9 I/0 mit Dateien

Wenn ein ausführbares Programm *prog* vorliegt, das I/O vom Bildschirm verlangt, so kann man mit *prog* < *ein* > *aus* die Eingabe von einer Datei *ein* und die Ausgabe in eine Datei *aus* erreichen. Benötigt man mehrere Dateien zur Ein- bzw. Ausgabe, so muß man diese Dateien vereinbaren und entsprechende Lese- und Schreibbefehle verwenden.

FILE-Deklaration:

FILE **pointer*; vereinbart einen Zeiger, der auf Informationen zu einer Datei verweist.

In der header-Datei `stdio.h` werden unter anderem auch folgende Funktionen definiert.

char *gets(char *s) liest eine Zeile von stdin bis newline oder EOF, prüft Puffergröße nicht, was zu Problemen führen kann, Zeilenenden stehen nicht im Puffer.

char *fgets(char *s, int n, FILE *name) Einlesen von der Datei *name* in den Puffer, auf den *s* zeigt, das Lesen endet, wenn EOF erreicht wurde oder *n-1* Zeichen eingelesen wurden. An das Eingelesene wir \0 als Ende-Kennung angefügt. Die Tastatur heißt stdin, Zeilenenden stehen im Puffer. Im Fehlerfall und beim Erreichen von *EOF* wird *NULL* zurückgegegen.

```
char buffer[200];
char *zeile;
printf("Eingabe des Polynoms vom Grad <= 9\n");
fgets(buffer, sizeof(buffer), stdin);
zeile = &buffer[0];
```

int fputs(char *s, FILE *name) Die Zeichenkette, auf die der Pointer *s* zeigt, wird in die durch *name* definierte Datei geschrieben. Das Ende-Zeichen wird nicht ausgegeben. Rückgabewert ist positiv oder *EOF*, wenn ein Fehler aufgetreten ist.

int fopen(FILE *name, mode) die in *name* vereinbarte Datei wird im Modus *mode* geöffnet: “r“ (nur Lesen), “w“ (Schreiben, Überschreiben), “a“ (am Ende Anfügen), bzw. “r+“ (Schreiben und Lesen, Datei muß

existieren), “w+“ (Schreiben, Überschreiben und Lesen), “a+“ (Lesen und Anfügen).

int fclose(FILE *name) die in *name* vereinbarte Datei wird geschlossen.

int fprintf(FILE *name, const char *fmt, *vars*) schreiben in Datei *name* mit Formatangabe zu *vars*. Rückgabewert ist im Fehlerfall negativ, sonst die Zahl der ausgegebenen Zeichen.

int fscanf(FILE *name, const char *fmt,*vars*) lesen von Datei *name* mit Formatangabe zu *vars*. Rückgabewert ist *0*, wenn kein Zeichen eingelesen werden kann, sonst die Zahl der fehlerfrei eingegebenen Zeichen. Bei Eingabefehlern vor einer Konversion wird *EOF* zurückgegeben.

Es gibt Befehle zum Positionieren innerhalb von Dateien (*fseek, ftell* u.a.m.). Damit kann der sequentielle Dateizugriff umgangen werden.

Listing 43: Ausgabe-Datei (ausgabe.c)

```
#include <stdio.h>
int main()
{
        char z[9] = "Hi Jane!";
        FILE *pausgabe;
        pausgabe = fopen("aus", "w");
        fprintf(pausgabe, "%s\n", z);
        fclose(pausgabe);
        return 0;
}
```

Listing 44: Einlesen long double (readldbl.c)

```
#include <stdio.h>
#define STDIN_EOF '\n'
long double get_ldbl(void)
{
long double a;
int w = 4711;
while (1) {
printf("\nEingabe einer long double  > 0: ");
        if (!(w = scanf("%Lf", &a)) || a <= 0) {
                while (STDIN_EOF != getchar());
/* <CR> (oder mehr) noch im Eingabepuffer */
                printf
                    ("\nUnzulaessige Eingabe."
                     " Bitte wiederholen. %d", w);
                } else
                        break;
        }
return a;
}

int main(void)
{
        long double z, get_ldbl(void);
        z = get_ldbl();
        printf("%Lf\n", z);
        return 0;
}
```

Listing 45: Ausgeben long double (writeldbl.c)

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

void ausgabe(long double x, int s)
{
/* Ausgabe von x mit s Nachkommastellen,
 * Format: \n%.sLf\n */
        char format[15] = "\n %.";
        char nm[5] = "";
        sprintf(nm, "%i", s);
/* schreibt nach 'nm' */
        strcat(format, nm);
/* haengt 'nm' an 'format' an */
        strcat(format, "Lf\n");
        printf(format, x);
        return;
}

int main(void)
{
        long double x = 12.5657L;
        ausgabe(x, 9);
        return 0;
}
```

Argumente der Kommandozeile

Argumente der Kommandozeile:

int main (int argc, char *argv[]); Argumente, die hinter dem Programmaufruf angegeben werden, stehen so dem Hauptprogramm zur Verfügung. `argc` enthält die um 1 verminderte Zahl der Argumente, wobei `argv[0]` der Name des C-Programms ist, und mit
`argv[1], argv[2], ..., argv[argc -1]`
die Argumente der Kommandozeile sind.

Beispiele:

Listing 46: Kommandlinemodus (commandline1.c)

```
#include <stdio.h>

int main(int argc, char *argv[])
{
        int i;
        for (i = 0; i <= argc - 1; i++)
                printf("Arg %i: %s\n", i, argv[i]);
        return 0;
}
```

Listing 47: Kommandlinemodus (commandline2.c)

```
#include <stdio.h>

int main(int argc, char *argv[]){
while (--argc > 0)
        printf("%s%s", *++argv,
                        (argc > 1) ? " " : "");
printf("\n");
return 0;
}
```

9.1 Strukturen und eigene Datentypen

Struktur:

struct *structname* { *definition* }; legt eine Struktur mit Namen *structname* fest. Dabei werden in *definition* die Datentypen der Komponenten erklärt, aus denen die Struktur besteht.

Beispiel:

```
struct complex {
        double re;
        double im;
};
struct complex var;
```

Selektion der Komponenten:
Auf die einzelnen Komponenten eines durch `struct` festgelegten Datentyps kann man mit . zugreifen, im obigen Beispiel liefert somit *var.re* den Realteil, *var.im* den Imaginärteil. Wenn *pt* ein Zeiger auf *var* ist, erfolgt der Zugriff auf den Realteil durch *pt*->*re*.

Listing 48: Rationale Arithmetik (rational.c)

```
#include <stdio.h>

struct frac {
        int z;
        int n; };

int ggt(int z1, int z2) {
        int rest;
        while ((rest = z1 % z2)) {
                z1 = z2;
                z2 = rest;
        }
        return z2; }

struct frac add(struct frac x, struct frac y) {
        int zwischen, p1, p2;
        zwischen = x.z * y.n + y.z * x.n;
        p1 = ggt(zwischen, x.n);
        if (p1 != 1) {
                zwischen /= p1;
                x.n /= p1; }
        p2 = ggt(zwischen, y.n);
        if (p2 != 1) {
                zwischen /= p2;
                y.n /= p2; }
        x.z = zwischen;
        x.n = y.n * x.n;
        return x; }
```

```
int main() {
        struct frac x, y, res;
        struct frac add(struct frac, struct frac);
        void printfrac(struct frac);
        struct frac readfrac(void);

        x = readfrac();
        y = readfrac();
        res = add(x, y);
        printfrac(res);
        return 0; }

struct frac readfrac(void) {
        struct frac x;
        printf("Eingabe Zaehler Nenner: ");
        (void) scanf("%d %d", &x.z, &x.n);
        return x; }

void printfrac(struct frac res) {
        if (res.n == 1)
                printf("%d\n", res.z);
        else
                printf("%d/%d\n", res.z, res.n); }
```

Alias Namen:

typedef type *aliastype*; vereinbart *aliastype* als Alias für den Datentyp `type`.

Neue Datentypen:

typedef struct *structname* { *definition* }
typename; vereinbart `struct` *structname* und vergibt gleichzeitig als Alias *typename*. Eine Variable des neuen Datentyp kann mit
`struct` *structname var;*
oder mit
typename var; vereinbart werden.

Listing 49: complex.h (complex.h)

```
typedef struct {
        double re;
        double im;
} complex;

#define cset(z,r,i)      (z.re=r, z.im=i)
#define cput(z) (printf("(%f,%f)",z.re,z.im))
#define eps      (3 * DBL_EPSILON)
```

Listing 50: Komplexe Arithmetik (komplex.c)

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <float.h>
#include "complex.h"

complex cadd(complex x, complex y) {
        complex z;
        z.re = x.re + y.re;
        z.im = x.im + y.im;
        return (z); }

complex csub(complex x, complex y) {
        complex z;
        z.re = x.re - y.re;
        z.im = x.im - y.im;
        return (z); }

complex cmult(complex x, complex y) {
        complex z;
        z.re = x.re * y.re - x.im * y.im;
        z.im = x.re * y.im + x.im * y.re;
        return (z); }

complex cdiv(complex x, complex y) {
        complex z;
        double h;
        h = y.re * y.re + y.im * y.im;
        if (fabs(h) < eps) {
                printf("divion by zero\n");
                exit(EXIT_FAILURE);
        }
        z.re = (x.re * y.re + x.im * y.im) / h;
        z.im = (x.im * y.re - x.re * y.im) / h;
        return (z); }
```

```
complex conj(complex x) {
        complex z;
        z.re = x.re;
        z.im = -x.im;
        return z; }

double cabs(complex x) {
        return sqrt(x.re * x.re + x.im * x.im); }

int main() {
        double RE, IM;
        complex x, y, z;
        printf("Eingabe Re Im\n");
        (void) scanf("%lf%lf", &RE, &IM);
        cset(x, RE, IM);
        printf("Eingabe Re Im\n");
        (void) scanf("%lf%lf", &RE, &IM);
        cset(y, RE, IM);
        z = csub(x, y);
        printf("x - y\n");
        cput(z);
        z = cadd(x, y);
        printf("\nx + y\n");
        cput(z);
        printf("\n");
        printf("\nbetrag(x)\n");
        printf("%g\n", cabs(x));
        return 0; }
```

Die folgenden Funktionen stammen aus `stdlib.h`.

char * strchr(*zkette*,*zeichen*) liefert Zeiger auf das erste Zeichen *zeichen* in *zkette*, falls vorhanden sonst `NULL`.

int atoi(const char * *zkette*) liest *zkette* als `int`.

double strtod(const char* zkette,char new)** liest den Anfang von *zkette* als `double`, Rest der Zeichenkette bei *new, falls das nicht `NULL` gesetzt wurde.

double strtol(const char* zkette,char new, int *base*)** liest den Anfang von *zkette* als `long int` zur Basis *base*, Rest der Zeichenkette bei *new, falls das nicht `NULL` gesetzt wurde.

char * strtok(char * *zkette*, char* *delim*) liefert Zeiger auf den ersten Teil von *zkette*, der durch ein Zeichen aus *delim* begrenzt wird. Bei mehrfachem Anwenden *delim* `NULL` setzen.

Listing 51: Parser (parser.c)

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

int main(void) {
void findout(char *, double *);
void transform(char *, char *);
char buffer[200];
char *zeile, *teil, *newzeile;
double coeff[10] =
{ 0., 0., 0., 0., 0., 0., 0., 0., 0., 0. };
int i;
printf("Eingabe des Polynoms vom Grad <= 9\n");
fgets(buffer, sizeof(buffer), stdin);
zeile = &buffer[0];
newzeile = (char *)
calloc( (size_t) (strlen(zeile)+10), sizeof(char));
if (newzeile == NULL)
        exit(0);
transform(zeile,newzeile);
free(zeile);
teil = strtok(newzeile, ":");
while (teil != NULL) {
        findout(teil,coeff);
        teil = strtok(NULL, ":");
}
for ( i = 0; i<=9; i++)
        printf("%f ", coeff[i]);
printf("\n");
free(newzeile);
return 0;
}
```

```
void findout(char *teil, double *coeff) {
char zeichen;
double wert;
int i = 0, expo;
if (strchr(teil,'x') == NULL) { /* konstante */
        coeff[0] = strtod(teil,NULL);
} else if (strchr(teil,'^') == NULL) {/* grad 1 */
        wert = strtod(teil, NULL);
        if ( wert == 0.0 ) {
                        if  (teil[0] == '-' )
                                wert = -1.0;
                        else
                                wert = 1.0; }
        coeff[1] += wert;
} else {        /* grad > 1 */
        zeichen = teil[0];
        while (zeichen != '\0') {
                if (zeichen == 'x') {
                        wert = strtod(teil,NULL);
                        if ( wert == 0.0 ) {
                                if  (teil[0] == '-' )
                                        wert = -1.0;
                                else
                                        wert = 1.0; }
                        teil = &teil[i + 2];
                        expo = atoi(teil);
                        coeff[expo] += wert;
                        break; }
                i++;
                zeichen = teil[i]; }
} }
```

```
void transform(char * zeile, char* newzeile) {
char zeichen;
int i=0, j = 0;
zeichen = zeile[0];
while (zeichen != '\0') {
        if ( (zeichen == '+' ) || (zeichen == '-')) {
                newzeile[j] = ':'; j++;
                newzeile[j] = zeichen; j++;
        } else if ( zeichen != ' ' ) {
                newzeile[j] = zeichen;
                j++;
        }
        i++; zeichen = zeile[i]; }
}
```