

Ein- und Ausgabe

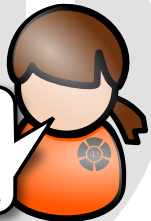
C - Kurs 2010


Mario Bodemann

15. September 2010



This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 3.0 License



- 
- 1 Wiederholung
 - 2 Ausgabe
 - Einschub: Dateien
 - 3 Eingabe
 - 4 Beispiel
 - 5 Ausblick

1 Wiederholung

2 Ausgabe

- Einschub: Dateien

3 Eingabe

4 Beispiel

5 Ausblick



4!

- Was gelehrt wurde
 - Syntax von C
 - Typen
 - Operatoren
 - Kontrollfluss
 - Funktionen
 - Hello World

A large, light gray circular graphic containing the text '4!' in a bold, white, sans-serif font. The background of the slide features a large, stylized gear or sun-like pattern with several triangular segments.

Listing 1: src/recap.c

```
13  int a = 4;  
14  int b = 4 + a;  
15  int c = ++b + a++;  
16  int d = (a == b ? c : --b + c);  
  
17  
18  if( 9 == d )  
19  {  
20      d %= 8;  
21  }  
22  else  
23  {  
24      d <<= 2;  
25  }
```

Wiederholung: Hello World

Listing 2: src/hello.c

```
1 #include <stdio.h>
2 int main(int argc, char **argv)
3 {
4     printf("Hello_World\n");
5     return 0;
6 }
```

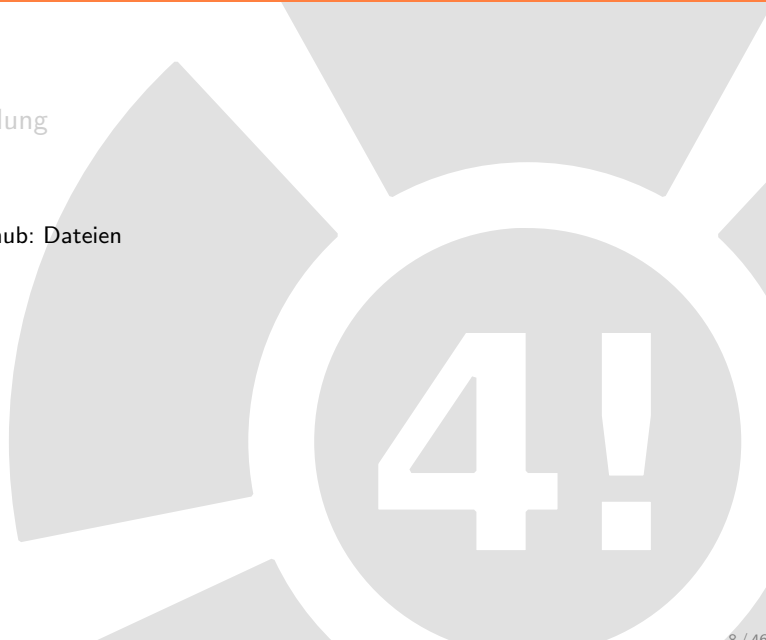
- Einfache Ausgabe
- Aber: Wo für steht das 'f' bei printf?

Ziele dieser Vorlesung

- Lerne, wie du Variablen ausgeben kannst
- Lerne, wie diese Ausgaben formatiert werden
- Speichere und lade Variablen in/aus Dateien
- Alle Beispiele sind online¹ verfügbar

A large, light gray graphic in the background of the slide. It consists of a circle containing a large white number '4' followed by a white exclamation mark '!'.

¹www.freitagsrunde.org/Ckurs2010/Vortrag02

- 
- 1 Wiederholung
 - 2 Ausgabe**
 - **Einschub: Dateien**
 - 3 Eingabe
 - 4 Beispiel
 - 5 Ausblick

- Dient zur Ausgabe von Text und Variablen
- printf heißt die Funktion
- Das f steht für formatted, also für formatierte Ausgabe
- printf kann `int`, `float`, `double`, Strings und weitere Typen ausgeben

4!

Beispiel *printf*

Beispiel: Ausgaben

Listing 3: src/output.c

```
14 printf("      Name | Anzahl |      Gewicht\n");  
15 printf("      +-----+-----+\n");  
16 printf(" %s | %d | %f\n", "Samus", 12, 2.718 );
```



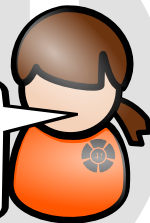
Beispiel *printf*

Beispiel: Ausgaben

Listing 4: src/output.c

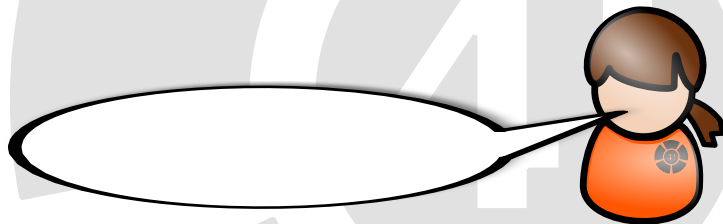
```
14 printf("      Name | Anzahl |      Gewicht\n");  
15 printf("      +-----+-----+\n");  
16 printf(" %s | %d | %f\n", "Samus", 12, 2.718 );
```

Name	Anzahl	Gewicht
Samus	12	2.718000



Typen für printf

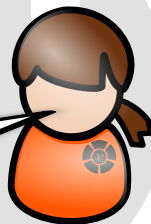
Symbol	Beschreibung	Bespielsausgabe
%d	ganze Zahlen	3
%x	ganze Zahlen in Hex	0xD0DAAFFE
%f	gebrochene Zahlen	3.1415927
%s	Strings	HalloWelt
%%	Ausgabe eines %	100%



Typen für printf

Symbol	Beschreibung	Bespielsausgabe
%d	ganze Zahlen	3
%x	ganze Zahlen in Hex	0xD0DAAFFE
%f	gebrochene Zahlen	3.1415927
%s	Strings	HalloWelt
%%	Ausgabe eines %	100%

Für alle weiteren
Typen siehe *Manpage*:
man 3 printf



Rechtsbündig

- Wir wollen eine Tabelle ausgeben
- Ausgabe ist falsch
- Vorschlag: rechtsbündige Ausgabe

Beispiel: Ausgabe rechtsbündig

Listing 5: src/output.c

```
16 printf("  %s |  %d |  %f\n", "Samus", 12, 2.718 );  
17 printf(" %9s | %6d | %12f\n", "Tails", 1, 1.0 );
```



Rechtsbündig

- Wir wollen eine Tabelle ausgeben
- Ausgabe ist falsch
- Vorschlag: rechtsbündige Ausgabe

Beispiel: Ausgabe rechtsbündig

Listing 6: src/output.c

```
printf(" %s | %d | %f\n", "Samus", 12, 2.718 );  
printf(" %9s | %6d | %12f\n", "Tails", 1, 1.0 );
```

Name	Anzahl	Gewicht
Samus	12	2.718000
Tails	1	1.000000



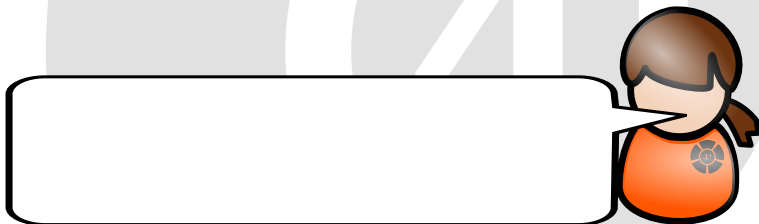
Auffüllung

- Ändert das Zeichen, mit dem aufgefüllt wird
- Sinnvoll bei rechtsbündiger Ausgabe
- Möglich: ' ' und '0' (Standard ist ' ')

Beispiel: Ausgeben von Werten

Listing 7: src/output.c

```
printf("  %9s  | %6d  | %12f\n", "Tails", 1, 1.0 );  
printf("  %09s  | %06d  | %012f\n", "Toad", 43, 31.33 );
```



Auffüllung

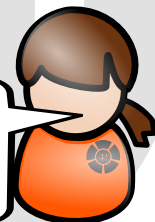
- Ändert das Zeichen, mit dem aufgefüllt wird
- Sinnvoll bei rechtsbündiger Ausgabe
- Möglich: ' ' und '0' (Standard ist ' ')

Beispiel: Ausgeben von Werten

Listing 8: src/output.c

```
printf("  %9s | %6d | %12f\n", "Tails", 1, 1.0 );  
printf("  %09s | %06d | %012f\n", "Toad", 43, 31.33 );
```

```
Samus | 12 | 2.718000  
  Tails |      1 |      1.000000  
  Toad | 000043 | 00031.330000
```



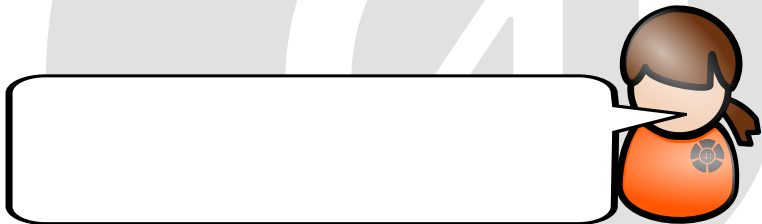
Linksbündig

- '-' richtet Ausgabe linksbündig aus

Beispiel: Ausgeben von Werten

Listing 9: src/output.c

```
printf(" _%09s | _%06d | _%012f\n", "Toad", 43, 31.33 );  
printf(" _%-9s | _%-6d | _%012f\n", "GLaDOS", 87, 3.14 );
```



Linksbündig

- '-' richtet Ausgabe linksbündig aus

Beispiel: Ausgeben von Werten

Listing 10: src/output.c

```
printf("  _%09s |  _%06d |  _%012f\n", "Toad", 43, 31.33 );  
printf("  _%-9s |  _%-6d |  _%-12f\n", "GLaDOS", 87, 3.14 );
```

Tails		1		1.000000
Toad		000043		00031.330000
GLaDOS		87		3.140000



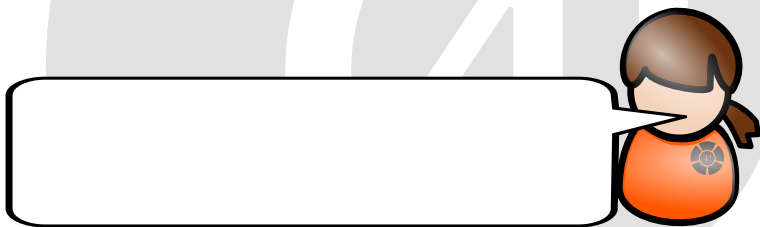
Nachkommastellen

- Ausgabe wird normalerweise auf 6 Stellen gerundet
- Zuerst %, dann evtl nen Punkt, Anzahl der Stellen

Beispiel: Ausgeben von Werten

Listing 11: src/output.c

```
printf(" %-9s | %-6d | %-12f\n", "GLaDOS", 87, 3.14 );  
printf(" %-9s | %-6d | %-12.1f\n", "Kerrigan", 3, -0.16 );
```



Nachkommastellen

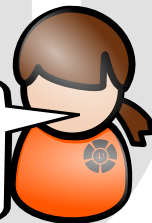
- Ausgabe wird normalerweise auf 6 Stellen gerundet
- Zuerst %, dann evtl nen Punkt, Anzahl der Stellen

Beispiel: Ausgeben von Werten

Listing 12: src/output.c

```
printf(" %-9s | %-6d | %-12f\n", "GLaDOS", 87, 3.14 );  
printf(" %-9s | %-6d | %-12.1f\n", "Kerrigan", 3, -0.16 );
```

Toad	000043	00031.330000
GLaDOS	87	3.140000
Kerrigan	3	-0.2



Beispiel: Ausgeben von Werten

Listing 13: src/output.c

```
16 printf(" %s | %d | %f\n", "Samus", 12, 2.718 );  
17 printf(" %9s | %6d | %12f\n", "Tails", 1, 1.0 );  
18 printf(" %09s | %06d | %012f\n", "Toad", 43, 31.33 );  
19 printf(" %-9s | %-6d | %-12f\n", "GLaDOS", 87, 3.14 );  
20 printf(" %9s | %6d | %12.1f\n", "Kerrigan", 3, -0.16 );
```

Samus		12		2.718000
Tails				1.000000
Toad		000043		00031.330000
GLaDOS		87		3.140000
Kerrigan		3		-0.2

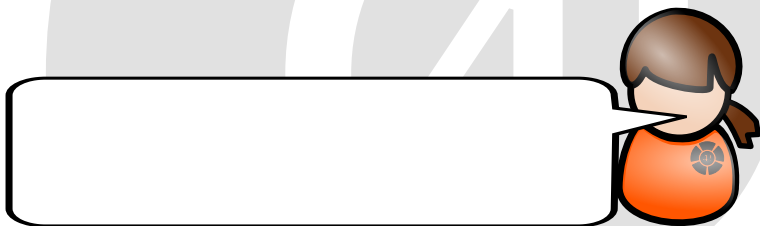


Aufgabe

Was geben folgende Zeilen aus:

Listing 14: src/exercise.c

```
14 float f = 3.1415926535897932384626433832795 f;  
15 printf(" PI = %12f \n", f);  
16 printf(" PI = %12.2f \n", f);  
17 printf(" PI = %012.7f \n", f);  
18 printf(" PI = %-012.7f \n", f);
```



Aufgabe

Was geben folgende Zeilen aus:

Listing 15: src/exercise.c

```
14 float f = 3.1415926535897932384626433832795 f;  
15 printf(" PI = %12f | \n", f);  
16 printf(" PI = %12.2f | \n", f);  
17 printf(" PI = %012.7f | \n", f);  
18 printf(" PI = %-012.7f | \n", f);
```

PI = 3.141593 |



Aufgabe

Was geben folgende Zeilen aus:

Listing 16: src/exercise.c

```
14 float f = 3.1415926535897932384626433832795 f;  
15 printf(" PI = %12f | \n", f);  
16 printf(" PI = %12.2f | \n", f);  
17 printf(" PI = %012.7f | \n", f);  
18 printf(" PI = %-012.7f | \n", f);
```

```
PI =      3.141593 |  
PI =           3.14 |
```



Aufgabe

Was geben folgende Zeilen aus:

Listing 17: src/exercise.c

```
14 float f = 3.1415926535897932384626433832795 f;  
15 printf(" PI = %12f \n", f);  
16 printf(" PI = %12.2f \n", f);  
17 printf(" PI = %012.7f \n", f);  
18 printf(" PI = %-012.7f \n", f);
```

```
PI =      3.141593 |  
PI =              3.14 |  
PI = 0003.1415927 |
```



Aufgabe

Was geben folgende Zeilen aus:

Listing 18: src/exercise.c

```
14 float f = 3.1415926535897932384626433832795 f;  
15 printf(" PI = %12f \n", f);  
16 printf(" PI = %12.2f \n", f);  
17 printf(" PI = %012.7f \n", f);  
18 printf(" PI = %-012.7f \n", f);
```

```
PI =      3.141593 |  
PI =           3.14 |  
PI = 0003.1415927 |  
PI = 3.1415927    |
```



Wohin mit dem Ergebnis?

- Nicht nur die Konsole kann ein Ziel sein!
- Strings können befüllt werden
- Dateien können befüllt werden
- Name der Funktion ändert sich
 - Bei Strings benutzt man `sprintf` (s = string)
 - Und bei Dateien benutzt man `fprintf` (f = file)
- Es kommt jeweils ein neuer Parameter hinzu

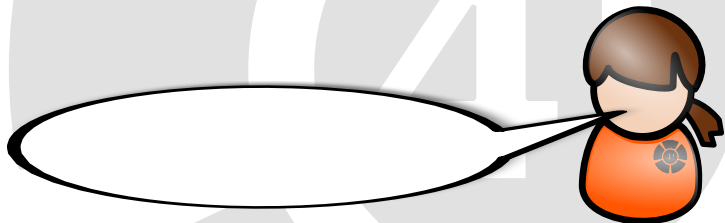
Ausgabe in einen String

- Strings werden hier nur kurz erwähnt
- Siehe dazu nächsten Vortrag

Ausgabe in String

Listing 19: src/sprintf.c

```
14 char pc[255];  
15 sprintf(pc, "PI = %.129f", 3.14f);
```



Ausgabe in einen String

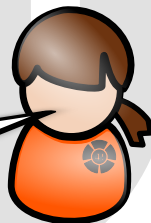
- Strings werden hier nur kurz erwähnt
- Siehe dazu nächsten Vortrag

Ausgabe in String

Listing 20: src/sprintf.c

```
14 char pc[255];  
15 sprintf(pc, "PI = %12.9f", 3.14f);
```

Inhalt von *pc*:
PI = 3.140000105



Ausgabe in eine Datei

Beispiel: Ausgabe in eine Datei

Listing 21: src/fprintf.c

```
fprintf(pFile , " HelloFile , _%10.1f\n" , 3.14f);
```

4!

Ausgabe in eine Datei

Beispiel: Ausgabe in eine Datei

Listing 22: src/fprintf.c

```
fprintf(pFile , " HelloFile , _%10.1f\n" , 3.14f);
```

pFile ???



1 Wiederholung

2 Ausgabe

- Einschub: Dateien

3 Eingabe

4 Beispiel

5 Ausblick

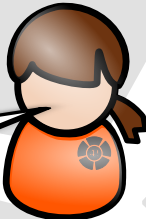


4!

- Schema zum lesen/schreiben einer Datei immer gleich
 - ❶ Öffnen der Datei mit *fopen*
 - ❷ Überprüfung ob Öffnen erfolgreich
 - ❸ Arbeiten auf der Datei mit *fprintf* oder *fscanf*
 - ❹ Schließen der Datei mit *fclose*

4!

Schritt 1: Öffnen einer Datei



```
FILE* fopen(  
    const char *path,  
    const char *mode )
```

- *FILE** ist der Rückgabetypp
 - Objekt, das die Datei repräsentiert
 - Kann definiert oder undefiniert (*NULL*) sein
- *path*
 - Name und Pfad der zu bearbeitenden Datei
- *mode*
 - "r": nur lesend (am Anfang der Datei)
 - "w": nur schreibend (erzeugt neue Datei)
 - "a": wie "w", nur schreibt ans Ende der Datei
 - "r+", "w+" oder "a+": lesend und schreibend

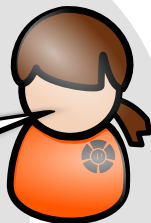
Schritt 1: Öffnen einer Datei - Beispiel

Beispiel: Ausgabe in eine Datei

Listing 23: src/fprintf.c

```
FILE *pFile = fopen("./test.dat", "w");
```

Öffnet *test.dat*
nur *schreibend*
und erzeugt *pFile*

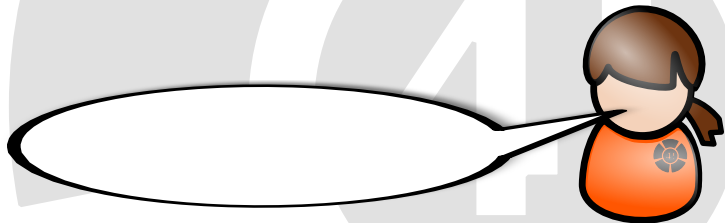


Schritt 2: Überprüfung

Beispiel: Ausgabe in eine Datei

Listing 24: src/fprintf.c

```
17 if( NULL == pFile )  
18 {  
19     // error handling ...
```



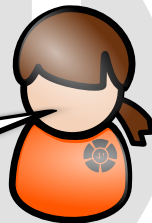
Schritt 2: Überprüfung

Beispiel: Ausgabe in eine Datei

Listing 25: src/fprintf.c

```
17 if( NULL == pFile )  
18 {  
19     // error handling ...
```

In dem then-Block des
`ifs` muss die Fehler-
auswertung kommen.

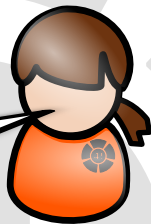


Schritt 3: Arbeit mit der Datei



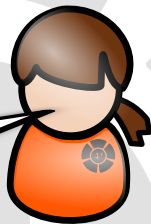
Schritt 3: Arbeit mit der Datei

Schreiben: *fprintf*




Schritt 3: Arbeit mit der Datei

Schreiben: *fprintf*
Lesen: *fscanf*

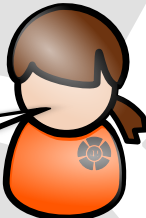


Schritt 3: Arbeit mit der Datei



Schreiben: *fprintf*
Lesen: *fscanf*
man 3 fread

Schritt 4: Schließen einer Datei



```
int fclose(  
FILE* pFile  
)
```

- Fehlercode (`int`) als Rückgabe
- *pFile* Dateiojekt, welches geschlossen werden soll
- Datei muss wieder geschlossen werden
- Danach kann das Dateiojekt nicht mehr benutzt werden

Listing 26: src/fprintf.c

```
13 // open a file
14 FILE *pFile = fopen("./test.dat", "w");
15
16 // check for errors
17 if( NULL == pFile )
18 {
19     // error handling ...
20     fprintf(stderr, "Konnte Datei nicht finden...\n");
21     return -1;
22 }
23
24 // write something to a file
25 fprintf(pFile, "HelloFile, %10.1f\n", 3.14f);
26
27 // now close it again
28 fclose(pFile);
```

```
stderr für Fehler  
stdout für Sonstiges  
printf entspricht fprintf(stdout ..  
stdin für Eingabe  
Alle niemals schließen!2
```



²Außer es gibt einen wichtigen Grund!

1 Wiederholung

2 Ausgabe

- Einschub: Dateien

3 Eingabe

4 Beispiel

5 Ausblick



4!

- Eingabe ist analog zu Ausgabe
- Statt `printf` benutzt man nun `scanf`
- Alle Typen von `printf` gelten auch für `scanf`
- Auch die Formatangaben (Linksbündigkeit, Stellenzahl, etc.) gelten

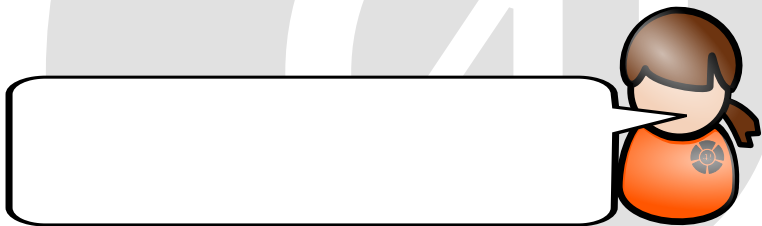
4!

Eingabe Beispiel

Beispiel: Eingabe zweier Werte

Listing 27: src/input.c

```
13 int a = 0;  
14 int b = 0;  
15 int iValues = scanf("%d %d", &a, &b);  
16 printf("a + b = %d\n", a+b);  
17 printf("iValues = %d\n", iValues);
```



Eingabe Beispiel

Beispiel: Eingabe zweier Werte

Listing 28: src/input.c

```
13 int a = 0;  
14 int b = 0;  
15 int iValues = scanf("%d %d", &a, &b);  
16 printf("a + b = %d\n", a+b);  
17 printf("iValues = %d\n", iValues);
```

a + b = ???
iValues = 2



Funktionsweise von *scanf*

- Es speichert die Eingabe des Benutzer
- Überprüft, ob die Eingabe zu den angegebenen Typen passt.
- Ignoriert Leerzeichen, Tabulatoren und Zeilenumbrüche
- Nicht Formatangaben müssen vorhanden sein (siehe Beispiel)
- Rückabewert beinhaltet Anzahl der erfolgreich gelesenen Werte.
- Daher: *Immer* den Rückgabewert überprüfen

4!

Probleme beim Einlesen

- `"%s"` liest alle Eingaben bis zur neuen Zeile
 - Auch andere Formatangaben werden überlesen (`"%d"`, `"%f"`)
 - Daher: Genauer formulieren mit bspw. `"%[A-Z]"`
 - Registriert alle Buchstaben zwischen A und Z
 - Für alle Zeichen: `"%[A-Za-z]"`
- `"%d"` und Buchstaben werden eingegeben?
 - Rückgabewert ist kleiner als erwartet
 - "Rest" der Eingabe liegt noch im Speicher, muss gelöscht werden
 - `scanf("%s", pcTemp);` liest den Rest ein

- Genau wie bei `printf` gibt es auch bei `scanf` weitere Varianten
- `sscanf` analog zu `sprintf`: liest aus einem String
- `fscanf` analog zu `fprintf`: liest aus einer Datei
 - *mode* muss dann mindestens `"r"` oder `"+"` sein
 - Reihenfolge von oben beachten

4!

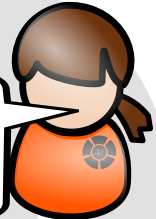
- 
- 1 Wiederholung
 - 2 Ausgabe
 - Einschub: Dateien
 - 3 Eingabe
 - 4 Beispiel**
 - 5 Ausblick

- Einlesen einer CSV³-Datei
- Benutzt als einfaches Datenbankformat
- Beispiel umfasst Datei lesen, schreiben und Ausgabe von Werten



³coma seperated values

- Einlesen einer CSV³-Datei
- Benutzt als einfaches Datenbankformat
- Beispiel umfasst Datei lesen, schreiben und Ausgabe von Werten



```
Turok; 21; 81.13;  
Gordon; 1231; 1.13;  
Tommy; 0; -0.12;
```

³coma seperated values

Beispiel: Lesen der CSV-Datei

Listing 29: src/csv.c

```
47 do
48 {
49     // read values
50     iltemsRead = fscanf(pFile, "[%A-Za-z]; %d; %f; ", &
        pcName, &iAmount, &fWeight );
51
52     // check values read
53     if( iltemsRead == 3 )
54     {
55         printf("%12s | %12d | %012.3f\n", pcName, iAmount,
            fWeight );
56     }
57 } while( iltemsRead == 3 );
```


Benutzer überprüfen

Beispiel: Benutzereingaben überprüfen

Listing 30: src/csv.c

```
78 printf("Anzahl: ");  
79 if( 1 != scanf("%d", &iAmount) )  
80 {  
81     fprintf(stderr, "Konnte Anzahl nicht lesen\n");  
82     scanf("%s", &pcName);  
83     continue;  
84 }
```

Beispiel: Benutzereingaben schreiben

Listing 31: src/csv.c

```
94 // everything read successfully
95 int iCharsWritten = fprintf( pFile , "%s ; %d ; %f ; \n" ,
    pcName, iAmount, fWeight );
96 if (iCharsWritten <= 0)
97 {
98     fprintf(stderr , "Konnte Daten nicht schreiben ! \n" );
99 }
```



Ausgabe des CSV Beispiels

```
Pigmin; 738; 0.0;  
Mii; 98; 10.20;  
Thaddaeus; 9999; 999;
```

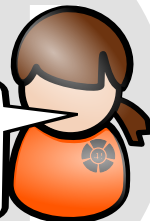


Ausgabe des CSV Beispiels

```
Pigmin; 738; 0.0;  
Mii; 98; 10.20;  
Thaddaeus; 9999; 999;
```



Name	Anzahl	Gewicht
Pigmin	738	00000000.000
Mii	98	00000010.200
Thaddaeus	9999	00000999.000



1 Wiederholung

2 Ausgabe

- Einschub: Dateien

3 Eingabe

4 Beispiel

5 Ausblick



4!

Zusammenfassung: Ein- und Ausgabe

- Ausgabe von Strings, ganzen Zahlen, gebrochenen Zahlen
- Ausgabe in Datei umleiten
- Ausgabe in String umleiten
- Eingabe = -Ausgabe
- Eingabe kann auch aus einer Datei oder aus einem String lesen.

4!

- Was bedeutet &?
- Was bedeutet *?
- Was ist genau ist char * oder FILE *?
- Was ist eine Struktur (bsp. FILE)?

A large, light gray circular graphic containing a white number '4' followed by a white exclamation mark '!', positioned on the right side of the slide.

Danke



- *man 3 printf*
- *man 3 scanf*
- *man 3 fopen*
- *man 3 fclose*
- *man 3 fread*

Alle Grafiken stammen aus der openCliparts Library und wurden von Mario Bodemann bearbeitet. Ausnahme bildet hierbei nur das Logo der Freitagsrunde. Dies wurde in einer Coproduktion von Mario Bodemann und Sebastian D. aus dem alten Logo der Freitagsrunde erstellt. Die Präsentationsvorlage darf weiter benutzt werden, jedoch besteht dabei keinerlei Garantie oder Gewährleistung.