[FileIO] Ein-/Ausgabe mit Files

- File öffnen:

```
File *fopen (const char *dateiname,
             const char *zugriff);
```

```
r   Datei lesen; sie muss bereits existieren
w   Datei schreiben; ihr Inhalt wird überschrieben, wenn sie
    bereits existiert.
a   An bestehende Datei anhängen, andernfalls Datei neu erstellen.
r+  Datei lesen und schreiben; sie muss bereits existieren
w+  Datei lesen und schreiben; falls die Datei existiert, wird ihr
    Inhalt überschrieben.
a+  Aus bestehender Datei lesen oder anhängen; falls Datei noch
    nicht existiert, diese erstellen.
```

[FileIO] Mit Files arbeiten

```
int fscanf(FILE *datei, const char *format, ...);
int fgetc(FILE *datei);
int fprintf(FILE *datei, const char *format, ...);
int fputc(int c, FILE *datei);
```

scanf und printf sind Sonderfälle von fscanf und fprintf, bei denen die Standardeingabe bzw Standardausgabe benutzt wird:

```
fscanf(stdin , ...)

fprintf(stdout , ...)
```

[FileIO] Mit Files arbeiten: Ein Beispiel

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

void datei_kopieren(FILE *ziel, FILE *quelle);
void argument_extraction(FILE *quelle);

int main()
{       char infile[] = "argument.txt";
        char outfile[]= "copy.txt";
        FILE *datei_in;
        FILE *datei_out;
        datei_in = fopen(infile,"r");
        if(datei_in == NULL)
        {       printf("Datei %s nicht verfuegbar.\n",infile);
                return 1;
        }
        datei_out = fopen(outfile,"w");
        if(datei_out == NULL)
        {       printf("Datei %s nicht
verfuegbar.\n",outfile);
                return 1;
        }
        datei_kopieren(datei_out,datei_in);
        argument_extraction(datei_in);
        fclose(datei_in);
        fclose(datei_out);
        return 0;
}
```

```
void datei_kopieren(FILE *ziel, FILE *quelle)
{       int c;
        while ((c = fgetc(quelle)) != EOF)
                fputc(c, ziel);
}
```

```
void argument_extraction(FILE *quelle)
{       int c,i=0;
        char line[120];
        while ((c = fgetc(quelle)) != EOF)
        {       line[i++]=c;
                if (c=='\n')
                {       line[i]='\0';
                        i=0;
                        if(strstr(line,"Yes") != NULL   ||
                          strstr(line,"yes") != NULL    ||
                          strstr(line,"No") != NULL     ||
                          strstr(line,"no") != NULL  )
                                printf("%s",line);
                }
        }
}
```

[FileIO] Formatierte und binäre Ein-/Ausgabe

Ein- und Ausgabe, wie bisher kennengelernt, ist formatiert, d.h.

- bei der Eingabe werden Zeichen interpretiert und in eine binäre Darstellung überführt
- bei der Ausgabe wird eine interne, binäre Darstellung in eine Zeichenfolge umgewandelt.

Bei der unformatierten (=binären) Ein- und Ausgabe werden Bitfolgen nicht verändert sondern direkt weitergegeben.
Die Binäre Ein-und Ausgabe ist damit viel schneller, da keine Umrechnungen notwendig sind.

[FileIO] Formatierte und binäre Ein-/Ausgabe

Aber:

- Nicht auf jedem Ein-/Ausgabegerät ist eine binäre Ein-/Ausgabe möglich. (z.B. Tastatur, Bildschirm, Drucker)
- Binärdaten sind nicht portabel, d.h. die Interpretation einer binären Folge kann auf verschiedenen Rechnern unterschiedlich sein (je nach Darstellung und Speicherbedarf der Typen).
  Auch formatierte Daten können nicht ohne weiteres übertragen werden, verlangen aber lediglich eine Zeichentabelle.

```
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);

size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);
```

[FileIO] Ausgabeformate

Allgemeine Form der Formatbeschreiber für die Ausgabe:
%<*modus*><*laenge*><.*stellen*><*typ*>*kennung*

```
Ausgabe ganzer Zahlen:
i,d     signed int (Ausgabe dezimal)
u       unsigned int (Ausgabe dezimal)
o,x,X   unsigned int  (Ausgabe oktal, hexadezimal mit Klein-
        bzw Grossbuchstaben)

Ausgabe von Gleitkommazahlen:
f       exponentenfrei
e E     halblogarithmisch, Exponententeil beginnt mit e bzw E
        (die Darstellung ist so normiert, dass links vom Dezimalpunkt
        genau eine Ziffer steht, die nur dann Null ist, wenn der Wert
        insgesamt Null ist)
g G     wie f oder e, je nach Größe des Ausgabewertes
        (Nullen am Ende der Ziffernfolge werden unterdrückt, der
        Dezimalpunkt wird nur geschrieben, wenn ihm eine Ziffer folgt.)

Zeichen und Strings
c       der int-Wert wird in unsigned char umgewandelt geschrieben
s       Der Ausgabeparameter wird als Zeiger auf einen String betrachtet
```

[FileIO] Ausgabeformate

```
Kennungen für verschiedene Zwecke
p   der Wert wird als Zeiger betrachtet
n   es wird keine Ausgabe erzeugt, sondern die Anzahl der bislang
    übertragenen Zeichen in den nächsten Parameter der Ausgabeliste
    geschrieben. Dieser Wert muss entsprechend den Typ int* besitzen.
%   ein Prozentzeichen wird geschrieben (dem Formatbeschreiber wird
    kein Ausgabewert zugeordnet)
```

```
Längeneintrag:
legt die Mindestanzahl auszugebender Stellen fest
>0 rechtsbündige Darstellung
<0 linksbündige Darstellung
* statt einer Zahl: Der nächste Wert aus der Ausgabeliste wird als Länge
                    genommen

Stellen:
- Bei Dezimalzahlen: Auszugebende Stellen, evtl. führende Nullen
- Bei Gleitkommazahlen: Stellen hinter dem Komma
- Bei Strings: Höchstens angebene Zahl an Zeichen des Strings werden
  dargestellt

Modus:
+   Vorzeichen wird (auch bei Plus) mit dargestellt
```

[FileIO] Ausgabeformatierung

```
int i =13,j=4;
float f=127.1345;

/* Längeneintrag */

printf("__%d__\n", i);          /* __13__ */
printf("__%6d__\n", i);         /* __    13__ */
printf("__%-6d__\n", i);        /* __13    __ */
printf("__%*d__\n", j , i);     /* __  13 __ */

/* Stelleneintrag */

printf("__%6.4d__\n", i);       /* __  0013__ */
printf("__%6.2f__\n", f);       /* __127.13__ */

/* Modus */
printf("__%+6d__\n", i);        /* __   +13__ */
```

[FileIO] Eingabeformate

Allgemeine Form der Formatbeschreiber für die Eingabe:
%<*><*laenge*><*typ*>*kennung*

```
Ausgabe ganzer Zahlen:
i       ganze Zahl mit oder ohne Vorzeichen
        bei hexadezimaler Darstellung mit einleitendem 0x oder 0X,
        bei oktaler Darstellung mit Einleitendem O
d       ganze Zahl mit oder ohne Vorzeichen in dezimaler Darstellung
u       ganze Zahl ohne Vorzeichen
o       ganze Zahl mit oder ohne Vorzeichen in oktaler Darstellung
x,X     ganze Zahl mit oder ohne Vorzeichen in hexadezimaler Darstellung
        (aber ohne Einleitendes 0x bzw. 0X)

Ausgabe von Gleitkommazahlen:
f,e,E,g,G
        Eingabe eines gültigen Gleitkommawertes mit oder ohne Vorzeichen,
        mit oder ohne Exponententeil, mit oder ohne Dezimalpunkt

Zeichen und Strings
s       beliebige Zeichenfolge, beendet durch ein whitespace
        An die Zeichenfolge wird automatisch ein '\0' angehängt
c       einzelnes Zeichen, welches auch ein whitespace sein kann.
[       wie s, jedoch kann explizit angegeben werden, bei welchen Zeichen
        die Übertragung stoppen soll.
```

[FileIO] Eingabeformate

Beginnt ein Formatbeschreiber mit einem Stern, so wird die entsprechende Zeichenfolge zwar interpretiert, nicht aber in eine Eingabevariable abgelegt.

Im Zusammenwirken mit [] ist die Auswertung regulärer Ausdrücke möglich.

```
int i , anzahl;
float f;
double d;

scanf("%*[a-zA-Z]%d",&i);
printf("i      : %d\n",i);

anzahl = scanf("%5d %c %f %lf",&i, &c, &f, &d);
printf("Anzahl: %d\n",anzahl);
printf("i      : %d\n",i);
printf("c      : %c\n",c);
printf("f      : %f\n",f);
printf("d      : %lf\n",d);
```

[FileIO] Vorausschau beim Lesen: *ungetc*

Um ein Zeichen, welches aus dem Eingabepuffer gelesen wurde,wieder zurückzuschreiben, kann man die Funktion ungetc verwenden:

```
int ungetc(int c, FILE *datei);
```

[FileIO] Ein Beispiel

Einlesen der Studentendaten aus einem File (unter Beibehaltung der vorherigen Funktionalität)

```
/* stud_struct.h  */

#ifndef STUD_STRUCT_H
#define STUD_STRUCT_H

typedef struct adresse_s
{
        char wohnort[30];
        int plz;
        char strasse[30];
        int hausnummer;
} adresse_t;

typedef struct student_s
{
        char vorname[30];
        char nachname[30];
        int matrikelnummer;
        adresse_t adresse;
} student_t ;


extern FILE *eingabe;

void satz_lesen(student_t *const s);
void satz_schreiben(student_t *const s);
void adresse_lesen(adresse_t *const a);

#endif
```

```
#include "stud_struct.h"

FILE *eingabe;

void satz_lesen(student_t *const s)
{       if (eingabe == NULL)
                eingabe = stdin;
        printf("Nachname: ");
        fscanf(eingabe,"%s", (*s).nachname);
        printf("Vorname: ");
        fscanf(eingabe,"%s", s->vorname);

        adresse_lesen(&s->adresse);
}
void adresse_lesen(adresse_t *const a)
{       printf("Ort: ");
        fscanf(eingabe,"%s", a->wohnort);
        printf("Plz: ");
        fscanf(eingabe,"%d", &a->plz);
        printf("Strasse: ");
        fscanf(eingabe,"%s", a->strasse);
        printf("Hausnummer: ");
        fscanf(eingabe,"%d", &a->hausnummer);
}
void satz_schreiben(student_t *s)
{       printf("Nachname:  %s\n",s->nachname);
        printf("Vorname:   %s\n",s->vorname);

        printf("Ortname:   %s\n",s->adresse.wohnort);
        printf("PLZ:       %d\n",s->adresse.plz);
        printf("Strasse:   %s\n",s->adresse.strasse);
        printf("Hausnummer:%d\n",s->adresse.hausnummer);
}
```

[FileIO] Ein Beispiel

```
#include <stdio.h>
#include "stud_mem.h"
int main()
{       char test,  input[] = "studenten.txt";
        student_t *stud_p;
        int anzahl_stud,i=0,j=0;
        printf("Weviele Saetze werden eingegeben: ");
        scanf("%d",&anzahl_stud);
        stud_p = getstudmem(anzahl_stud);
        if (stud_p == NULL)   return 0;
        while (getchar() != '\n');
        printf("Lesen aus Datei (D)oder stdin (S)? ");
        test = getchar();
        if (test == 'D' || test == 'd')
        {       eingabe = fopen(input,"r");
                if (eingabe == NULL)
                {       printf("Einlesen nicht möglich! Lese von stdin.");
                        eingabe = stdin;
                }
        }
        else    eingabe = stdin;
        while (getchar() != '\n');
        while ( i < anzahl_stud )
        {
                printf("Noch einen Datensatz eingeben (J/N)");
                test= fgetc(eingabe);
                if (test == 'J' || test == 'j')
                        satz_lesen(&stud_p[i++]);
                if (test == 'N' || test == 'n')
                        break;
                while (fgetc(eingabe) != '\n');
        }
        printf("Datensaetze eingegeben: %d\n",i);
        while( j < i)
                satz_schreiben(&stud_p[j++]);
        freestudmem(stud_p);
        return 0;
}
```

```
GCCFLAGS = -ansi -pedantic -Wall

struct3: struct3.o stud_struct.o
        gcc -o struct3 struct3.o stud_struct.o

struct3.o : struct3.c stud_struct.h
        gcc $(GCCFLAGS) -c struct3.c

struct4: struct4.o stud_struct.o stud_mem.o
        gcc -o struct4 struct4.o stud_struct.o stud_mem.o

struct5: struct5.o stud_struct.o stud_mem.o
        gcc -o struct5 struct5.o stud_struct.o stud_mem.o

struct4.o : struct4.c stud_struct.h stud_mem.h
        gcc $(GCCFLAGS) -c struct4.c

struct5.o : struct5.c stud_struct.h stud_mem.h
        gcc $(GCCFLAGS) -c struct5.c

stud_struct.o : stud_struct.c stud_struct.h
        gcc $(GCCFLAGS) -c stud_struct.c

stud_mem.o : stud_mem.c stud_mem.h
        gcc $(GCCFLAGS) -c stud_mem.c

clean:
        rm -f *~ struct*o struct3 struct4 struct5
```

[FileIO] Noch ein Beispiel

Sortieren der Studentendaten mit der einfach verketteten Liste (unter geringfügiger Modifizierung der vorhandenen Module)

```
void listeZeigen(struct knoten *anfang)
{       satz_schreiben(anfang->daten);
        if (anfang->nachfolger !=NULL)
                listeZeigen(anfang->nachfolger);
}
struct knoten* neueListe(struct student_s *daten)
{       struct knoten *anfang = erzeugeKnoten(daten);
        anfang->nachfolger = NULL;
        return anfang;
}
```

```
#include "stud_evl.h"
#include <string.h>
struct knoten* elementEinfuegen(struct knoten *anfang, struct student_s *daten)
{       if (strcmp(daten->nachname, anfang->daten->nachname) < 0)
        {       struct knoten *knotenneu = erzeugeKnoten(daten);
                knotenneu->nachfolger = anfang;
                return knotenneu;
        }
        else if (anfang->nachfolger != NULL)
                anfang->nachfolger=elementEinfuegen(anfang->nachfolger,daten);
        else
        {       struct knoten *knotenneu = erzeugeKnoten(daten);
                anfang->nachfolger = knotenneu;
                knotenneu->nachfolger = NULL;
        }
        return anfang;
}
struct knoten* erzeugeKnoten(struct student_s *daten)
{       struct knoten *a;
        a = malloc(sizeof(struct knoten));
        if (a == NULL)
                printf("Speicherallokation fehlgeschlagen.");
        a->daten=daten;
        return a;
}
void loescheListe(struct knoten *anfang)
{       if (anfang->nachfolger != NULL)
                loescheListe(anfang->nachfolger);
        printf("Loesche nun %s\n",anfang->daten->nachname);
        free(anfang->daten);
        free(anfang);
}
```

[FileIO] Noch ein Beispiel (Forts.)

```
/* stud_evl.h */

#ifndef STUD_EVL_H
#define STUD_EVL_H

#include <stdio.h>
#include <stdlib.h>
#include "stud_struct.h"

struct knoten
{
        struct student_s* daten;
        struct knoten *nachfolger;
};

struct knoten*      erzeugeKnoten       (struct student_s *daten);
struct knoten*      neueListe           (struct student_s *daten);
void                loescheListe        (struct knoten *anfang);
int                 listeLeer           (struct knoten *anfang);
struct knoten*      elementEinfuegen    (struct knoten *anfang, struct student_s *daten);
void                listeZeigen         (struct knoten *anfang);

void                elementLoeschen     (struct knoten *anfang, struct student_s *daten);
void                elementSuchen       (struct knoten *anfang, struct student_s *daten);

#endif
```

[FileIO] Noch ein Beispiel (Forts.)

```
#include <stdio.h>
#include "stud_evl.h"
#include "stud_struct.h"
int main()
{       char test; input[] = "studenten.txt";
        student_t *stud_p;
        struct student_s dummy = {"  dummy"};
        struct knoten *anfang;
        int i=0;
        printf("Lesen aus Datei (D)oder stdin (S)? ");
        test = getchar();
        if (test == 'D' || test == 'd')
        {       eingabe = fopen(input,"r");
                if (eingabe == NULL)
                {       printf("Einlesen nicht möglich! Lese von stdin.");
                        eingabe = stdin;
                }
        }
        else    eingabe = stdin;
        while (getchar() != '\n');
        anfang = neueListe(&dummy);
        while (1)
        {       printf("Noch einen Datensatz eingeben (J/N)");
                test= fgetc(eingabe);
                if (test == 'J' || test == 'j')
                {       stud_p = (student_t*) malloc(sizeof(student_t));
                        if (stud_p == NULL)   printf("Speicherallokation fehlgeschlagen.");
                        satz_lesen(stud_p);
                        elementEinfuegen(anfang,stud_p);
                }
                if (test == 'N' || test == 'n')           break;
                while (fgetc(eingabe) != '\n');
        }
        printf("Datensaetze eingegeben: %d\n",i);
        listeZeigen(anfang);
        loescheListe(anfang->nachfolger);
        return 0;
}
```

[FileIO] Noch ein Beispiel (Forts.)

```
GCCFLAGS = -ansi -pedantic -Wall

struct3: struct3.o stud_struct.o
	gcc -o struct3 struct3.o stud_struct.o

struct3.o : struct3.c stud_struct.h
	gcc $(GCCFLAGS) -c struct3.c

struct4: struct4.o stud_struct.o stud_mem.o
	gcc -o struct4 struct4.o stud_struct.o stud_mem.o

struct5: struct5.o stud_struct.o stud_mem.o
	gcc -o struct5 struct5.o stud_struct.o stud_mem.o

struct6: struct6.o stud_struct.o stud_evl.o
	gcc -o struct6 struct6.o stud_struct.o stud_evl.o

struct4.o : struct4.c stud_struct.h stud_mem.h
	gcc $(GCCFLAGS) -c struct4.c

struct6.o : struct6.c stud_struct.h
	gcc $(GCCFLAGS) -c struct6.c

stud_struct.o : stud_struct.c stud_struct.h
	gcc $(GCCFLAGS) -c stud_struct.c

stud_mem.o : stud_mem.c stud_mem.h
	gcc $(GCCFLAGS) -c stud_mem.c

stud_evl.o : stud_evl.c stud_evl.h
	gcc $(GCCFLAGS) -c stud_evl.c

clean:
	rm -f *~ struct*o stud*o struct3 struct4
```