Übung 4

Erarbeitet von: M.Eng. Michael Finsterbusch
Modulverantwortlicher: Prof. Dr. rer. nat. Matthias Krause
Stand: 3. Oktober 2017

Ziel der Übung ist das Kennenlernen und Vertiefen von:

- File I/O
- Schleifen
- Kontrollstrukturen
- formatierte Ausgabe
- Strukturen

Abgabe:
- http://praktomat.hft-leipzig.de unter „Tutorial: DKMI/DAI-17 C-Progr.“
- sämtliche Abgabemodalitäten sind im Praktomat hinterlegt

Aufgaben

Gegeben ist die Header-Datei *uebung4.h*, in der Funktionsdeklarationen enthalten sind, sowie die Datei *uebung4.c*, in der die Funktionen implementiert werden. Kopieren Sie diese in Ihr Arbeitsverzeichnis. Die Funktionen sollen entsprechen den folgenden Vorgaben implementiert werden. Um die korrekte Funktionsweise Ihrer Implementation zu testen, verwenden Sie die Funktionen in der `main()`-Funktion, die in der Datei *main.c* implementiert werden soll. Diese kann zum Beispiel so aussehen:

Listing 1: main.c

```c
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include "uebung4.h"

int main(int argc, char* argv[])
{
  int result;
  char* filename = "test.txt";

  result = print_file(filename);
  if(result == 1)
    printf("Fehler beim Oeffnen der Datei '%s': %s\n", filename,
       ➥strerror(errno));

  [...]

  return 0;
}
```

1. Implementieren Sie die Funktion `print_file()` in der Datei *uebung4.c*. Dieser Funktion wird als einziger Parameter der Name/Pfad einer Datei übergeben. Diese Datei soll geöffnet, ausgelesen und der Inhalt auf die Konsole ausgegeben werden. Dazu muss die Datei

zum Lesen mit `fopen()` geöffnet werden. Als Rückgabewert erhält man einen Dateizeiger (file descriptor) vom Typ `FILE*`. Kann die Datei, aus welchem Grund auch immer, nicht geöffnet werden, ist der Rückgabewert `NULL` und die Funktion soll sich mit einem Fehlerwert beenden (siehe unten). Lesen Sie dann mit der Funktion `fread` Daten in 512 Byte Blöcken aus der Datei in einen entsprechend dimensionierten Puffer vom Typ `char`:

```
size = fread((void*)buffer, 1, BUFFER_SIZE, file);
```

Geben Sie anschließend alle eingelesenen Zeichen mit `printf()` aus:

```
printf("%.*s", size, buffer);  // Was bedeutet der
   ➥Formatstring?
```

Wiederholen Sie diesen Vorgang, bis die komplette Datei ausgegeben wurde oder ein Fehler auftritt. Ist das Dateiende erreicht oder tritt ein Fehler auf, werden weniger als die angeforderte Anzahl von Zeichen ausgegeben. Mit `feof()` bzw. `ferror()` können Sie prüfen, welcher Fall vorliegt. Schließen Sie die Datei mit `fclose()` bevor die Funktion verlassen wird. Die Funktion soll folgende Werte zurückgeben:

- **0** Bei Erfolg
- **1** Fehler beim Öffnen der Datei
- **2** Fehler beim Auslesen der Datei

Der Programmablaufplan in Abbildung 1 stellt den gesamten Funktionsumfang dar.

Informieren Sie sich mit Hilfe der beigefügten Dokumentation über die Funktionsweise, Parameter und Rückgabewerte der zu verwendeten Funktionen.

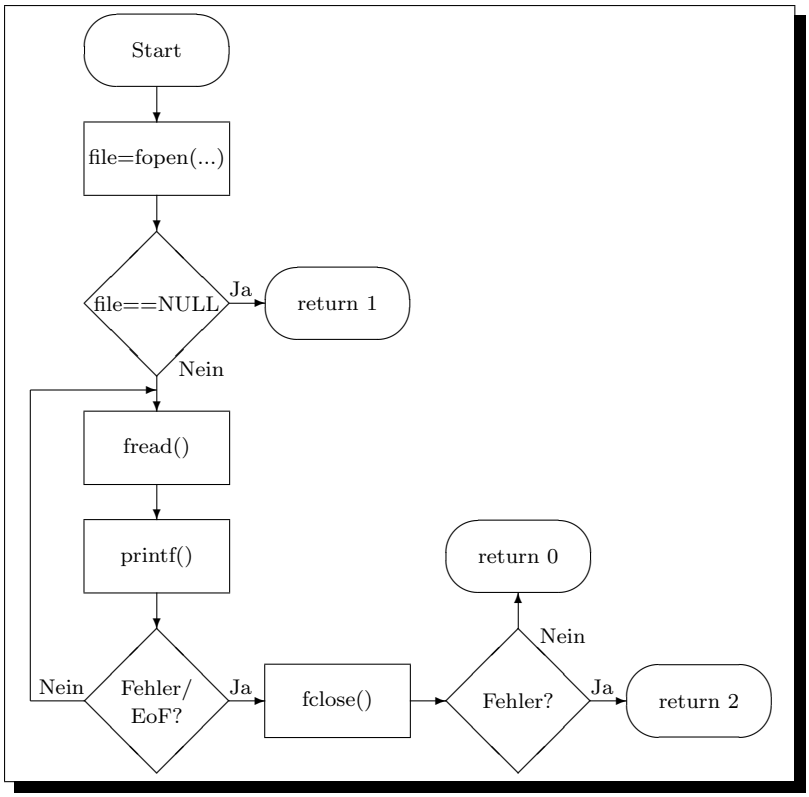

Abbildung 1: Programmablaufplan der Funktion `print_file()`

2. Implementieren Sie die Funktion `copy_file()` in der Datei *uebung4.c*. Die Aufgabe der Funktion ist es, eine Datei zu kopieren. Die Namen der Quell- und Zieldatei werden als Parameter übergeben. Zuerst müssen beide Dateien geöffnet werden. Die Quelldatei wird dabei zum Lesen und die Zieldatei zum Schreiben geöffnet (siehe Funktionsreferenz). Aus der Quelldatei werden sequenziell Daten mit `fread()` blockweise ausgelesen und mit `fwrite()` in die Zieldatei geschrieben. Am Ende sollen beide Dateien wieder geschlossen werden. Die Funktion soll folgende Werte zurückgeben:

   0 Bei Erfolg

   1 Fehler beim Öffnen der Quelldatei

   2 Fehler beim Öffnen der Zieldatei

   3 Fehler beim Auslesen der Quelldatei

   4 Fehler beim Schreiben der Zieldatei

3. Implementieren Sie die Funktion `addressbook_save()` in der Datei *uebung4.c*. Die Aufgabe der Funktion ist es, die Informationen eines Adressbuches in einer Datei zu speichern. Jeder Datensatz des Adressbuches hat eine fest vorgegebene Struktur. Diese ist in Listing 2 aufgeführt.

Listing 2: Struktur eines Adressbucheintrags

```
1  enum e_addressbook_sex {
2    ADDRESSBOOK_SEX_MALE=0,
3    ADDRESSBOOK_SEX_FEMALE,
4    ADDRESSBOOK_SEX_UNKNOWN
5  };

7  struct s_date_of_birth {
8    int year:23;
9    unsigned int month:4;
10   unsigned int day:5;
11 };

13 struct s_addressbook_record {
14   unsigned int id;
15   char name[30];
16   char firstname[30];
17   enum e_addressbook_sex sex;
18   struct s_date_of_birth birth;
19 };
```

Die Funktion `addressbook_save()` erhält als Parameter den Dateinamen, in den die Daten gespeichert werden sollen, ein Array mit Datensätzen, die Länge des Arrays sowie die Information ob die Datei neu angelegt oder die Daten an die Datei angefügt werden sollen. Da die Datensätze eine feste Struktur besitzen, können diese einfach mit der Funktion `fwrite` in die Datei geschrieben werden:

```
size=fwrite((void*)&addressbook[i],sizeof(struct
   ➥s_addressbook_record),1,file);
```

Die Funktion soll folgende Rückgabewerte zurückliefern:

0 Bei Erfolg

1 Fehler beim Öffnen der Zieldatei

2 Fehler beim Schreiben der Zieldatei

4. Implementieren Sie die Funktion `addressbook_load()` in der Datei *uebung4.c*. Die Aufgabe der Funktion ist es, die Informationen eines Adressbuchs, die in Aufgabe 3 gespeichert wurden, aus einer Datei zu laden. Aufbau und Struktur der Daten ist so wie in Aufgabe 3 beschrieben. Mit Hilfe der Funktion `fread` können die Datensätze als ganzes ausgelesen werden:

```
size=fread((void*)&addressbook[i],sizeof(struct
   ➥s_addressbook_record),1,file);
```

Die Funktion soll folgende Rückgabewerte zurückliefern:

0 Bei Erfolg

1 Fehler beim Öffnen der Quelldatei

2 Fehler beim Lesen der Quelldatei

3 Weitere Datensätze in der Quelldatei, aber das Ziel-Array ist voll

5. (Optional) Implementieren Sie die Funktion `print_id3()` in der Datei *uebung4.c*. Die Aufgabe der Funktion ist es, den ID3-Tag einer MP3-Datei auszulesen und auszugeben. Das ID3-Tag[1] enthält Zusatzinformationen (Metadaten), die in Audiodateien des MP3-Formats enthalten sein können. Das ID3-Tag ist optional. Wenn es in einer MP3-Datei enthalten ist, befindet es sich immer am Ende der Datei. Das ID3v1 Tag hat folgende Struktur [http://id3.org, http://id3.org/ID3v1]:

Offset*	Länge*	Bedeutung
0	3	Kennzeichnung des Tags mit dem Wert „TAG“
3	30	Songtitel
33	30	Künstler/Interpret
63	30	Album
93	4	Erscheinungsjahr
97	30	Beliebiger Kommentar
127	1	Genre

* Angaben in Byte

Alle Felder außer `Genre` enthalten Strings, die allerdings nicht null-terminiert sind! `Genre` ist ein 8-Bit Zahlenwert, der als Index für die Genretabelle dient (siehe Tabelle 1). Die ersten drei Byte/Zeichen des ID3-Tags enthalten den Wert `TAG`. Ist dies nicht so, enthält die MP3-Datei kein ID3-Tag.

Definieren Sie die Struktur **struct id3tag**, die dem Format des ID3-Tags entspricht, in der Datei *uebung4.c*!

Um das ID3-Tag aus einer Datei auszulesen, öffnen Sie zunächst die Datei im Lesemodus. Anschließend positionieren Sie mit der Funktion `fseek()` den Dateipositionszeiger so, dass er 128 Byte vor dem Ende der Datei steht (siehe dazu die Funktionsreferenz). Prüfen Sie, ob es sich bei den ausgelesenen Daten um ein ID3-Tag handelt (Beginn mit `TAG`). Danach kann das Tag ausgelesen werden:

[1] In dieser Übung soll nur ID3v1, die erste Version des Tags, unterstützt werden.

```
struct id3tag tag;
[...]
size = fread((void*)&tag, sizeof(tag), 1, file);
```

Geben Sie anschließend die Informationen des Tag zeilenweise aus (eine Zeile pro Attribut).

Die Funktion soll als Rückgabewert folgende Werte zurückgeben:

- **0** Bei Erfolg
- **1** Fehler beim Öffnen der Datei
- **2** Kein ID3-Tag in der Datei
- **3** Fehler beim Auslesen der Datei

Der Programmablaufplan in Abbildung 2 stellt den gesamten Funktionsumfang dar.

Anhang

Tabelle 1: Liste der ID3-Tag Genres [http://id3.org]

0. Blues	32. Classical	64. Native American	96. Big Band
1. Classic Rock	33. Instrumental	65. Cabaret	97. Chorus
2. Country	34. Acid	66. New Wave	98. Easy Listening
3. Dance	35. House	67. Psychadelic	99. Acoustic
4. Disco	36. Game	68. Rave	100. Humour
5. Funk	37. Sound Clip	69. Showtunes	101. Speech
6. Grunge	38. Gospel	70. Trailer	102. Chanson
7. Hip-Hop	39. Noise	71. Lo-Fi	103. Opera
8. Jazz	40. AlternRock	72. Tribal	104. Chamber Music
9. Metal	41. Bass	73. Acid Punk	105. Sonata
10. New Age	42. Soul	74. Acid Jazz	106. Symphony
11. Oldies	43. Punk	75. Polka	107. Booty Bass
12. Other	44. Space	76. Retro	108. Primus
13. Pop	45. Meditative	77. Musical	109. Porn Groove
14. R&B	46. Instrumental Pop	78. Rock & Roll	110. Satire
15. Rap	47. Instrumental Rock	79. Hard Rock	111. Slow Jam
16. Reggae	48. Ethnic	80. Folk	112. Club
17. Rock	49. Gothic	81. Folk-Rock	113. Tango
18. Techno	50. Darkwave	82. National Folk	114. Samba
19. Industrial	51. Techno-Industrial	83. Swing	115. Folklore
20. Alternative	52. Electronic	84. Fast Fusion	116. Ballad
21. Ska	53. Pop-Folk	85. Bebob	117. Power Ballad
22. Death Metal	54. Eurodance	86. Latin	118. Rhythmic Soul
23. Pranks	55. Dream	87. Revival	119. Freestyle
24. Soundtrack	56. Southern Rock	88. Celtic	120. Duet
25. Euro-Techno	57. Comedy	89. Bluegrass	121. Punk Rock
26. Ambient	58. Cult	90. Avantgarde	122. Drum Solo
27. Trip-Hop	59. Gangsta	91. Gothic Rock	123. A capella
28. Vocal	60. Top 40	92. Progressive Rock	124. Euro-House
29. Jazz+Funk	61. Christian Rap	93. Psychedelic Rock	125. Dance Hall
30. Fusion	62. Pop/Funk	94. Symphonic Rock	
31. Trance	63. Jungle	95. Slow Rock	

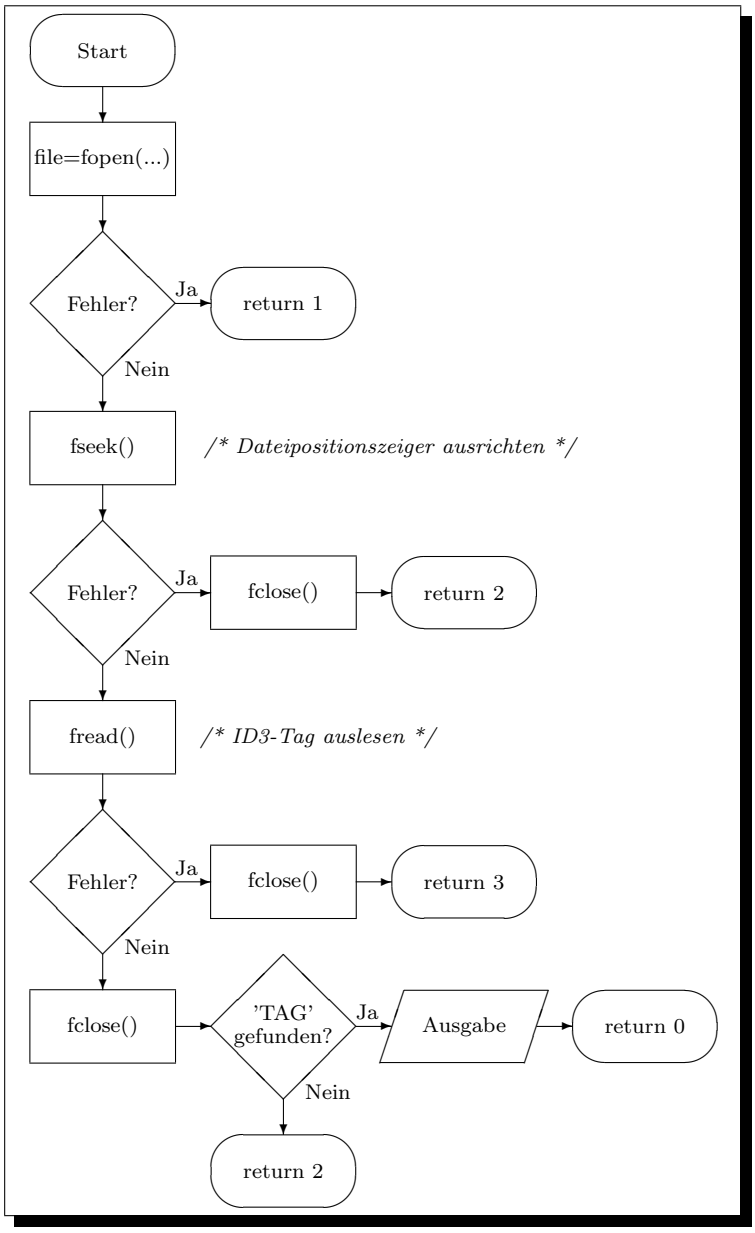

Abbildung 2: Programmablaufplan der Funktion `print_id3()`