Technische Informatik 2

C/C++ Programmierung

3. Dateien in C

Prof. Dr. Ivo Wolf

Institut für Medizinische Informatik

hochschule mannheim

Dateiverarbeitung

- Allumfassendes Dateikonzept in Unix:
  jedes Gerät wird als Datei betrachtet.
- starke Konsistenz zwischen Funktionen zur:
  - Bildschirm-E/A und
  - Funktionen zur Datei-E/A
    (gleiche Aufrufsyntax und -semantik)

Dateiverarbeitung - Dateikonzept in C

- Datei = beliebig lange, unstrukturierte Folge von Einzelzeichen.
- Die Dateiverarbeitung erfolgt prinzipiell immer zeichenorientiert.
- für C-Programme ist es ganz egal ob sie auf
  - einer Textdatei,
  - einer Datenbank oder
  - einem ausführbaren Programm arbeiten.
- Erst mit der Verarbeitung durch ein Programm erhält eine Folge von Einzelzeichen eine tiefere Bedeutung.
- Aus Gründen der Bequemlichkeit werden viele Funktionen bereitgestellt, die einer Datei bestimmte strukturelle Eigenschaften unterstellen.

Dateiverarbeitung - Phasen

Zugriff auf Dateien erfolgt in 3 Phasen :

1. Öffnen der Datei.
2. Zugriff auf die geöffnete Datei.
3. Schließen der Datei.

Öffnen einer Datei

Syntax:

```
#include <stdio.h>

FILE* fopen (char *fname, char *mode)
```

Semantik:

Öffnen der Datei fname im Modus mode (z.B. lesen, schreiben)

Rückgabewert:

Zeiger auf die Datei vom Datentyp FILE oder
NULL-Zeiger, wenn sich Datei nicht öffnen läßt.

Achtung:

In Visual Studio gilt `fopen` als „deprecated“ (überholt/veraltet) am
Damit es funktioniert, vor dem Include von „`stdio.h`“:

```
#define _CRT_SECURE_NO_DEPRECATE
```


Öffnen einer Datei

```
#include <stdio.h>
main ()
{
  FILE* f1;
  f1 = fopen ("test.txt", "r");
  if (f1 == NULL) {
    printf("Kann Datei nicht öffnen\n");
  }
}
```

Bearbeitungsmodi

Modus	Bedeutung
r	Öffnen einer existierenden der Datei zum Lesen.
r+	Öffnen einer existierenden Datei zum Verändern (Lesen + Schreiben).
w	Anlegen einer neuen Datei zum Schreiben.
w+	Anlegen einer neuen Datei zum Verändern (Schreiben + Lesen). Eine bereits existierende Datei gleichen Namens wird zuvor gelöscht.
a	Öffnen der Datei zum Schreiben am Ende.
a+	Öffnen der Datei zum Lesen oder Schreiben am Ende. Falls die Datei noch nicht vorhanden ist, wird sie automatisch angelegt.

Bearbeitungsmodi

Modus	Bedeutung
b	Öffnen der Datei im Binärmodus: Jedes einzelne Zeichen wird so auf Datei geschrieben wie es von der Schreibfunktion übergeben wird bzw. so an das Programm weitergegeben, wie es auf der Datei steht. D.h. es werden in keiner Richtung implizite Konvertierungen vorgenommen.
t (default)	Öffnen der Datei im Textmodus. Es werden einige (system-spezifische) Konvertierungen vorgenommen. Die Datei sollte nur lesbare Textzeichen enthalten.

Binärmodus vs Textmodus

```
Z   ⟶  liefert 'Z'
1   ⟶  liefert '1'
\r  ⟶  liefert 0x0D
\n  ⟶  liefert 0x0A
Z   ⟶  liefert 'Z'
2   ⟶  liefert '2'
^Z  ⟶  liefert 0x1A
       liefert EOF
```

Binärmodus

```
Z   ⟶  liefert 'Z'
1   ⟶  liefert '1'
\r  ↘
\n  ⟶  liefert 0x0A
Z   ⟶  liefert 'Z'
2   ⟶  liefert '2'
^Z  ⟶  liefert EOF
```

Textmodus

Schließen einer Datei

Syntax:

```
#include <stdio.h>
FILE *fclose (FILE *fp)
```

Semantik:

Schließen einer geöffneten, mit dem Datei-Zeiger fp verbundenen Datei .

Mit dem Schließen wird der Inhalt der Dateipuffer physikalisch in die Datei geschrieben.

Rückgabewert:

Nach erfolgreichem Schließen: 0,

Im Fehlerfall: EOF.

Formatierte Ein- / Ausgabe

Syntax:

```
#include <stdio.h>
int fprintf (FILE *fp, Formatstring,
    Argumentliste);
int fscanf (FILE *fp, Formatstring,
Argumentliste);
```

Semantik:

wie printf bzw. scanf

Rückgabewert:

wie printf bzw. scanf

Beispiel

```
#include <stdio.h>
main ()
{
  FILE *f1;
  if ((f1 = fopen("test.txt", "w+")) == NULL)
      printf("Kann Datei test.txt nicht öffnen!\n");
  fprintf(f1, "Dies ist die erste Zeile\n");
  fprintf(f1, "Dies ist ein int: %i\n", 42);

  fclose(f1);
}
```

Positionieren in Datei

Syntax:

```
#include <stdio.h>
int fseek (FILE *fp, long offset, int basis)
```

Semantik:

Positionieren des Satzzeigers.

Position = basis + offset; offset kann positiven oder negativen Wert annehmen.

Rückgabewert:

Nach erfolgreichem Positionieren: 0,

Im Fehlerfall: Wert ungleich 0.

Positionieren in Datei

- Für `basis` gibt es 3 Bezugspunkte, die durch 3 symbolische Konstanten definiert sind:

Symbol	numerischer Wert	Offsetberechnung ab
SEEK_SET	0	Anfang der Datei
SEEK_CUR	1	aktuelle Position des Satzzeigers
SEEK_END	2	Ende der Datei

Aktuelle Position des Satzzeigers

Syntax:

```
#include <stdio.h>
long ftell (FILE *fp)
```

Semantik:

Ermittlung der Position des Satzzeigers in der Datei, die mit dem Datei-Zeiger `fp` verbunden ist.

Rückgabewert:

Offset des Satzzeigers vom Beginn der Datei,
Im Fehlerfall: -1L.

Speicherblöcke lesen

Syntax:

```
#include <stdio.h>
int fread (void *buf, int size, int cnt, FILE *fp)
```

Semantik:

cnt Elemente der Größe size einlesen

Rückgabewert:

Anzahl der tatsächlich gelesenen Datensätze.
Falls < cnt, dann Fehler.

Lesen von typisierten Dateien mittels des Lesens von Speicherblöcken

```
#include <stdio.h>
typedef struct { .... } ARTIKEL;

int main ()
{
  ARTIKEL record [100];
  FILE *f1;
  if ((f1 = fopen ("artikel.dat", "rb")) != NULL) {
  if (fread (record, sizeof(ARTIKEL), 100, f1) < 100 &&
      !feof(f1))
    fprintf(stderr, "Fehler beim Lesen der Datei
  artikel.dat“);
   /*
       100 Datensätze, die record stehen, verarbeiten
   */
  }
}
```


Speicherblöcke schreiben

Syntax:

```
#include <stdio.h>
int fwrite (void *buf, int size, int cnt, FILE *fp)
```

Semantik:

cnt Elemente der Länge size aus dem Speicherbereich, auf den buf zeigt, in die Datei schreiben

Rückgabewert:

Anzahl der tatsächlich geschriebenen Datensätze.

Falls < cnt, dann Fehler.

Physikalisches Schreiben des Puffers

Syntax:

```
#include <stdio.h>

int fflush (FILE *fp)
```

Semantik:

gepufferte, noch nicht geschriebene Daten, physikalisch in die Datei schreiben.

Rückgabewert:

Nach erfolgreichem Leeren: 0,
Im Fehlerfall: EOF.

Schreiben eines Zeichens*

Syntax:

```
#include <stdio.h>

int putc (int c,  FILE *fp)

int fputc (int c,  FILE *fp)
```

Semantik:

Schreibt ein einzelnes Zeichen c auf die aktuelle Position der Datei, die mit dem Datei-Zeiger fp verbunden ist. Nach jedem Aufruf wird der (unsichtbare) Satzzeiger um eine Position weiterbewegt.

Rückgabewert:

Das ausgegebene Zeichen.
Im Fehlerfall: EOF

Schreiben eines Zeichens*

```
#include <stdio.h>
main ()
{
  FILE *f1;
  char c;
  if ((f1 = fopen ("test.txt", "a")) == NULL)
    fprintf(stderr, "Datei lässt sich nicht öffnen!\n");
  else {
    for (c = 'A'; c <= 'Z' && putc(c, f1) != EOF; c++);
    printf("Buchstaben an Datei test.txt angehängt\n");
  }
}
```


Lesen eines Zeichens*

Syntax:

```
#include <stdio.h>
int getc (FILE *fp)
```

Semantik:

Liest ein einzelnes Zeichen von der aktuellen Position der Datei, die mit dem Datei-Zeiger fp verbunden ist. Nach jedem Aufruf wird der (unsichtbare) Satzzeiger um eine Position weiterbewegt.

Rückgabewert:

Das eingelesene Zeichen.

Bei Dateiende und im Fehlerfall: EOF

Lesen eines Zeichens*

```
#include <stdio.h>
main ()
{ FILE *f1;
  char c;
  if ((f1 = fopen ("test.txt", "r")) == NULL)
     fprintf(stderr, "Datei lässt sich nicht
  öffnen!\n");
  else {
    while ((c = getc(f1)) != EOF)
    putchar(c);
  }
}
```


Zusammenfassendes Beispiel

```
#include <stdio.h>
main ()
{
  FILE *f1, *f2;
  char name[100];
  char c;
  int i;
  printf("Name der Quelldatei eingeben: ");
  scanf("%99s", name);
  if ((f1 = fopen (name, "r")) == NULL)
    printf("Kann Datei %s nicht öffnen!\n", name);
  else {
    if ((f2 = fopen ("cnt.txt", "w+")) == NULL)
      printf("Kann Datei cnt.txt nicht öffnen!\n");
    else {
    do {
       i=0;
       while ((c = getc(f1)) != EOF && c != '\n') i++;
       fprintf(f2, "%i\n", i);
    } while (c != EOF);
    fclose(f2);
    }
  fclose(f1);
  }
}
```