

Input, Output , Dateien

C-Kurs 2013, 1. Tutorium

Armelle Vérité

<http://wiki.freitagrunde.org>

11. September 2013



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

- ▶ ISIS-Kurs (Infos, Links):
<https://www.isis.tu-berlin.de/course/view.php?id=8675>
- ▶ Vortragsfolien:
https://docs.freitagrunde.org/Veranstaltungen/ckurs_2013/unterlagen/
- ▶ Übungsaufgaben:
<http://wiki.freitagrunde.org/Ckurs/Übungsaufgaben>
- ▶ Buch „C von A bis Z“ (online frei verfügbar):
http://openbook.galileocomputing.de/c_von_a_bis_z/
- ▶ Online:
<http://www.cplusplus.com/reference/clibrary/cstdio>



4!

```
int a;
```

Eine Variable deklarieren

- ▶ Irgendwo ist ein Platz alloziert, der groß genug ist, um einen int zu speichern.
- ▶ Das ist die Adresse von a: `&a`
- ▶ manche Funktionen brauchen als Parameter eine Adresse, und keine variable. (Parameter mit `*`)

4!

Ausgabe mit printf

src/helloworld.c

```
1 int main()  
2 {  
3     printf(" Hello World!\n");  
4  
5     return 0;  
6 }
```

4!

Ausgabe mit printf

src/helloworld.c

```
1 int main()  
2 {  
3     printf(" Hello World!\n");  
4  
5     return 0;  
6 }
```

Signatur

`int printf (const char * format, ...)`

- ▶ Parameter: Formatstring *format* und beliebig viele Parameter jeden Basistyps
- ▶ Rückgabe(success): Anzahl geschriebene Zeichen
- ▶ Rückgabe(error): negativer Wert

Formatstring

...Kann sowohl Text als auch Formatzeichen enthalten.

src/printfDataTypes.c

```
1  const char *einString="I got you,";  
2  int  einInt=47806;  
3  double einDouble=12.345;  
4  
5  printf("\n\nYeah, %s %x\n", einString, einInt);
```

Formatzeichen

sind Platzhalter im
Formatstring für die
Parameter, die danach
angegeben werden

Formatzeichen	dargestellter Datentyp
%d	Integer
%f	Float (Fließkommazahl)
%x	Hexadezimale Ausgabe
%s	String

vollständige syntax

`%[flags][width][.precision][length]specifier`

- ▶ *flags* verschiedene Flags, wie Vorzeichen, etc
- ▶ *width* wie breit soll die Ausgabe sein (Anzahl Zeichen)
- ▶ *.precision* Genauigkeit (Datentypabhängig)
- ▶ *length* Datentypgröße
- ▶ *specifier* d, f, x, s, ...

4!

Formatzeichen

vollständige syntax

`%[flags][width][.precision][length]specifier`

- ▶ *flags* verschiedene Flags, wie Vorzeichen, etc
- ▶ *width* wie breit soll die Ausgabe sein (Anzahl Zeichen)
- ▶ *.precision* Genauigkeit (Datentypabhängig)
- ▶ *length* Datentypgröße
- ▶ *specifier* d, f, x, s, ...

src/printfFormatting.c

```
1 int  einInt=42;
2 double  einDb=1.2345;
3
4 printf(" %10.4d|%-10.4d|%10.10d|  % -15.10lf\n", einInt ,
        einInt , einInt , einDb);
```



4!

Eingabe mit scanf

src/scanfDataTypesLsg.c

```
1 printf("Geben sie einen Integer ein: ");  
2 scanf("%d",&einInt);
```

4!

Eingabe mit scanf

src/scanfDataTypesLsg.c

```
1 printf("Geben sie einen Integer ein: ");  
2 scanf("%d",&einInt);
```

Signatur

int scanf (const char * format, ...)

- ▶ Parameter: Formatstring *format* enthält jetzt Formatzeichen für das, was gelesen werden soll
- ▶ beliebig viele Variablen-Adressen
- ▶ Rückgabe(success): Anzahl gelesenen Werten
- ▶ Rückgabe(error): EOF (vordefinierte Konstante)

Syntax des Formatstrings

`%[*][width][modifier]type`

- ▶ *** : gelesener Wert wird nicht in Variable gespeichert
- ▶ *width*: Maximale Zeichenanzahl, die eingelesen wird- *wichtig für Strings*
- ▶ *modifier* : Datentypgröße
- ▶ *type* : bekannt aus printf, also d, f, s, etc

Eingabe mit scanf und Textvorgabe

src/scanfUhrzeit.c

```
1 int stunden;  
2 int minuten;  
3  
4 printf("\nBitte Uhrzeit eingeben im Format hh:mm\n");  
5 int returnValue=scanf("%d:%d",&stunden, &minuten);
```

- ▶ Wenn normaler Text im Formatstring enthalten ist, wird dieser als Vergleich mit der Eingabe benutzt
 - ▷ weicht die Eingabe ab, wird abgebrochen
 - ▷ ist sie korrekt, werden alle Parameter eingelesen



4!

Files öffnen und schliessen

- ▶ öffnen mit *fopen*
 - ▶ schliessen mit *fclose*
-
- ▶ Sind nicht nur Bereiche auf der Festplatte
 - ▶ Auch Netzwerkschnittstellen o.ä.
 - ▶ Aber auch Standard Ein- und Ausgabe (Terminal)

fopen

```
FILE *fp=fopen("filesOpenClose.c", "r");  
fclose(fp);
```

fileOpenClose.c

4!

fopen

```
FILE *fp=fopen("filesOpenClose.c", "r");  
fclose(fp);
```

fileOpenClose.c

Signatur

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

- ▶ Parameter: Dateiname *filename* als String
- ▶ Dateimodus *mode* als String
- ▶ Rückgabe(success): Dateideskriptor
- ▶ Rückgabe(error): NULL (vordefinierte Konstante)

```
FILE *fp=fopen("filesOpenClose.c", "r");  
fclose(fp);
```

fileOpenClose.c

- ▶ *r* : Datei nur zum Lesen öffnen
- ▶ *w* : Datei zum (Über-)schreiben öffnen
 - ▷ existiert sie noch nicht, wird sie angelegt
 - ▷ existiert sie, wird der bisherige Inhalt überschrieben
- ▶ *a* : Datei zum Anhängen öffnen
 - ▷ existiert sie noch nicht, wird sie angelegt
 - ▷ existiert sie, werden geschriebene Daten am Ende angelegt (append)

Dateideskriptoren

- ▶ Abstraktes Object zur Kommunikation
- ▶ Bereits vom Betriebssystem implementiert (Standardstreams in Linux)
 - ▷ *stdout*: Standard Output, dorthin gibt printf Daten aus
 - ▷ *stdin* : Standard Input, von dort liest scanf Daten
 - ▷ *stderr* : Standard Error, Fehlerausgabe
- ▶ Sehr einfach umzuleiten (piping in der Konsole)

```
$ outputUhrzeit | scanfUhrzeit
```

```
Bitte Uhrzeit eingeben im Format hh:mm
```

```
13 Stunden und 52 Minuten
```

```
$
```

Piping

fclose

```
FILE *fp=fopen("filesOpenClose.c", "r");  
fclose(fp);
```

fileOpenClose.c

4!

fclose

```
FILE *fp=fopen("filesOpenClose.c", "r");  
fclose(fp);
```

fileOpenClose.c

Signatur

```
int fclose (FILE * stream);
```

- ▶ Parameter: Stream *stream*, der vorher mit `fopen` geöffnet wurde
- ▶ Rückgabe(success): 0
- ▶ Rückgabe(error): EOF (vordefinierte Konstante)



4!

Dateien Input/Output

Anstatt den Standard-io zu verwenden, benutzen wir den Stream, den wir mit `fopen()` erhalten haben, und lesen aus/schreiben in einer Datei mit `fscanf/fprintf`¹

src/filesInput.c

```
1 fscanf ( fp , "%d %127s" , &anzahl , bezeichnung );
```

src/filesOutput.c

```
1 fprintf ( fp , "Guten Tag." );
```

fscanf

```
int fscanf (FILE * stream,  
const char * format, ...);
```

fprintf

```
int fprintf (FILE * stream,  
const char * format, ...);
```

¹Wie `printf/scanf`, beachte zusätzlich *stream* Parameter

feof: ist die Datei zu Ende?

src/filesKuehlschrank.c

```
1  if (feof(fp)) {  
2      break;  
3  }
```

4!

feof: ist die Datei zu Ende?

src/filesKuehlschrank.c

```
1  if (feof(fp)) {  
2      break;  
3  }
```

Signatur

`int feof (FILE * stream)`

- ▶ Parameter: Stream *stream* aus `fopen`
- ▶ Rückgabe(eof): 1 (wahr)
- ▶ Rückgabe(nicht eof): 0 (falsch)

und jetzt, üben

