7 Dateien

Um Daten dauerhaft zu sichern, müssen sie auf den Sekundärspeicher (die Festplatte) abgespeichert werden. Beim Umgang mit Peripheriegeräten unter Windows und UNIX ist zu beachten

- Die Verbindung zu Peripheriegeräten (und somit auch das Lesen und Schreiben von Dateien vom Sekundärspeicher) erfogt über sogenannte Deskriptoren, die die Verbindung zu den Geräte-Treibern schaffen, kapseln (kein direkter Hardware-Zugriff).
- ImGegensatz zu anderen Betriebssystemen unterstützen Windows und UNIX nur einen einzigen Dateityp: Sequentielle Datei aus einer (nicht strukturierten) Folge von Bytes (streamfile).
- Das Schreiben und Lesen von Daten aus Dateien erfolgt sequentiell. Die Steuerung erfolgt über einen intern geführten Dateizeiger, der nach jedem Lese-/Schreibvorgang auf das nächste zu bearbeitende Zeichen zeigt.
- Im allgemeinen werden Schreibvorgänge am Ende der Datei ausgeführt. Durch spezielle Befehle kann der Dateizeiger aber auch auf eine andere Position gesetzt werden. Hierbei muß die Software (also nicht das Betriebssystem) die Verwaltung des Zeigers übernehmen.
- Wir unterschieden zwei Typen von *streamfiles*:
  - Textdateien
    - Folge von (ASCII-) Zeichen
    - Gut lesbar, kann mit jedem Texteditor editiert werden.
    - Wird häufig von Konfigurationsdateien (unter Windows ini-Dateien) verwendet.
    - Positionierungen innerhalb der Datei eher ungewöhnlich, d.h. die Daten werden von vorne nach hinten gelesen
  - Binärdatei
    - Folge von Bytes
    - Interpretation nur durch Programme, die das verwendete Format (Folge von Datentypen) kennen, möglich.
    - In der Regel kompakter als Textdateien
    - Durch die Kenntnis von Datentyplängen auch Positionierungen innerhalb der Datei möglich.

Zum Lesen und Schreiben von Dateien stehen in C verschiedene Befehle zur Verfügung. Der grundsätzliche Zugriff auf Dateien erfolgt in drei Schritten

1. Öffnen der Datei: Befehl `fopen`
2. Zugriff auf die Daten (Lesen/Schreiben)
3. Schließen der Datei: Befehl `fclose`

7.1 Öffnen und Schließen einer Datei

Durch das Öffnen einer Datei wird die Datei weder gelesen noch beschrieben, sondern das Betriebssystem liefert einen Datei-Deskriptor zurück. Mit diesem kann auf die Daten zugegriffen werden.

In C ist der Datei-Deskriptor ein Zeiger (FILE *) auf eine Datenstruktur (struct), die in `<stdio.h>` definiert ist. Komponenten dieser Datenstruktur sind u.a.

- Ein Datenpuffer (in einem Lesevorgang wird auf Effizienzgründen stets ein ganzer Datencluster gelesen)
- Ein Dateizeiger, der auf die aktuell zu lesende (oder schreibende) Position in der Datei zeigt.

Genaues Wissen über den Aufbau dieser Datenstruktur ist nicht erforderlich (da der eigentliche Datenzugriff nicht direkt über die Datenstruktur erfolgt). Der Prototyp des Befehls `fopen` zum Öffnen einer Datei in C lautet

```
FILE *fopen(char *cpDateiname, char *cpModus)
```

- Rückgabewert ist - wie oben bereits beschrieben - ein Datei-Deskriptor vom Typ FILE *. Falls das Öffnen fehlschlägt, wird NULL zurückgeleiefert.
- Der Parameter "cpDateiname" ist eine Zeichenkette, der den Dateinamen mit Pfad (absolut oder relativ) angibt, z.B
  - "messReihe.txt"
  - "c:/temp/uebung/loesung.cpp"

  - "c:\\temp\\uebung\\loesung.cpp"
  - "daten/messReihe.txt"

- Der Parameter "cpModus" ist ein String und gibt den Zugriffsmodus für die Datei an:
  - "r" (read): Nur Lesen (Datei muß existieren)
  - "w" (write): Nur Schreiben, ggf. Löschen einer berits bestehenden Datei gleichen Namens
  - "a" (append): Anfügen am Dateiende (nur Schreiben)
  - "r+": Schreiben und Lesen (Datei muß existieren)
  - "w+": Schreiben und Lesen (Existierende Datei wird zuerst gelöscht
  - "a+": Anfügen am Dateiende Schreiben und Lesen)

- Obige Liste des Parameters "cpModus" gilt für das Lesen und Schreiben von Textdateien. Sollen Binärdateien gelesen oder geschreiben werden, so ist dem Modus ein "b" hinzuzufügen: "rb", "wb", "ab", "r+b", "w+b", "a+b"

Nach Beendigung der Lese- bzw. Schreibzugriffe auf die Datei (s. nächster Abschnitt) muß sie geschlossen werden. Erst das stellt sicher, daß tatsächlich alle Daten in der Datei landen (d.h. der Dateipuffer wird geschrieben)! Der Prototyp des Befehls `fclose` zum Schließen einer Datei in C lautet

```
void fclose(FILE *filePointer)
```

7.2 Dateioperationen

Man muß eigentlich nichts neues lernen: fscanf liest Daten ein (wie bei der Tastatur/Standardeingabe) und fprintf gibt Daten aus (wie beim Bildschirm/Standardausgabe). Es gibt aber noch wesentlich mehr Befehle um komfortabel mit Dateien zu arbeiten. Nachfolgend sind die wichtigsten aufgelistet:

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

   fprintf verhält sich genauso wie printf mit Ausnahme des ersten formalen Parameters, der die Datei angibt, in die zu schreiben ist. Rückgabewert ist die Anzahl der geschriebenen Bytes oder im Fehlerfall (z.B. wegen fehlender Schreibrechte) eine negative Zahl.

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

   fscanf verhält sich genauso wie scanf mit Ausnahme des ersten formalen Parameters, der die Datei angibt, aus der zu lesen ist. Rückgabewert ist die Anzahl der gelesenen Objekte oder die in stdio.h definierte Konstante EOF, falls v o r der ersten Leseoperation das Dateiende erreicht wurde.

3. `int fputs(const char *s, FILE *stream)`

   fputs schreibt die Zeichenkette **s** in **stream**. Die Funktion liefert einen nichtnegativen Wert oder die Konstante **EOF** bei einem Fehler.

   ```
   if (fputs(s, fp) == EOF) printf("Fehler beim Schreiben\n");
   ```

4. **`char *fgets(char *s, int n, FILE *stream)`**

   fgets liest höchstens die nächsten **n-1** Zeichen in **s** ein und hört vorher auf, wenn Zeilentrenner gefunden wird. Der Zeilentrenner wird im Vektor abgelegt. Der Vektor wird mit **'\0'** abgeschlossen. **fgets** liefert **s** oder **NULL** bei Dateiende oder bei einem Fehler.

   ```
   if (fgets(s, 120, fp)!=NULL) printf("Eingelesen wurde: %s", s)
   ```

5. `int fputc(int c, FILE *stream)`

   fputc schreibt das Zeichen **c** (umgewandelt in **unsigned char**) in **stream**. Die Funktion liefert das ausgegebene Zeichen oder **EOF** bei Fehler.

   ```
   fputc('a', fp);
   ```

6. `int fgetc(FILE *stream)`

   fgetc liefert das nächste Zeichen aus **stream** als **unsigned char** (umgewandelt in **int**) oder **EOF** bei Dateiende oder bei einem Fehler.

7. `int feof(FILE *stream)`

feof liefert einen von Null verschiedenen Wert, wenn das Dateiende von **stream** ereicht ist.

```
#include <stdio.h>

int main(void) {
   FILE *stream;

    /* Öffnen einer Datei zum Lesen */
   stream = fopen("dummy.txt", "r");

    /* Liest ein Zeichen aus einer Datei */
   fgetc(stream);

    /* Prüft auf Dateiende (EOF) */
   if (feof(stream))
      printf("We have reached end-of-file\n");

    /* Schließt die Datei */
   fclose(stream);
   return 0;
}
```

7.3 Ein Beispiel

Ein Krankenhaus speichert die Daten der Neugeborenen in einer Datei "baby.dat" der Gestalt

```
Leon Daniel
Frauke Schneider
3500 55 10 12 2002
Melanie
Konstanze Taylor
3700 58 9 12 2002
Julia
Stephanie Behr
3000 48 11 12 2002
```

Zu erstellen ist ein Programm, das die Daten einliest, auf dem Bildschirm ausgibt und anschließend in der Datei "babyKopie.dat" schreibt.

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

typedef struct{
    short sTag;
    short sMonat;
    long lJahr;
} TDatum;

typedef struct{
    char cFeld120Name[120];
    char cFeld120Mutter[120];
    TDatum sctGebDat;
    short sGewicht;
    short sGroesse;
} TBaby;

void babyDatenAusgeben(TBaby sctBaby);
void lineFeedBeseitigen(char *s);

int main(void){
    FILE *FBaby;
    TBaby sctBabyFeld[100];
    int i=0,j;
    char ch;

    // Datei zum Lesen oeffnen
    if (FBaby=fopen("baby.dat", "r")){
        printf("Eintraege in der Datei:\n");
        printf("----------------------\n");
        while (fgets(sctBabyFeld[i].cFeld120Name,120,FBaby)!=NULL){
             fgets(sctBabyFeld[i].cFeld120Mutter,120,FBaby);
```

```
            lineFeedBeseitigen(sctBabyFeld[i].cFeld120Name);
            lineFeedBeseitigen(sctBabyFeld[i].cFeld120Mutter);
            fscanf(FBaby,"%d", &sctBabyFeld[i].sGewicht);
            fscanf(FBaby,"%d", &sctBabyFeld[i].sGroesse);
            fscanf(FBaby,"%d%d%d", &sctBabyFeld[i].sctGebDat.sTag,
                &sctBabyFeld[i].sctGebDat.sMonat, &sctBabyFeld[i].sctGebDat.lJahr);
            // Eingelesenen Datensatz ausgeben (Bildschirm)
            babyDatenAusgeben(sctBabyFeld[i]);
            // Zeilenumbruch zwischen den Datensätzen berücksichtigen!
            fscanf(FBaby,"%c", &ch);
            i++;
        }
        fclose(FBaby);
        // Daten auf Festplatte schreiben
        if (FBaby=fopen("babyKopie.dat", "w")){
            for (j=0; j<i; j++){
                fprintf(FBaby, "%s\n", sctBabyFeld[j].cFeld120Name);
                fprintf(FBaby, "%s\n", sctBabyFeld[j].cFeld120Mutter);
                fprintf(FBaby, "%d %d ", sctBabyFeld[j].sGewicht,
sctBabyFeld[j].sGroesse);
                fprintf(FBaby,"%d %d %d", sctBabyFeld[j].sctGebDat.sTag,
                   sctBabyFeld[j].sctGebDat.sMonat, sctBabyFeld[j].sctGebDat.lJahr);
                // Zeilenwechsel zwischen den Datensätzen
                if (j+1<i) fprintf(FBaby,"\n");
            }
            fclose(FBaby);

        }
        else
            printf("Fehler, kann Babys nicht kopieren :-)");
    }
    else
        printf("Fehler beim Öffnen der Datei, keine Babys vorhanden :-)");


    getchar();
    return 0;
}

void babyDatenAusgeben(TBaby sctBaby){
    printf("Vorname: %s\n", sctBaby.cFeld120Name);
    printf("Mutter: %s\n", sctBaby.cFeld120Mutter);
    printf("Gewicht: %dg\n", sctBaby.sGewicht);
    printf("Groesse: %dcm\n", sctBaby.sGroesse);
    printf("GebDat.: %d.%d.%d\n", sctBaby.sctGebDat.sTag,
        sctBaby.sctGebDat.sMonat, sctBaby.sctGebDat.lJahr);
    printf("----------------------\n");
}

void lineFeedBeseitigen(char *s){
    if (s[strlen(s)-1]=='\n')
        s[strlen(s)-1]='\0';
}
```

8 Schnittstellen mit dem Betriebssystem

Jedes Programm kann mit dem Betriebssystem kommunizieren, um z.B. Datem vom Betriebssystem zu erhalten oder an das Betriebssystem zu schicken.

8.1 Parameterübergabe an ein Programm

Jede Verknüfung unter Windows kann neben dem Namen der auszuführenden Exe-Datei (z.B. WinWord.exe) noch zusätzliche Parameter enthalten, die beim Programmstart an das jeweilige Programm übermittelt werden. In einer UNIX/LINUX-Shell oder dem DOS-Fenster werden diese Parameter einfach dem Namen der ausführbaren Datei getrennt durch ein Leerzeichen nachgestellt. Zum Beispiel wäre im vorangehenden Beispiel (der Babyverwaltung) ein Parameter denkbar, der angibt, wie die Kopie der Datei "baby.dat" heißen soll. Der Aufruf auf Kommandozeilenebene sähe dann wie folgt aus

```
baby schreiHals.dat
```

Das Programm baby.exe wird ausgeführt. Der Text "schreiHals.dat" wird an das Programm als Parameter übergeben.

Um die Parameter im Programm baby.exe auswerten zu können ist der Kopf der main-Routine folgendermaßen zu ändern:

```
int main(int argc, char *argv[])
```

Dabei sind

- `argc`: Anzahl der an das Programm übergebenen Parameter
- `argv`: Feld von Strings. Jedem String entspricht einem Parameterwert

Zu beachten ist, daß **`argv[0]`** standardmäßig der Name des ausgeführten Programms ist und argc deshalb mindestens 1 ist.

Im Beispiel

```
baby schreiHals.dat
```

ist `argc = 2`, `argv[0]="baby.exe"` und `argv[1]="schreiHals.dat"`.

Wenn wir zusätzlich auch noch die zu lesende Datei angeben, sieht der Aufruf von baby.exe wie folgt aus

```
baby baby.dat schreiHals.dat
```

Die Daten sind aus der Datei "baby.dat" zu lesen und in die Datei "schreiHals.dat" zu kopieren. Es ergibt sich folgendes geändertes Programm

```
// Bis main wie oben

int main(int argc, char *argv[]){
    char c80QuellDatei[80];
    char c80ZielDatei[80];

    FILE *FBaby;
    TBaby sctBabyFeld[100];
    int i=0,j;
    char ch;

    if (argc>1) strcpy(c80QuellDatei ,argv[1]);
    else strcpy(c80QuellDatei, "baby.dat" );

    if (argc>2) strcpy( c80ZielDatei,argv[2]);
    else strcpy(c80ZielDatei, "babyKopie.dat)" ;

    // Datei zum Lesen oeffnen
    if (FBaby=fopen(c80QuellDatei , "r")){
        printf("Eintraege in der Datei:\n");
        printf("----------------------\n");
        while (fgets(sctBabyFeld[i].cFeld120Name,120,FBaby)!=NULL){
             fgets(sctBabyFeld[i].cFeld120Mutter,120,FBaby);
             lineFeedBeseitigen(sctBabyFeld[i].cFeld120Name);
             lineFeedBeseitigen(sctBabyFeld[i].cFeld120Mutter);
             fscanf(FBaby,"%d", &sctBabyFeld[i].sGewicht);
             fscanf(FBaby,"%d", &sctBabyFeld[i].sGroesse);
             fscanf(FBaby,"%d%d%d", &sctBabyFeld[i].sctGebDat.sTag,
                 &sctBabyFeld[i].sctGebDat.sMonat, &sctBabyFeld[i].sctGebDat.lJahr);
             // Eingelesenen Datensatz ausgeben (Bildschirm)
             babyDatenAusgeben(sctBabyFeld[i]);
             // Zeilenumbruch zwischen den Datensätzen berücksichtigen!
             fscanf(FBaby,"%c", &ch);
             i++;
        }
        fclose(FBaby);
        // Daten auf Festplatte schreiben
        if (FBaby=fopen(c80ZielDatei , "w")){
            for (j=0; j<i; j++){
                fprintf(FBaby, "%s\n", sctBabyFeld[j].cFeld120Name);
                fprintf(FBaby, "%s\n", sctBabyFeld[j].cFeld120Mutter);
                fprintf(FBaby, "%d %d ", sctBabyFeld[j].sGewicht,
sctBabyFeld[j].sGroesse);
                fprintf(FBaby,"%d %d %d", sctBabyFeld[j].sctGebDat.sTag,
                   sctBabyFeld[j].sctGebDat.sMonat, sctBabyFeld[j].sctGebDat.lJahr);
```

```
                // Zeilenwechsel zwischen den Datensätzen
                if (j+1<i) fprintf(FBaby,"\n");
            }
            fclose(FBaby);

        }
        else
            printf("Fehler, kann Babys nicht kopieren :-)");
    }
    else
        printf("Fehler beim Öffnen der Datei, keine Babys vorhanden :-)");


    getchar();
    return 0;
}

// Rest wie oben!
```

Übungsaufgabe (hier und jetzt!)

Schreiben Sie ein Programm Binomialkoeffizient, das den Binomialkoeffizienten zweier nichtnegativer ganzer Zahlen n und k berechnet. Dabei sollen n und k als Argumente an das Programm übergeben werden. Welche Schwierigkeit ergibt sich?

Formel:

```
/   \
| n |       n!
|   |= ------------
| k |   (n-k)! k!
\   /
```