `Wert der Variablen i: 42`

```
printf("a(%d)+ b(%d) ist: %d\n", a, b, a + b);
```

Ausgabe (Beispiel): `a(102) + b (-60) ist: 42`

- Gedankenstütze: `printf("bla %d bla %d bla\n", x, 42)` gibt den Text in `"…"` am Bildschirm aus und ersetzt die `%d` durch die Ausdrücke hinter dem Komma

Zuweisung

- Weist einer Variablen einen Wert zu:
  - Operator: `=`

- Beispiele:

```
int x, y;          // Variablen x und y sind vom Typ Integer
x = 10;            // Zuweisung des Wertes 10 an x
y = - x;           // Zuweisung des negierten Wertes von x an y
x = y;             // Zuweisung des Wertes von y an x
```

Zuweisung

- Die Zuweisung besteht
  1. Aus der **Auswertung** der rechten Seite
  2. Aus der **Speicherung** des Ergebnisses der Auswertung in der Variablen der linken Seite

- Beispiel

```
1. int x;
2. x = 5;
3. int y;
```

Zustand	x	?		
Zustand	x	5		
Zustand	x	5	y	?

Mathematische Operationen

- Standardsatz an Operationen:
  - Basisoperatoren: +, -, *, /, %, …
  - Erlauben das Rechnen mit den Daten mit Hilfe von Variablen

- Beispiele:
  - `int x, y;` `// Variablen x und y vom Typ Integer`
  - `x + y` `// Addition`
  - `x - y` `// Subtraktion`
  - `x * y` `// Multiplikation`
  - `x % y` `// Modulo (Rest nach Division)`

Zuweisung: Merke

- Das Zeichen „=“ ist **kein Gleichheitszeichen**, sondern der Zuweisungsoperator
- Es ist also kein Gleichheitszeichen im Sinne einer Aussage „x hat den gleichen Wert wie y“, sondern hat die Bedeutung „x nimmt den Wert von y an“

  - `x = 5;` bedeutet: „x wird der Wert 5 zugewiesen“
  - `x = x + 1;` bedeutet: „x wird um 1 erhöht“

- Andere Programmiersprachen verwenden zum Teil andere Zeichen.

Weitere mathematische Operationen

- Logische Operationen:
  - Rechnen mit Wahrheitswerten („true“, „false“)
  - Logisches „und“ (∧): **&&**, logisches „oder“ (∨): **||**, … (kommt später genauer)
- Vergleiche:
  - Vergleichen zweier Werte
  - Kleiner: **<**, größer: **>**, kleiner gleich: **<=**, größer gleich: **>=**, …
  - **Gleichheit: ==**
- Beispiele:

```
int x, y;        /* Variablen x und y vom Typ Integer */
x > y            /* Vergleich: Größer als */
x == y           /* Test auf Gleichheit */
```

Deklaration von Variablen

- Jede Variable **muss** vor ihrer ersten Verwendung **deklariert** werden.

```
int month;    // Deklaration
month = 10;   // Initialisierung mit einem Wert (Zuweisung)

int year = 2023;  // Deklaration und Initialiserung in
                  // 1 Schritt
```

Ausdrücke (expressions)

- Während der Programmausführung entstehen neue Werte, die in Variablen gespeichert werden können.

```
int year = 2000;     // declaration and initialization
year = year + 23;    // evaluation of expression; year is now 2023
```

Ausdrücke (expressions)

- Ausdrücke sind die elementaren funktionalen Einheiten eines Programms.
  - Neue Werte entstehen durch Auswertung von Ausdrücken.

Ausdrücke (expressions)

- Ausdrücke sind die elementaren funktionalen Einheiten eines Programms.
  - Neue Werte entstehen durch Auswertung von Ausdrücken.
- C-Ausdrücke werden nach einer bestimmten Syntax gebildet, welche weitgehend der Syntax mathematischer Ausdrücke entspricht.

Ausdrücke (expressions)

- Ausdrücke sind die elementaren funktionalen Einheiten eines Programms.
  - Neue Werte entstehen durch Auswertung von Ausdrücken.
- C-Ausdrücke werden nach einer bestimmten Syntax gebildet, welche weitgehend der Syntax mathematischer Ausdrücke entspricht.
- Ausdrücke werden durch Einsetzen der aktuellen Werte ausgewertet

```
int celsius = 0;
int fahrenheit = 92;
```

Ausdrücke (expressions)

- Ausdrücke sind die elementaren funktionalen Einheiten eines Programms.
  - Neue Werte entstehen durch Auswertung von Ausdrücken.
- C-Ausdrücke werden nach einer bestimmten Syntax gebildet, welche weitgehend der Syntax mathematischer Ausdrücke entspricht.
- Ausdrücke werden durch Einsetzen der aktuellen Werte ausgewertet

```
int celsius = 0;
int fahrenheit = 92;
```

Zustand	fahrenheit	92	celsius	0

Ausdrücke (expressions)

- Ausdrücke sind die elementaren funktionalen Einheiten eines Programms.
  - Neue Werte entstehen durch Auswertung von Ausdrücken.
- C-Ausdrücke werden nach einer bestimmten Syntax gebildet, welche weitgehend der Syntax mathematischer Ausdrücke entspricht.
- Ausdrücke werden durch Einsetzen der aktuellen Werte ausgewertet

```
int celsius = 0;
int fahrenheit = 92;
```

Zustand	fahrenheit	92	celsius	0

```
celsius = (fahrenheit - 32) * 5 / 9;
```

Ausdrücke (expressions)

- Ausdrücke sind die elementaren funktionalen Einheiten eines Programms.
  - Neue Werte entstehen durch Auswertung von Ausdrücken.
- C-Ausdrücke werden nach einer bestimmten Syntax gebildet, welche weitgehend der Syntax mathematischer Ausdrücke entspricht.
- Ausdrücke werden durch Einsetzen der aktuellen Werte ausgewertet

```
int celsius = 0;
int fahrenheit = 92;
```

Zustand	fahrenheit	92	celsius	0

```
celsius = (fahrenheit - 32) * 5 / 9;
```

Zustand	fahrenheit	92	celsius	33

Beispiel: Swap

- In Programmen tritt häufig der Fall auf, dass zwei Variable ihre Werte vertauschen sollen (swap):

```
int x = 5;
int y = 7;
```

Beispiel: Swap

- In Programmen tritt häufig der Fall auf, dass zwei Variable ihre Werte vertauschen sollen (swap):

```
int x = 5;
int y = 7;

// swap values of x and y
x = y;
y = x;
```

Beispiel: Swap

- In Programmen tritt häufig der Fall auf, dass zwei Variable ihre Werte vertauschen sollen (swap):

```
int x = 5;
int y = 7;

// swap values of x and y
x = y;
y = x;
```

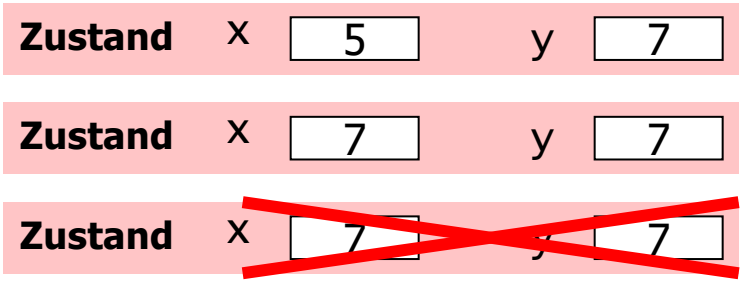
Zustand	x	5	y	7
Zustand	x	7	y	7
Zustand	x	7	y	7

Beispiel: Swap

- In Programmen tritt häufig der Fall auf, dass zwei Variable ihre Werte vertauschen sollen (swap):

```
int x = 5;
int y = 7;

// swap values of x and y
x = y;
y = x;
```



Falsch!

Beispiel: Swap

- Man braucht eine Hilfsvariable zum Zwischenspeichern

```
int x = 5;
int y = 7;
```

Beispiel: Swap

- Man braucht eine Hilfsvariable zum Zwischenspeichern

```
int x = 5;
int y = 7;
// swap values of x and y
int z;
```

Beispiel: Swap

- Man braucht eine Hilfsvariable zum Zwischenspeichern

```
int x = 5;
int y = 7;
// swap values of x and y
int z;

z = x;
x = y;
y = z;
```

Beispiel: Swap

- Man braucht eine Hilfsvariable zum Zwischenspeichern

```
int x = 5;
int y = 7;
// swap values of x and y
int z;

z = x;
x = y;
y = z;
```

Zustand	x	5	y	7	z	?
Zustand	x	5	y	7	z	5
Zustand	x	7	y	7	z	5
Zustand	x	7	y	5	z	5

Kommentare im Quellcode

Bevor wir anfangen ...

- „Code wird von Menschen für Menschen geschrieben.“

Bevor wir anfangen ...

- „Code wird von Menschen für Menschen geschrieben.“

- **Lesbarkeit** für andere Programmierer*innen (und einen selbst!) ist **entscheidend für die Wartbarkeit** von Software.
  - Namensgebung
  - Kommentierung
  - Stil und Struktur (Übersichtlichkeit, Formatierung, ...)

Quellcode-Kommentare

- Kommentare haben keinerlei Einfluss auf den Programmablauf.
- Kommentare sind trotzdem sehr wichtig:
  - Für andere, die das Programm lesen und verstehen wollen.
  - Für den/die Programmierer*in (Autor*in) selbst, der/die nach wenigen Wochen nicht mehr weiß, was da genau geschieht.
- **Kommentiert wird sofort beim Programmieren, nicht nachträglich!**
  - Nur nicht immer in der Vorlesung, dafür in den Übungen.

```
/* Kommentar über mehrere Zeilen
*/
// Kommentar bis Zeilenende
```

Quellcode-Kommentare

- C hat zwar kein festes Kommentierschema
- Aber folgende Konventionen sind sinnvoll:
  - Kommentare zu jeder Funktion

```
// GOOD: Calculate distance of point (a,b) to origin
// BAD:  Do some distance calculations
        int dist_to_orgin( int a, int b ) {
          ...
        }
```

  - Zusammenfassung der Funktionalität in eigenen Worten
  - Beschreibung der Parameter
- Kommentare zu jedem größeren Codeblock

Kontrollstrukturen

Bedingte Anweisung

- Manche Anweisungen sollen nur unter bestimmten Bedingungen ausgeführt werden.

Bedingte Anweisung

- Manche Anweisungen sollen nur unter bestimmten Bedingungen ausgeführt werden.
  - Z.B. berechne den Absolutwert einer Variable:

```
if ( x < 0 ) {
  x = -x;
}
```

Bedingte Anweisung

- Manche Anweisungen sollen nur unter bestimmten Bedingungen ausgeführt werden.
  - Z.B. berechne den Absolutwert einer Variable:

```
if ( x < 0 ) {
  x = -x;
}
```

  - Syntaktische Form:

```
if ( <condition> ) <block>
```

Bedingte Anweisung

- Syntaktische Form:

```
if ( <condition> ) <block>
```

- Ablauf:
  1. Werte die Bedingung `<condition>` aus.
  2. Falls Ergebnis das „true“ („wahr“) ist, führe Anweisung(en) aus.

Bedingte Anweisung

- Syntaktische Form:

```
if ( <condition> ) <block>
```

- Ablauf:
  1. Werte die Bedingung `<condition>` aus.
  2. Falls Ergebnis das „true“ („wahr“) ist, führe Anweisung(en) aus.

- Bedingung: Logischer Ausdruck (boolean expression/condition), d.h. ein Ausdruck, dessen Auswertung „true“ oder „false“ ergibt.

Logische Ausdrücke (boolean expressions)

- Wie wird „true“ bzw. „false“ dargestellt bzw. gespeichert?
  - Wert == 0 ⇒ false / falsch
  - Wert != 0 ⇒ true / wahr

- Vergleichsoperatoren liefern die 0 (false) oder 1 (true):
  - **==** gleich
  - **!=** ungleich
  - **<** kleiner
  - **>** größer
  - **<=** kleiner gleich
  - **>=** größer gleich

Blöcke

- Ein Block ist eine Zusammenfassung einer Folge von Anweisungen.

Blöcke

- Ein Block ist eine Zusammenfassung einer Folge von Anweisungen.

```
{ // begin of block
   int z = x;
   x = y;
   y = z;
} // end of block
```

- Eine Zusammenfassung von Ausdrücken (ein Block) wird in C durch geschweifte Klammern `{...}` realisiert.

Schleifen und Wiederholungen

- Es gibt häufig Situationen, in denen ein Programmblock mehrmals mit jeweils sich ändernden Werten durchlaufen werden soll: **Schleifen** über den Programmblock.

Schleifen und Wiederholungen

- Es gibt häufig Situationen, in denen ein Programmblock mehrmals mit jeweils sich ändernden Werten durchlaufen werden soll: **Schleifen** über den Programmblock.

  - `while`-Schleife: Anzahl der Iterationen wird durch eine Bedingung bestimmt.

Schleifen und Wiederholungen

- Es gibt häufig Situationen, in denen ein Programmblock mehrmals mit jeweils sich ändernden Werten durchlaufen werden soll: **Schleifen** über den Programmblock.

  - `while`-Schleife: Anzahl der Iterationen wird durch eine Bedingung bestimmt.
  - `for`-Schleife: Anzahl der Iterationen ist bekannt.

while-Schleife

- Beispiel: Zählt von 0 bis 10.

```
int i = 0;
while ( i <= 10 ) {
    printf("i: %d\n", i);
    i = i + 1;
    // i++;  // Alternative Schreibweise
}
```

`for`-Schleife

- Beispiel: Zählt von 0 bis 10.

```
int i;
for ( i = 0; i <= 10; i = i + 1 ) {
  printf("i: %d\n", i);
}
```

for-Schleife

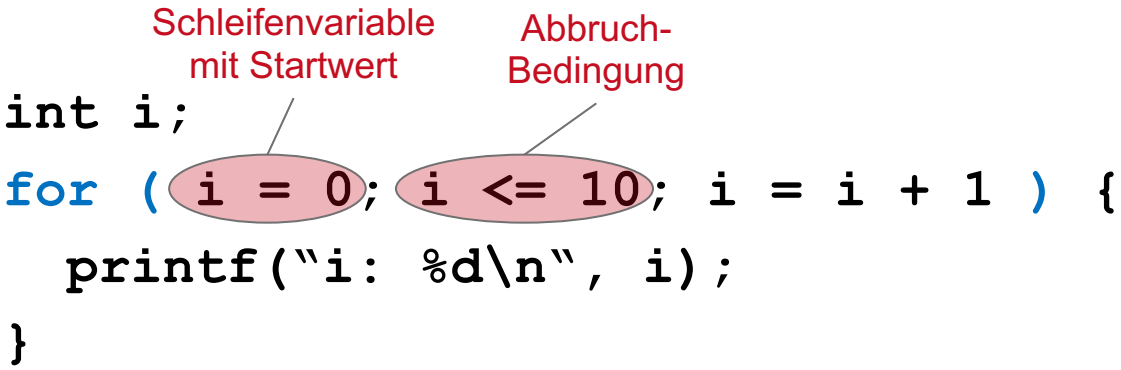
- Beispiel: Zählt von 0 bis 10.

Schleifenvariable mit Startwert

```
int i;
for ( i = 0; i <= 10; i = i + 1 ) {
  printf("i: %d\n", i);
}
```

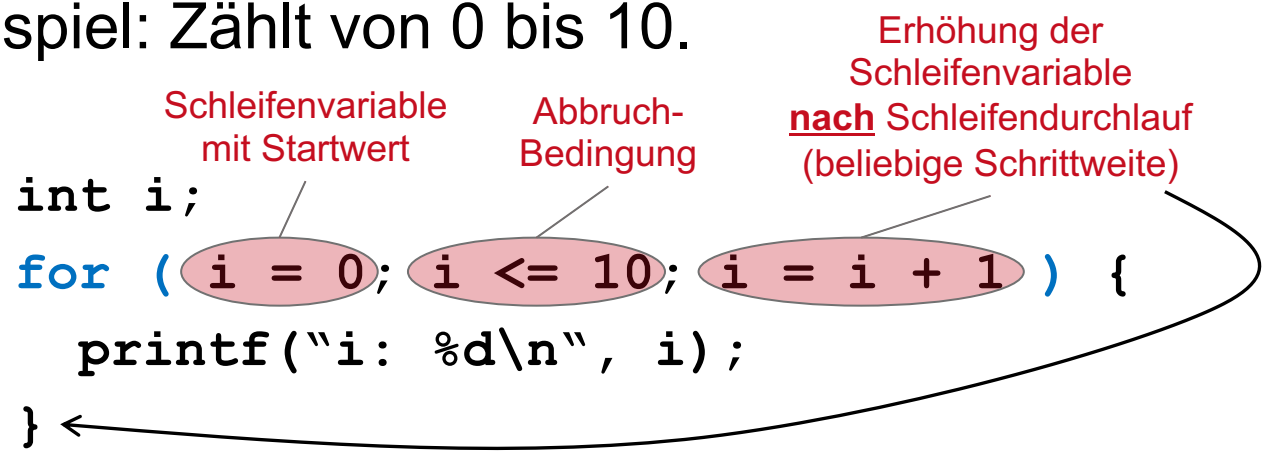
`for`-Schleife

- Beispiel: Zählt von 0 bis 10.



`for`-Schleife

- Beispiel: Zählt von 0 bis 10.



Zusammenfassung Beispielalgorithmus: Zweierpotenzen

Pseudocode

```
m = 0;
p = 1;
while (p < n)
  Ausgabe: "2^m ist p";
  m = m + 1;
  p = p * 2;
```

Zusammenfassung Beispielalgorithmus: Zweierpotenzen

Pseudocode

```
m = 0;
p = 1;
while (p < n)
  Ausgabe: "2^m ist p";
  m = m + 1;
  p = p * 2;
```

C-Code

Zusammenfassung Beispielalgorithmus: Zweierpotenzen

Pseudocode

```
m = 0;
p = 1;
while (p < n)
  Ausgabe: "2^m ist p";
  m = m + 1;
  p = p * 2;
```

C-Code

```
#include <stdio.h>
```

Zusammenfassung Beispielalgorithmus: Zweierpotenzen

Pseudocode

```
m = 0;
p = 1;
while (p < n)
  Ausgabe: "2^m ist p";
  m = m + 1;
  p = p * 2;
```

C-Code

```
#include <stdio.h>

int main() {

}
```

Zusammenfassung Beispielalgorithmus: Zweierpotenzen

Pseudocode

```
m = 0;
p = 1;
while (p < n)
  Ausgabe: "2^m ist p";
  m = m + 1;
  p = p * 2;
```

C-Code

```
#include <stdio.h>

int main() {
  int m = 0;
  int p = 1;
}
```

Zusammenfassung Beispielalgorithmus: Zweierpotenzen

Pseudocode

```
m = 0;
p = 1;
while (p < n)
  Ausgabe: "2^m ist p";
  m = m + 1;
  p = p * 2;
```

C-Code

```
#include <stdio.h>

int main() {
  int m = 0;
  int p = 1;
  int n = 10; // nur als Beispiel
}
```

Zusammenfassung Beispielalgorithmus: Zweierpotenzen

Pseudocode

```
m = 0;
p = 1;
while (p < n)
  Ausgabe: "2^m ist p";
  m = m + 1;
  p = p * 2;
```

C-Code

```
#include <stdio.h>

int main() {
  int m = 0;
  int p = 1;
  int n = 10; // nur als Beispiel
  while (p < n) {



  }
}
```

Zusammenfassung Beispielalgorithmus: Zweierpotenzen

Technische Universität Berlin

Pseudocode

```
m = 0;
p = 1;
while (p < n)
  Ausgabe: "2^m ist p";
  m = m + 1;
  p = p * 2;
```

C-Code

```c
#include <stdio.h>

int main() {
  int m = 0;
  int p = 1;
  int n = 10; // nur als Beispiel
  while (p < n) {
    printf("2^%d ist %d", m, p);
    m = m + 1;
    p = p * 2;
  }
}
```

Ausblick

VL 0 „Organisation und Inhalt“: Ablauf der Vorlesung, Termine
VL 1 „Hello World“: „Lebenswichtiges“, Programablauf, Programmierablauf, Kompilierung und Ausführung von Programmen
VL 2 „Die ersten Schritte“: Erstes C-Programm, Elementare C-Strukturen, Datentypen, Operatoren, Schleifen
VL 3 „Kontrollstrukturen & Funktionen“: Syntax, Semantik, bedingte Anweisungen, Blöcke, Sichtbarkeit
VL 4 „Rekursive Funktionen & Bibliotheken“: rekursive Funktionsaufrufe, Modularisierung
VL 5 „Typen“: Einfache und strukturierte Datentypen, Wertebereiche, Typendefinition
VL 6 „Speicher und Adressen“: Speicher, Pointer, Funktionsaufrufe „call by value“ vs. „call by reference“
VL 7 „Speicher und Arrays“: Speicher, Arrays, mehrdimensionale Arrays, Arrays und Pointer
VL 8 „Dynamische Speicherverwaltung“: Speicherallokation, Fehlerbehandlung, Rückgabewerte, Arrays/Pointer/Adressen
VL 9 „Strings, Kanäle, Git“: Strings und Arrays, Zeichensätze, Stringlänge, Ein- und Ausgabe, Arbeiten mit git

Literaturempfehlung

- Mordern C, J. Gustedt
  - https://modernc.gforge.inria.fr/
- Beej's Guide to C Programming, Brian “Beej” Hall
  - http://beej.us/guide/bgc/

Slides für Interessierte

Typen und Variablen

`int`: Ganze Zahl („integer“)

- Erlaubt das Speichern eines Integer (ganzzahligen) Wertes in einer Variable
- Typischerweise 32 Bit
- Wertebereich: -2.147.483.648 (-2^{32}) bis 2.147.483.647 (2^{32}-1) oder auch INT_MIN bis INT_MAX

Typen und Variablen

`int`: Ganze Zahl („integer“)

- Erlaubt das Speichern eines Integer (ganzzahligen) Wertes in einer Variable
- Typischerweise 32 Bit
- Wertebereich: -2.147.483.648 (-2^{32}) bis 2.147.483.647 ($2^{32}-1$) oder auch INT_MIN bis INT_MAX

```
int x, y;
x = 2014;
```

Typen und Variablen

`int`: Ganze Zahl („integer“)

- Erlaubt das Speichern eines Integer (ganzzahligen) Wertes in einer Variable
- Typischerweise 32 Bit
- Wertebereich: -2.147.483.648 (-2^{32}) bis 2.147.483.647 (2^{32}-1) oder auch INT_MIN bis INT_MAX

```
#include <limits.h>

int x, y;
x = 2014;
y = INT_MAX;        // y = -2.147.483.648
```

Typen und Variablen

`int`: Ganze Zahl („integer“)

- Erlaubt das Speichern eines Integer (ganzzahligen) Wertes in einer Variable
- Typischerweise 32 Bit
- Wertebereich: -2.147.483.648 (-2^{32}) bis 2.147.483.647 (2^{32}-1) oder auch INT_MIN bis INT_MAX

```
#include <limits.h>

    int x, y;
    x = 2014;
y = INT_MAX;        // y = -2.147.483.648
  printf("x = %d and y = %d \n", x, y);
```

Ausgaben: printf

- Formatierte Ausgabe in C mittels: `printf(fmt, args)`
- printf() gibt die Parameter `args` unter Kontrolle des sogenannten Formatstrings `fmt` aus
- Der Formatstring `fmt` ist eine Zeichenkette mit Platzhaltern
- Beispiele:

```
printf("Hello world\n");
```
Ausgabe: `Hello world`

```
printf("Wert der Variablen i: %d\n", i);
```
Ausgabe (Beispiel): `Wert der Variablen i: 42`

```
printf("a(%d)+ b(%d) ist: %d\n", a, b, a + b);
```
Ausgabe (Beispiel): `a(102) + b (-60) ist: 42`

- Platzhalter-Beispiele (Formatstrings):

`%d`	`Integer`	`int`
`%c`	`Einzelzeichen`	`char`

Typen und Variablen

`char`: Zeichen („charcter“)

- Erlaubt das Speichern eines Zeichens
  (Buchstaben werden durch Zahlen repräsentiert)
- Typischerweise: 8 Bit
- „Wertebereich“: –128 bis 127 (anderer „Typ“, andere Bedeutung)

Typen und Variablen

`char`: Zeichen („charcter“)

- Erlaubt das Speichern eines Zeichens
  (Buchstaben werden durch Zahlen repräsentiert)
- Typischerweise: 8 Bit
- „Wertebereich“: –128 bis 127 (anderer „Typ“, andere Bedeutung)

```
char a, b;
a = 97;       // ASCII Code für 'a'
```

Typen und Variablen

`char`: Zeichen („charcter“)

- Erlaubt das Speichern eines Zeichens
  (Buchstaben werden durch Zahlen repräsentiert)
- Typischerweise: 8 Bit
- „Wertebereich“: –128 bis 127 (anderer „Typ“, andere Bedeutung)

```
char a, b;
a = 97;      // ASCII Code für 'a'
b = 'a';
```

Typen und Variablen

`char`: Zeichen („charcter“)

- Erlaubt das Speichern eines Zeichens
  (Buchstaben werden durch Zahlen repräsentiert)
- Typischerweise: 8 Bit
- „Wertebereich“: –128 bis 127 (anderer „Typ“, andere Bedeutung)

```
char a, b;
a = 97;        // ASCII Code für 'a'
b = 'a';
printf("a = %c and b = %c \n", a, b);
```

Typen und Variablen

`char`: Zeichen („charcter“)

- Erlaubt das Speichern eines Zeichens
  (Buchstaben werden durch Zahlen repräsentiert)
- Typischerweise: 8 Bit
- „Wertebereich“: –128 bis 127 (anderer „Typ“, andere Bedeutung)

```
char a, b;
a = 97;        // ASCII Code für 'a'
b = 'a';
printf("a = %c and b = %c \n", a, b);
```

Ausgabe: `a = a and b = a`

Ausgaben: Formatzeichen

- Wichtige Formatzeichen
  - die meisten sind derzeit noch unverständlich
  - die Erklärungen kommen im Laufe der Vorlesung

`%c`	`Einzelzeichen`	`char`
`%d`	`Integer`	`int`
`%u`	`Unsigned Integer`	`unsigned int`
`%lu`	`Unsigned Long`	`long`
`%ld`	`Integer`	`long int`
`%lld`	`Integer`	`long long int`
`%f`	`Gleitkommazahl`	`float`
`%lf`	`Gleitkommazahl`	`double`
`%s`	`Zeichenkette/String`	`char *`

Ausgaben: Sonderzeichen

- Wichtige Sonderzeichen

```
\n   Newline, Zeilensprung
\t   Tabulator
\0   EOS - Endezeichen in String
```

- Maskierung (Escaping) von reservierten Zeichen

```
\'   einfaches Anführungszeichen '
\"   doppeltes Anführungszeichen "
%%   Prozentzeichen %
\\   Backslash \
```

Variablen und Typen

- In C ist jede **Variable** von einem bestimmten **Typ**.
- Der Typ gibt die Menge der Werte an, die eine Variable annehmen kann.
  - `int year` bedeutet, dass die Variable `year` nur ganzzahlige Werte (integer) annehmen kann.
- Der Typ gibt an, welche Operatoren auf eine Variable angewendet werden können.

- Jede Variable **muss** vor ihrer ersten Verwendung **deklariert** werden.

```
int month;   // Deklaration
month = 10;  // Initialisierung mit einem Wert (Zuweisung)

int year = 2006;  // Deklaration und Initialiserung in
                  // 1 Schritt
```

Logische Ausdrücke (boolean expressions)

- Für logische Ausdrücke gibt es in C keinen speziellen Typ.
  - Wert == 0 ⇒ false / falsch
  - Wert != 0 ⇒ true / wahr

- Vergleichsoperatoren liefern Integer-Werte 0 oder 1:
  - **==** gleich
  - **!=** ungleich
  - **<** kleiner
  - **>** größer
  - **<=** kleiner gleich
  - **>=** größer gleich