



C Programmierkurs

2. Stunde: Schleifen und Matrizen

Johannes Hayeß & Mirko Seibt

1. November 2018

Technische Universität Dresden

Ergänzung zur letzten Stunde

PC-Pools:

Alles was nicht in eurem Verzeichnis abgespeichert ist, wird beim Abmelden gelöscht.

```
cd ~/<s-nummer>-home
```

Windows:

```
cd /mnt/c # für C:\
```

Kleiner Unix-Exkurs

Ziel der heutigen Stunde

$$c_{ij} = \sum_{k=1}^n a_{ik} b_{kj}$$

Unser erstes C-Programm (Wiederholung)

Unser erstes C-Programm

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  /* The main thing that this program does. */
5  int main(void) {
6  // Declarations
7      double A[5] = {
8          [0] = 9.0,
9          [1] = 2.9,
10         [4] = 3.E+25,
11         [3] = .00007,
12     };
13     // Doing some work
14     for (size_t i = 0; i < 5; ++i) {
15         printf("element %zu is %g,\ntits square is %g\n", i, A[i], A[i] * A[i]);
16     }
17
18     return EXIT_SUCCESS;
19 }
```

vgl. Gustedt, *Modern C*, Seite 2

Includes:

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

Präprozessor-Anweisungen

Includes:

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

Kopiert Inhalte an den Beginn der Datei (copy & paste)

Präprozessor-Anweisungen

Includes:

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

Kopiert Inhalte an den Beginn der Datei (copy & paste)

Bibliotheken stellen zusätzliche Funktionen zur Verfügung

Präprozessor-Anweisungen

Includes:

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

Kopiert Inhalte an den Beginn der Datei (copy & paste)

Bibliotheken stellen zusätzliche Funktionen zur Verfügung

stdio.h liefert die Funktion:

```
printf();
```

Präprozessor-Anweisungen

Includes:

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

Kopiert Inhalte an den Beginn der Datei (copy & paste)

Bibliotheken stellen zusätzliche Funktionen zur Verfügung

stdio.h liefert die Funktion:

```
printf();
```

stdlib.h beinhaltet die Konstante:

```
EXIT_SUCCESS
```

Hauptfunktion:

```
int main(void) {  
    [Anweisung];  
    ...  
    return EXIT_SUCCESS;  
}
```

Jedes C-Programm enthält **genau eine** main-Funktion!

Allgemeines

Hauptfunktion:

```
int main(void) {  
    [Anweisung];  
    ...  
    return EXIT_SUCCESS;  
}
```

Jedes C-Programm enthält **genau eine** main-Funktion!

Anweisungen:

```
[Anweisung];  
  
/* Beispiel */  
printf("Hello World!");
```

For-Schleife

Grundsätzlicher Aufbau:

```
for([Initialisierung]; [Bedingung]; [Zählfunktion]) {  
    [Anweisung];  
    ...  
}
```

Beispiele: For-Schleifen

```
for (size_t i = 0; i < 5; ++i) {  
    printf("%zu. Ausgabe\n", i + 1);  
}
```

Beispiele: For-Schleifen

```
for (size_t i = 0; i < 5; ++i) {  
    printf("%zu. Ausgabe\n", i + 1);  
}
```

```
for (size_t i = 0, j = 5; i < j; ++i) { }
```

Beispiele: For-Schleifen

```
for (size_t i = 0; i < 5; ++i) {  
    printf("%zu. Ausgabe\n", i + 1);  
}
```

```
for (size_t i = 0, j = 5; i < j; ++i) { }
```

```
for (size_t i = 5; i; --i) { }
```

Beispiele: For-Schleifen

```
for (size_t i = 0; i < 5; ++i) {  
    printf("%zu. Ausgabe\n", i + 1);  
}
```

```
for (size_t i = 0, j = 5; i < j; ++i) { }
```

```
for (size_t i = 5; i; --i) { }
```

```
for(;;) { }
```

Variablen

Variablen

Deklaration:

```
[Typ] [Name];  
  
/* Beispiel */  
size_t i;
```

Variablen

Deklaration:

```
[Typ] [Name];  
  
/* Beispiel */  
size_t i;
```

Wertzuweisung:

```
[Variable] = [Wert];  
  
/* Beispiel */  
i = 0;
```

Variable und Wert sollten vom **selben** Typ sein!

Variablen

Variablen sollten **immer** initialisiert werden:

```
[Typ] [Name] = [Intitalisierung];
```

```
/* Beispiel */
```

```
size_t i = 0;
```

Variablen

Variablen sollten **immer** initialisiert werden:

```
[Typ] [Name] = [Intitalisierung];
```

```
/* Beispiel */
```

```
size_t i = 0;
```

Für Variablennamen sind **nur** englische Buchstaben, Zahlen sowie _ zugelassen. Also kein ä, ö, ü, ß oder Ähnliches

Variablen

Variablen sollten **immer** initialisiert werden:

```
[Typ] [Name] = [Intitalisierung];
```

```
/* Beispiel */
```

```
size_t i = 0;
```

Für Variablennamen sind **nur** englische Buchstaben, Zahlen sowie `_` zugelassen. Also kein `ä`, `ö`, `ü`, `ß` oder Ähnliches
Schlüsselwörter sind **nicht** erlaubt!

z.B. `for`, `return`, `void`, `int`, `double`, ...

Variablen

Variablen sollten **immer** initialisiert werden:

```
[Typ] [Name] = [Intitalisierung];
```

```
/* Beispiel */
```

```
size_t i = 0;
```

Für Variablennamen sind **nur** englische Buchstaben, Zahlen sowie `_` zugelassen. Also kein `ä`, `ö`, `ü`, `ß` oder Ähnliches
Schlüsselwörter sind **nicht** erlaubt!

z.B. `for`, `return`, `void`, `int`, `double`, ...

Stile: `camelCaseIdentifiers` oder `snake_case_identifiers`

Variablen

Variablen sollten **immer** initialisiert werden:

```
[Typ] [Name] = [Intitalisierung];
```

```
/* Beispiel */
```

```
size_t i = 0;
```

Für Variablennamen sind **nur** englische Buchstaben, Zahlen sowie _ zugelassen. Also kein ä, ö, ü, ß oder Ähnliches
Schlüsselwörter sind **nicht** erlaubt!

z.B. for, return, void, int, double, ...

Stile: camelCaseIdentifiers oder snake_case_identifiers

Gebt Euren Variablen **sinnvolle** Namen wie: height, width, ...

Festkommazahlen

Schlüsselwörter: **short**, **int**, **long**

Schlüsselwörter: **short**, **int**, **long**

Werden als Binärzahl mit **fester** Länge gespeichert

Schlüsselwörter: **short**, **int**, **long**

Werden als Binärzahl mit **fester** Länge gespeichert

Können vorzeichenbehaftet (**signed**)(voreingestellt) oder vorzeichenlos (**unsigned**) sein

Schlüsselwörter: **short**, **int**, **long**

Werden als Binärzahl mit **fester** Länge gespeichert

Können vorzeichenbehaftet (**signed**)(voreingestellt) oder vorzeichenlos (**unsigned**) sein

Wertebereiche sind abhängig von der Rechnerarchitektur!

Beispiele für 64Bit-Architekturen:

int von -2.147.483.648 bis 2.147.483.647

unsigned short von 0 bis 65535

Wahrheitswerte

Zweistelliger Ausdruck:

```
[Operant] [Operator] [Operant] -> {true, false}
```

Werte vergleichen

Zweistelliger Ausdruck:

```
[Operant] [Operator] [Operant] -> {true, false}
```

Operatoren: <, >, >=, <=, ==, !=

Wichtig: = für Zuweisungen, == für Vergleiche

Werte vergleichen

Zweistelliger Ausdruck:

```
[Operant] [Operator] [Operant] -> {true, false}
```

Operatoren: <, >, >=, <=, ==, !=

Wichtig: = für Zuweisungen, == für Vergleiche

Beispiele:

```
for (size_t i = 0; i < 5; ++i) { }
```

Werte vergleichen

Zweistelliger Ausdruck:

```
[Operant] [Operator] [Operant] -> {true, false}
```

Operatoren: <, >, >=, <=, ==, !=

Wichtig: = für Zuweisungen, == für Vergleiche

Beispiele:

```
for (size_t i = 0; i < 5; ++i) { }
```

$5 < 7 \mapsto \text{true};$

Einstelliger Ausdruck:

```
[Operand] -> {true, false}
```


Einstelliger Ausdruck:

```
[Operand] -> {true, false}
```

Es gilt: $0 \mapsto \text{false}$, $\text{sonst} \mapsto \text{true}$

Einstelliger Ausdruck:

```
[Operand] -> {true, false}
```

Es gilt: $0 \mapsto \text{false}$, $\text{sonst} \mapsto \text{true}$

Beispiel:

```
for (size_t i = 5; i; --i) { }
```

Einstelliger Ausdruck:

```
[Operand] -> {true, false}
```

Es gilt: $0 \mapsto \text{false}$, $\text{sonst} \mapsto \text{true}$

Beispiel:

```
for (size_t i = 5; i; --i) { }
```

$(5 < 7) == 1 \mapsto \text{true}$

Arithmetik

Grundsätzlicher Aufbau:

[Operant] [Operator] [Operant]

Arithmetische Funktionen

Grundsätzlicher Aufbau:

[Operant] [Operator] [Operant]

Operatoren: +, -, *, /, % (modulo)

Grundsätzlicher Aufbau:

[Operant] [Operator] [Operant]

Operatoren: +, -, *, /, % (modulo)

Es gelten die bekannten Gesetze:

Punkt vor Strichrechnung, Klammersetzung, ...

Arithmetische Funktionen

Grundsätzlicher Aufbau:

```
[Operant] [Operator] [Operant]
```

Operatoren: +, -, *, /, % (modulo)

Es gelten die bekannten Gesetze:

Punkt vor Strichrechnung, Klammersetzung, ...

Ergebnisse müssen wieder gespeichert werden:

```
[Variable] = [Arithmetischer Ausdruck];
```


Beispiele: Arithmetische Funktionen

```
int a = 5;
```

Beispiele: Arithmetische Funktionen

```
int a = 5;
```

```
int b = a * 2;
```

Beispiele: Arithmetische Funktionen

```
int a = 5;
```

```
int b = a * 2;
```

```
int c = (a + b) % 2;
```

Beispiele: Arithmetische Funktionen

```
int a = 5;
```

```
int b = a * 2;
```

```
int c = (a + b) % 2;
```

```
int d = a / 2; // Vorsicht!
```

Kurzschreibweisen

Kurzschreibweisen

```
int l = 1, s = 1;  
l = l * 5; /* lang */  
s *= 5; /* kurz */
```

Kurzschreibweisen

```
int l = 1, s = 1;  
l = l * 5; /* lang */  
s *= 5; /* kurz */
```

```
int l = 0, s = 0;  
l = l + 1; /* lang */  
++s; /* kurz */
```

Kurzschreibweisen

```
int l = 1, s = 1;  
l = l * 5; /* lang */  
s *= 5; /* kurz */
```

```
int l = 0, s = 0;  
l = l + 1; /* lang */  
++s; /* kurz */
```

Prä- vs. Postinkrement/-dekrement:

```
int s = 0;  
int x = ++s; /* x == 1, s == 1 */  
int y = s++; /* y == 0, s == 1 */
```


Vorsicht: Wertebereiche könnten überschritten werden!

Vorsicht: Wertebereiche könnten überschritten werden!

```
unsigned short i = 0 - 1;
```

Vorsicht: Wertebereiche könnten überschritten werden!

```
unsigned short i = 0 - 1;
```

Ergebnis:

```
i == 65535
```

```
i != -1
```

Arrays

Grundsätzlicher Aufbau:

```
[Typ] [Name][[Länge]] = { [[Feld]] = [Wert], ... };
```

Arrays

Grundsätzlicher Aufbau:

```
[Typ] [Name][[Länge]] = { [[Feld]] = [Wert], ... };
```

Beispiel:

```
double A[5] = {  
    [0] = 9.0,  
    [1] = 2.9,  
    [4] = 3.E+25,  
    [3] = .00007,  
};
```

Vergleichbar mit Vektoren

$$x = (x_1, x_2, \dots, x_n)^T$$

Arrays

Arrays beginnen mit 0!

Arrays

Arrays beginnen mit 0!

Besitzt n Elemente

```
int A[n];
```


Arrays

Arrays beginnen mit 0!

Besitzt n Elemente

```
int A[n];
```

$$\{x_0, x_1, \dots, x_{n-1}\}$$

Arrays

Arrays beginnen mit 0!

Besitzt n Elemente

```
int A[n];
```

$\{x_0, x_1, \dots, x_{n-1}\}$ Beispiele:

```
int A[5];  
for (size_t i = 1; i <= 5; ++i) { A[i-1] = 0; }
```

Arrays

Arrays beginnen mit 0!

Besitzt n Elemente

```
int A[n];
```

$\{x_0, x_1, \dots, x_{n-1}\}$ Beispiele:

```
int A[5];  
for (size_t i = 1; i <= 5; ++i) { A[i-1] = 0; }
```

```
int A[5];  
for (size_t i = 0; i < 5; ++i) { A[i] = 0; }
```

Mehrdimensionale Arrays

Idee: Arrays in Arrays

```
int A[5];  
int B[5];  
int* C[2] = { A, B };
```

Mehrdimensionale Arrays

Idee: Arrays in Arrays

```
int A[5];  
int B[5];  
int* C[2] = { A, B };
```

Das geht aber auch einfacher

```
int A[i][j] = { [0][0] = 1, ... };  
int A[i][j] = {{1,2,...}, {3,4,...}, ...};
```

Aufgabenstellung

Aufgabenstellung

Implementierung der Matrixmultiplikation für quadratische Matrizen fester Größe:

Implementierung der Matrixmultiplikation für quadratische Matrizen fester Größe:

$$c_{ij} = \sum_{k=1}^n a_{ik} b_{kj}$$

Aufgabenstellung

Implementierung der Matixmultipliktion für quadratische Matrizen fester Größe:

$$c_{ij} = \sum_{k=1}^n a_{ik} b_{kj}$$

Hilfsmittel:

```
/* Deklaration, Zuweisungen */  
[typ] [name] = [wert];  
/* Arrays */  
[typ] [name][n][m] = { [values] };  
/* For-Schleifen */  
for([init]; [bedingung]; [zähler]) { }  
/* Ausgabe */  
printf("| %d %d %d |", m[0][i], m[1][i], m[2][i]);
```

Matrixmultiplikation mit mehrdimensionalen Arrays

Gleitkommazahlen

Schlüsselwörter: `float`, `double`, `long double`

Schlüsselwörter: **float**, **double**, **long double**

Speicherung nach IEEE 754 Standard

pause Zur Darstellung wird statt dem Komma ein Punkt verwendet!

Beispiele: 2.9, 3.E+25, .00007

Man unterscheidet die Typen auch direkt in den Werten.

0.0F (Typ **float**), 0.0 (Typ **double**), 0.0L (Typ **long double**)

Schlüsselwörter: **float**, **double**, **long double**

Speicherung nach IEEE 754 Standard

pause Zur Darstellung wird statt dem Komma ein Punkt verwendet!

Beispiele: 2.9, 3.E+25, .00007

Man unterscheidet die Typen auch direkt in den Werten.

0.0F (Typ **float**), 0.0 (Typ **double**), 0.0L (Typ **long double**)

Gleitkommazahlen repräsentieren **nie** den exakten Wert, sondern nur eine präzisions-, rechnerabhängige Annäherung!

0.2 könnte im Rechner gespeichert sein als

0.20000000000000000111

Buchstaben/Zeichen

Schlüsselwort: **char**

Schlüsselwort: **char**

Können vorzeichenbehaftet sein (voreingestellt)

Speicherung: 1 Byte (8Bit) pro Zeichen

Schlüsselwort: **char**

Können vorzeichenbehaftet sein (voreingestellt)

Speicherung: 1 Byte (8Bit) pro Zeichen

Repräsentieren genau einen Buchstaben

Zeichen werden ASCII kodiert

A \mapsto 65 Beispiele:

char a = 'A'; oder **char** b = 65;



Jens Gustedt. *Modern C*. Lizenz: CC BY-NC-ND 4.0. URL: <https://creativecommons.org/licenses/by-nc-nd/4.0/>.



Richard Mörbitz. *C-Kurs TU-Dresden 2017*. Inspirierte den Abschnitt für Festkommazahlen, Variablen, Arithmetik und boolsche Vergleiche. Lizenz: CC BY 4.0. URL: <http://creativecommons.org/licenses/by/4.0/>.

Diese Präsentation ist lizenziert unter der **Creative Commons Attribution-ShareAlike 4.0 International** Lizenz.

