

C-Kurs

Die ersten Schritte

Programm vs. Algorithmus

- ❑ **Algorithmen** beschreiben was ein Computer ausführen soll (in schematischer Form)
- ❑ **Programmiersprachen** stellen eine Schnittstelle dar, um die Algorithmen auf dem Computer ausführen zu können

Programm vs. Algorithmus (2.)

- ❑ **Algorithmen** fokussieren auf Korrektheit, Vollständigkeit, und Komplexität
- ❑ **Programmiersprachen** müssen zusätzlich alle Details des Computers berücksichtigen

Beispiel: Kuchen backen

- ❑ Input: Zutaten
 - ❑ Softwareumgebung/Datenstrukturen: Werkzeuge
 - ❑ Algorithmus: Rezept
-
- ❑ Hardware: Ofen
 - ❑ Output: Kuchen 😊



Beispielalgorithmus: Zweier Potenzen

□ Berechne die zweier Potenzen bis n:

Sei m gleich 0

Sei p gleich 1

Solange p kleiner n ist, tue:

Gib „2 hoch m ist p“ auf der Konsole aus

Addiere zu m den Wert 1 und speichere das Resultat in m

Multipliziere p mit dem Wert 2 und speichere das Resultat in p

□ Der Algorithmus ist in natürlich sprachlich in einer Art
Pseudocode beschrieben!

□ Warum geben wir 2 hoch m zuerst aus,
bevor m erhöht wird?

Beispielalgorithmus: Zweier Potenzen in Pseudocode

□ Berechne die zweier Potenzen 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

- ❑ Ein Algorithmus ist eine Liste von Anweisungen, 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 Zweier Potenzen aus?
 - **Effizienz**: Wie viel Zeit und wie viel Speicherplatz braucht er?
 - **(Terminierung)**: Hält der Algorithmus immer an?)

Elemente von Pseudocode am Beispiel: Zweier Potenzen

□ Algorithmus: Gebe zweier Potenzen bis n aus:

$m \leftarrow 0,$

$p \leftarrow 1,$

while ($p < n$)

Ausgabe von: „ 2^m ist p “;

$p \leftarrow p * 2;$

$m \leftarrow m + 1;$

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");  
}
```

```
$ gcc -Wall -std=c99 -o hello hello.c
```

```
$ ./hello  
Hello World.
```

Erstes C Programm

Zweier Potenzen

Beispielalgorithmus: Zweier Potenzen

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;
    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 einfachste C Programm besteht nur aus einer Funktion: `main` Das ist der Einsprungspunkt
- ❑ Das untenstehende Programm ist ein korrektes Programm, das jedoch „leer“ ist, es tut nichts.

```
int main ( ) {  
  
}
```

Typen und Variablen

□ Variablen:

- Die Basiselemente eines Programms
- Erlauben 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 */`
- **char a;** `/* Variable a vom Typ Character */`

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, ..**
- **=, +, >, <, ...**
- **while, for, if, else, ...**
- **/*, */, //**

/* Typen */

/* Mathematische Operationen */

/* Kontrollstrukturen */

/* Kommentare */

Typen und Variablen

int: Ganze Zahl

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

```
#include <limits.h>
```

```
int x, y;
```

```
x = 2014;
```

```
y = INT_MAX;
```

```
printf("x = %d and y = %d \n", x, y);
```

Einschub: 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");`

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

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

- ❑ Platzhalter Beispiele:

`%d` **Integer**

`int`

`%c` **Einzelzeichen**

`char`

Einschub: Formatzeichen

□ Wichtige Formatzeichen:

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

Einschub: Sonderzeichen

❑ Wichtige Sonderzeichen

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

❑ Maskierung (Escaping) von reservierten Zeichen

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

Typen und Variablen

□ char:

- Erlaubt das Speichern eines Zeichens
(Buchstaben werden durch Zahlen repräsentiert)
- Typischerweise: 8 Bit
- Wertebereich: – 128 bis 127

```
char a, b;
```

```
a = 97;
```

```
b = 'a';
```

```
printf("a = %c and b = %c \n", a, b);
```

□ Weist einer Variablen einen Wert zu:

➤ Operator: =

□ Beispiele:

➤ **int x, y;** // Variablen x und y 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

➤ ...

□ 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	<input data-bbox="1238 882 1367 929" type="text" value="?"/>	
Zustand	x	<input data-bbox="1238 961 1367 1008" type="text" value="5"/>	
Zustand	x	<input data-bbox="1238 1039 1367 1086" type="text" value="5"/>	y <input data-bbox="1545 1039 1675 1086" type="text" value="?"/>

Berechnung, u.a.: Mathematische Operationen

□ Standardsatz an Operationen:

- Basisoperatoren $+$, $-$, $*$, $/$, $\%$, ...
- Erlaubt auf Daten mit Hilfe von Variablen zu rechnen

□ 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.

Berechnung, u.a.: Weitere mathematische Operationen

□ Logische Operationen:

- Rechnen mit Wahrheitswerten (true, false)
- Logisches „und“, „oder“, ...

□ Vergleiche:

- Vergleichen zweier Werte
- Kleiner: $<$, größer: $>$, kleiner gleich: \leq , größer gleich: \geq , ...
- Gleichheit: $==$

□ Beispiele:

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

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 Verwendung deklariert werden.

```
int year = 2006; // declaration and initialization
```

```
int month;           // declaration  
month = 10;          // initialization
```

Ausdrücke (expressions)

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

```
int year = 2000;           // declaration and initialization
year = year + 6;           // evaluation of expression
```

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 den mathematischen Ausdrücken entspricht.
- ❑ Ausdrücke werden durch Einsetzen der aktuellen Werte ausgewertet

```
int celsius = 0;
```

```
int fahrenheit = 93;
```

Zustand	fahrenheit	<input type="text" value="93"/>	celsius	<input type="text" value="0"/>
----------------	------------	---------------------------------	---------	--------------------------------

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

Zustand	fahrenheit	<input type="text" value="93"/>	celsius	<input type="text" value="33"/>
----------------	------------	---------------------------------	---------	---------------------------------

Beispiel: Swap

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

```
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

Falsch!

Beispiel: Swap

- ❑ Man braucht eine Hilfsvariable zum Zwischenspeichern

```
// 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.“
- ❑ Lesbarkeit für andere Programmierer (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 Programmierer (Autor) selbst, der 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, wie z.B. Java
- ❑ Aber folgende Konventionen sind sinnvoll:
 - Kommentare zu jeder Funktion

```
/*  
 * Calculate distance of point (a,b) to origin  
 */  
int dist_to_organ( 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.

➤ 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 „true“, führe Anweisung(en) aus.

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

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

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

□ Beispiel: Zählt von 0 bis 10.

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

while-Schleifen

□ Beispiel: Zählt von 0 bis 10.

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

Zusammenfassung

Beispielalgorithmus: Zweier Potenzen

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;
    while (p < n) {
        printf("2^%d ist %d", m, p);
        m = m + 1;
        p = p * 2;
    }
}
```