Hochschule **Rosenheim**
University of Applied Sciences

Prozedurale Programmierung

Dateien

Hochschule Rosenheim - University of Applied Sciences

WS 2018/19

Prof. Dr. F.J. Schmitt

Überblick

Problem: Wie speichert man Daten dauerhaft?

- Datenströme
- Öffnen und Schließen von Datenströmen
- Ein- und Ausgabe

Dateien

- Objekte, die vom Betriebssystem verwaltet werden
- Standard-Bibliothek
  - beinhaltet eine Reihe von Funktionen, die das komfortable Arbeiten mit Dateien ermöglichen
  - bietet zur Ein- und Ausgabe sogenannte Datenströme (engl. streams) an
  - erlaubt den Zugriff auf Dateien nur über Datenströme

Datenströme

- Objekte, in die Informationen geschrieben oder aus denen Informationen gelesen werden können
- Zugriff auf Dateien erfolgt in C über gepufferte Datenströme
- Einige Ströme sind immer vorhanden und müssen nicht explizit geöffnet werden:

Standard-Strom	Beschreibung
`stdout`	Standard-Strom für die Ausgabe
`stdin`	Standard-Strom für die Eingabe
`stderr`	Standard-Strom für Fehlermeldungen (anders als stdout ungepuffert)

- Header-Datei `stdio.h` muss inkludiert sein

Öffnen und Schließen von Datenströmen

- Ströme werden durch File Handles (Objekte des Typs FILE) repräsentiert

```
FILE *datei;   // file handle
datei = fopen("beispiel.txt", "r");
//...
fclose(datei);
```

- Funktionen für Ströme:

Standard-Strom	Beschreibung
`fopen(s,m)`	Öffnet Datei s mit dem Modus m und gibt einen Strom F zurück
`fclose(F)`	Schließt den Strom F
`fflush(F)`	Leert den Strom F

Modi für fopen

Modus	Beschreibung
"r"	Öffnen zum Lesen
"r+"	Öffnen zum Lesen und Schreiben (Datei muss existieren)
"w"	Öffnen zum Schreiben (evtl. vorhandene Datei wird überschrieben)
"w+"	Öffnen zum Lesen und Schreiben (evtl. vorhandene Datei wird überschrieben)
"a"	Öffnen zum Schreiben. Ist Datei vorhanden wird angehängt.
"a+"	Öffnen zum Lesen und Schreiben, sonst wie "a"

- Liefert Zeiger auf einen File Handle zurück oder NULL, wenn der Strom nicht geöffnet werden konnte.

Zusatzmodi für fopen

Modus	Beschreibung
"t"	Öffnen im Textmodus Windows: Daten werden konvertiert • Schreiben: \n → \r\n • Lesen: \r\n → \n Unix: Modus wird ignoriert, da unnötig; verwendet immer „b"
"b"	Öffnen im Binärmodus Daten werden unverändert geschrieben

- Angabe des Zusatzmodus erfolgt direkt nach dem Hauptmodus, also z.B. „rt", „w+b"

Beispiel

- Überprüfen, ob Datei korrekt geöffnet wurde

```
#include <stdio.h>

int main(void)
{
   FILE *datei;   // file handle
   datei = fopen("beispiel.txt", "rt");

  if(datei == NULL)
     printf("Fehler beim Öffnen der Datei!\n");
   else
    {
      //...
      fclose(datei);
    }
}
```

Ein- und Ausgabe (1)

Textbasierte Funktionen

Funktion	Beschreibung
fprintf (F, f, . . .)	Wie printf, nur erfolgt die Ausgabe in den angegebenen Strom F
fscanf (F, f, . . .)	Wie scanf, nur wird aus dem Strom F gelesen
fgets (s, n, F)	Liest eine Zeile aus dem Strom F und schreibt sie nach s, aber nicht mehr als n Zeichen
fputs (s, F)	Schreibt String s nach F
fgetc (F)	Liest ein Zeichen aus dem Strom F
fputc (c, F)	Schreibt das Zeichen c in den Strom F

Ein- und Ausgabe (2)

- Öffnen, Beschreiben und Schließen einer Textdatei

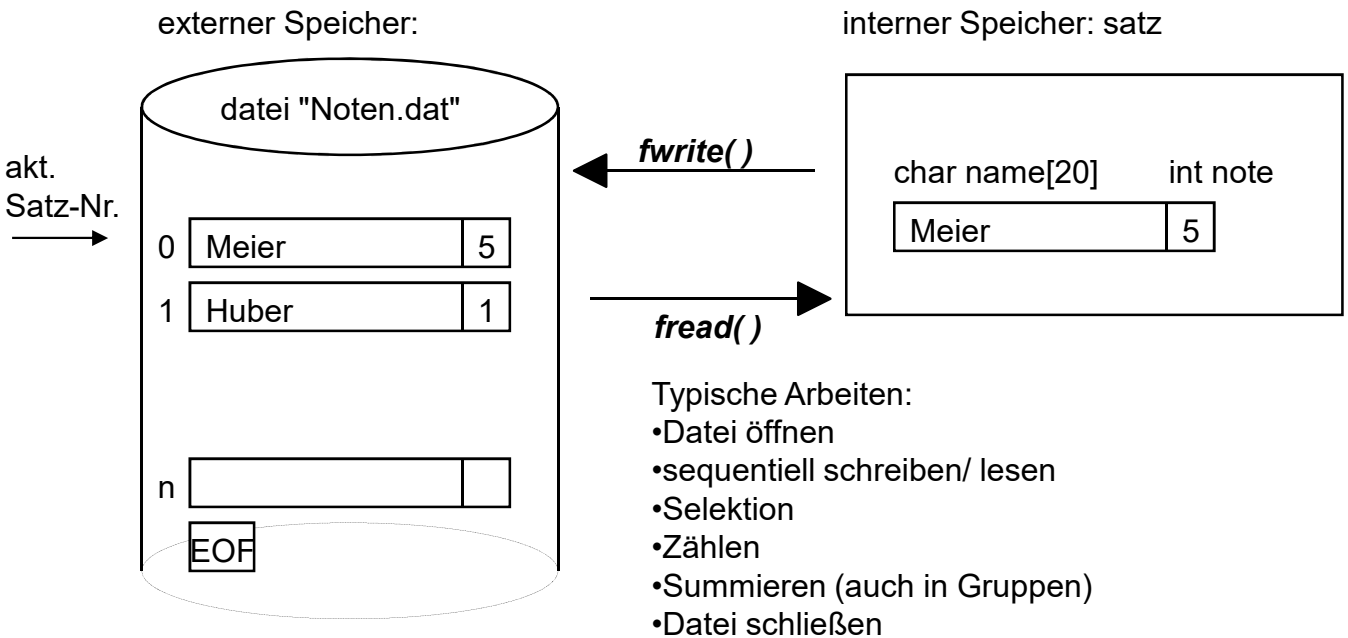
```
#include <stdio.h>

int main(void)
{
   FILE *datei;   //filehandle
   datei = fopen("beispiel.txt", "wt");

   if(datei == NULL)
      fprintf(stderr,"Fehler beim Öffnen der Datei!\n");
   else
    {
      fprintf(datei, "Hallo Welt!\n");
      fclose(datei);
    }
}
```

Ein- und Ausgabe (3)

- Textdateien: Speicherung in ASCII-Code
- Binäre Dateien: Speicherung so, wie die Daten im Speicher stehen

  $\Rightarrow$ Blockorientiertes Lesen und Schreiben

Beispiel



Typische Arbeiten:
- Datei öffnen
- sequentiell schreiben/ lesen
- Selektion
- Zählen
- Summieren (auch in Gruppen)
- Datei schließen

fread() / fwrite()

Rückgabewert = Anzahl der erfolgreich geschriebenen oder gelesenen Elemente

Funktion	Beschreibung
`fread(b, g, n, F)`	Liest n Elemente der Größe g Byte aus F und speichert sie in b
`fwrite(b, g, n, F)`	Schreibt n Elemente der Größe g Byte von b nach F

Deklaration

- ```
  size_t fread
  (void *b, size_t g, size_t n, FILE *F);
  ```
- ```
  size_t fwrite
  (const void *b, size_t g, size_t n, FILE *F);
  ```

Ein- und Ausgabe (5)

Beispiel: Speicherung Adressbuch

```
Adresse_t Adressbuch[] = {...};
const int LEN = sizeof(Adressbuch) / sizeof(Adresse_t);

FILE *datei;
datei = fopen("adressen", "w");
if(datei == NULL)
   fprintf(stderr,"Fehler beim Öffnen der Datei!\n");
else
{
   if (fwrite(Adressbuch, sizeof(Adresse_t), LEN, datei) < LEN)
      fprintf(stderr,"Daten konnten nicht geschrieben werden\n");
   fclose(datei);
}
```

Erkennung Dateiende

- Funktion: `feof (FILE *F);`
- Positiver Rückgabewert, wenn Dateiende erreicht ist
- Dateiende wird erst erkannt , **nachdem** fread auf das Dateiende gestoßen ist

Typisches Vorgehen Lesen

```
while (fread(&satz, sizeof(satz), 1, datei))
{
   // Verarbeitung des gelesenen Einzelsatzes
   // z.B. selektieren, (in Gruppen) zählen, summieren
}
```

oder

```
fread(&satz, sizeof(satz), 1, datei);
while (!feof(datei))
{
   // Verarbeitung des gelesenen Einzelsatzes
   // z.B. selektieren, (in Gruppen) zählen, summieren
   fread(&satz, sizeof(satz), 1, datei);
}
```

Positionierung des Dateizeigers

- jede Datei hat einen Schreib-/Lesezeiger
  - gibt aktuelle Position innerhalb der Datei an
- der Zeiger kann mit `fseek()` geändert werden
  - z.B., um direkt an eine bestimmte Position zu springen
- den aktuellen Zeiger erhält man mit `ftell()`
- `rewind()` springt an den Dateianfang
- der Zeiger bewegt sich byteweise
- das erste Byte hat die Position 0
- bei jedem Schreib-/Lesezugriff erhöht sicher der Zeiger um die Zahl der übertragenen Bytes

fseek()

- `int fseek(FILE *F, long offset, int origin);`
- Rückgabewert: 0, wenn erfolgreich
- `offset`: Anzahl Byte, um die der Zeiger bewegt werden soll
- `origin`: Bezugspunkt der Bewegung
  - `SEEK_SET` Dateianfang
  - `SEEK_CUR` aktuelle Position
  - `SEEK_END` Dateiende

fseek() – Beispiele

- gehe an Dateiende:
```
fseek(datei, 0L, SEEK_END);
```
- gehe an Dateianfang:
```
fseek(datei, 0L, SEEK_SET);
```
- gehe von Dateianfang 100 Byte vor:
```
fseek(datei, 100L, SEEK_SET);
```
- gehe von aktueller Position 100 Byte vor:
```
fseek(datei, 100L, SEEK_CUR);
```
- gehe von aktueller Position 100 Byte zurück:
```
fseek(datei, -100L, SEEK_CUR);
```

rewind()

- `void rewind(FILE *F);`
- springt an Dateianfang

- `rewind(datei)`
  hat gleichen Effekt wie
  `fseek(datei, 0L, SEEK_SET);`

ftell()

- `long ftell(FILE *F);`
- Rückgabewert:
  - aktuelle Position des Schreib-/Lesezeigers in Byte, gemessen vom Dateianfang
  - negativer Wert, wenn Fehler aufgetreten ist

fseek() – Anmerkungen

- wie man sieht, ist der Offset als long definiert
- man kann fseek() damit also nur in Dateien bis zu einer Größe von 2GB verwenden
- für größere Dateien gibt es _fseeki64() und _ftelli64() (nicht ANSI C)

Zusammenfassung Beispiele

- Annahme
  - `t_datensatz satz;` *// struct, definiert Datenstruktur*
  - `FILE *datei;` *// Zeiger auf Datei*
  - `long curpos;` *// Position Schreib-/Lesezeiger*
- Beispiele

```
datei = fopen ("Noten.dat", "w+b");
fwrite (&satz, sizeof(satz), 1, datei);
fread  (&satz, sizeof(satz), 1, datei);
feof   (datei);
fseek  (datei, -100L, SEEK_CUR);
curpos = ftell (datei);
rewind (datei);
fclose (datei);
```

Anmerkungen

- Dateien, die mit `fwrite` geschrieben wurden, können nur mit `fread` gelesen werden
- Daten werden mit `fwrite` in binärer Form abgespeichert!
- Problematisch: Portabilität
  - beim Lesen/Schreiben von struct kann es zu Problemen kommen, da typischerweise mit Füllbytes z.B. auf eine 4 Byte Grenze aufgefüllt wird
  - dies ist von Compiler zu Compiler unterschiedlich und kann auch innerhalb eines Compilers eingestellt werden
  - sicherer (und aufwändiger) ist das Schreiben/Lesen ohne Verwendung einer struct, wenn die Dateien von anderen Programmen gelesen werden müssen
  - dann besteht „nur“ noch das Little/Big-Endian Problem zwischen verschiedenen CPU-Architekturen

Zusammenfassung

- Standarddatenströme
- Öffnen/Schließen von Dateien
  - fopen(), fclose()
- Ein-/Ausgabe von Text
  - fprintf(), fscanf(), fgets(), fputs(), fgetc(), fputc()
- Blockorientierte Ein-/Ausgabe
  - fread(), fwrite()
- weitere Dateifunktionen
  - End-of-File: feof()
  - Dateizeiger positionieren: fseek()
  - Dateizeiger auslesen: ftell()