

Dateioperationen in C

27.05.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K12_Dateioperationen.ppt, Seite 1

Dateneingabe und Datenausgabe in C

- In diesem Kapitel wollen wir zeigen:
 - ➔ wie man auf Dateien und Peripherie zugreifen kann
 - ➔ was Datenströme sind
 - ➔ was Text- und Binärdateien sind
 - ➔ was sequentieller Dateizugriff (*sequential access*) ist
 - ➔ was wahlfreier Dateizugriff (*random access*) ist
 - ➔ welche Bibliotheksfunktionen für Dateizugriffe zur Verfügung stehen.

27.05.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K12_Dateioperationen.ppt, Seite 2

Datenströme

- Ein ➡ **Datenstrom**
ist **Quelle** oder **Ziel** von Daten.
- Er wird mit einer Datei auf einem Peripheriegerät
(z.B. Festplatte, Drucker, Scanner) verbunden.
- Man unterscheidet:
 - ➡ **Textdatenstrom**
 - ➡ **Binärdatenstrom**
- Bei Textdateien unter *MS DOS* wird im Unterschied
zu Binärdateien:
 - ➡ beim Schreiben eines Zeilenendes ein **LF**
(*line feed*) durch ein **CR** (*carriage return*) **LF** ersetzt.
 - ➡ beim Lesen die umgekehrte Ersetzung vorgenommen.

27.05.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K12_Dateioperationen.ppt, Seite 3

Textdatenströme

Textdatenstrom:

- Ein **Textdatenstrom** ist eine **Folge von Zeilen**.
- Jede Zeile enthält keines oder mehrere *ASCII*-Zeichen.
 - Jede Zeile wird mit **'\n'** abgeschlossen.
 - Bei der Eingabe und Ausgabe sind
Zeichenumwandlungen (Formatierung) notwendig
(z.B. **'\n'** in *ASCII CR + LF*)
- Eine Datei, die Textdatenströme enthält, nennt man
 - ➡ **Textdatei**
 - Eine **Textdatei** kann mit einem üblichen
➡ **Texteditor** (z.B. *MS Windows' Notepad*)
bearbeitet werden.

27.05.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K12_Dateioperationen.ppt, Seite 4

Binärdatenströme

➡ Binärdatenstrom

- Ein Binärstrom ist eine Folge von Bytes zur Speicherung interner Daten.
- Die Daten sind in der Regel nicht formatiert.
- Eine Datei, die Binärströme enthält, nennt man

➡ Binärdatei.

27.05.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K12_Dateioperationen.ppt, Seite 5

Speicherung von Datenströmen

- Häufig sind Datenströme in Dateien **auf der Festplatte** eines Rechners gespeichert.
 - Einem Datenstrom ist ein **eindeutiger Dateiname** zugeordnet:
Name.Dateierweiterung
- Der **Dateiname** (*file name*) ist im Rahmen der Vorgaben des Betriebssystems frei wählbar.
- Die **Dateierweiterung** (*file extension*) gibt normalerweise Auskunft über den Typ der gespeicherten Daten.

27.05.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K12_Dateioperationen.ppt, Seite 6

Dateinamen im Betriebssystem *MS DOS* (1)

- Beim **Betriebssystem *MS DOS*** darf:
 - ➔ der **Dateiname** nur aus **maximal 8 Buchstaben**, Zahlen und einigen Sonderzeichen bestehen. Groß- und Kleinschreibung wird nicht unterschieden.
 - ➔ die **Dateierweiterung** aus maximal **3 Zeichen** nach denselben Regeln wie bei Dateinamen bestehen. Sie gibt normalerweise **Auskunft über den Typ** der gespeicherten Daten.

Beispiele: Die Quelldatei eines C-Programms namens *test.c* ist eine **Textdatei**.
 Der *C-Compiler* und *-Linker* erstellt daraus die ausführbare Datei *test.exe*.
test.exe ist eine **Binärdatei**.
 Textdateien haben oft die Namensweiterung *.txt*.



Dateinamen im Betriebssystem *MS DOS* (2)

- Beim **Betriebssystem *MS DOS*** sind für **Dateinamen** und **Dateierweiterungen** nur die folgenden **Sonderzeichen** erlaubt:

_ ^ \$ ~ ! # % & - { } () @ ' `

- Eigene Dateinamen dürfen folgende **reservierte Namen nicht verwenden**:

CLOCK\$ CON AUX COMn LPTn (n=1..3) NUL PRN

- *MS DOS* verwendet folgende Dateierweiterungen:
 - .EXE** (*executable*: ausführbares Programm)
 - .SYS** (*system*: Dateien mit Systeminformation)
 - .COM** (*command*: Systembefehlsdateien)
 - .BAT** (*batch* = Stapel: Dateien, die Befehle enthalten)



Pfadangaben im Betriebssystem *MS DOS*

- Im Betriebssystem *MS DOS* gelten für **Pfadangaben** die folgenden **zusätzlichen Bedingungen**:
 - ➔ **Groß- und Kleinschreibung** wird nicht unterschieden.
 - ➔ Das einem einzelnen Buchstaben **nachgestellte Sonderzeichen :** kennzeichnet ein logisches oder physikalisches Plattenlaufwerk.
Beispiel: **C:** Laufwerk C:
 - ➔ Das einer Zeichenkette von maximal 8 Buchstaben **vorangestellte Sonderzeichen ** ist ein Trennzeichen zwischen Dateiverzeichnissen und Dateinamen.
Beispiel: **e:\msvclwork\test.c**
Datei **test.c** im Unterverzeichnis **work** des Verzeichnisses **msvc** auf dem Plattenlaufwerk **e:**

27.05.2004



Datei öffnen: ***fopen (..)**

- Die Bibliothek **stdio.h** enthält **Prototypen** von Funktionen **zur Verarbeitung von Datenströmen**.
- Bevor man auf die Daten einer Datei zugreifen kann muss man den **Datenstrom der Datei zuordnen**:
 - ➔ **Öffnen der Datei (open):**
FILE *fopen (char *filename, char *mode)
 - Die Funktion (file open) öffnet eine Datei.
 - ➔ Sie liefert einen **Zeiger auf die Struktur FILE**, die bei den Dateioperationen benutzt wird.
 - ➔ ***filename** ist ein **Zeiger auf den Dateinamen**
 - ➔ ***mode** ist ein **Zeiger auf den Operationsmodus** der Datei.

27.05.2004



Der Rückgabewert von `*fopen (..)`

- Die Struktur `FILE` in der Bibliothek `stdio.h` enthält **Informationen über die Datei.**
 - Ein Zeiger auf die Struktur `FILE` wird vom Anwender vereinbart.
`FILE` wird normalerweise vom Anwender **nicht verändert.**
 - In der Praxis wird der Zeiger auf `FILE`, den `fopen (..)` zurückgibt, zur **Fehlerabfrage** verwendet.
 - Im Fehlerfall wird der **Nullzeiger** (Adresse 0 = NULL) zurückgegeben.
- 💣 Der Zugriff auf eine **nicht vorhandene Datei** führt u.U. zu **schwerwiegenden Fehlern.**

27.05.2004



Die Struktur `FILE`

- Die Struktur `FILE` in der Bibliothek `stdio.h` enthält **Informationen über die Datei.**
- Ein Zeiger auf die Struktur `FILE` muss vom Anwender vereinbart werden.

```
#include <stdio.h>          /* Bibliothek <stdio.h> */
                             /* Struktur für den Dateisteuerblock. */

typedef struct {
    short    level;
    unsigned flags;
    char     fd;
    unsigned char hold;
    short    bsize;
    unsigned char *buffer, *curp;
    unsigned istemp;
    short    token;
} FILE;
```

27.05.2004



Parameter **mode** der Funktion ***fopen (.)** (1)

- Die Parameter **mode** steuert den ➡ **Zugriffsmodus**

mode	Bedeutung
"r"	Textdatei nur zum Lesen (<i>read</i>) öffnen
"w"	Textdatei zum Schreiben (<i>write</i>) erzeugen , gegebenenfalls den alten Inhalt löschen
"a"	Textdatei zum Schreiben (<i>write</i>) öffnen und Text anfügen (<i>append</i>), gegebenenfalls die Datei erzeugen
"r+"	bestehende Textdatei zum Ändern (<i>read & write</i>) öffnen
"w+"	Textdatei zum Ändern (<i>read & write</i>) erzeugen , falls vorhanden den alten Inhalt löschen
"a+"	Textdatei zum Ändern (<i>read & write</i>) erzeugen , falls vorhanden Text anfügen (<i>append</i>)

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K12_Dateiooperationen.ppt, Seite 13

27.05.2004

Parameter **mode** der Funktion ***fopen (.)** (2)

- Die Parameter **mode** steuert den ➡ **Zugriffsmodus**

mode	Bedeutung
"rb"	Binärdatei zum Lesen (<i>read</i>) öffnen
"wb"	Binärdatei zum Schreiben (<i>write</i>) erzeugen , gegebenenfalls den alten Inhalt löschen
"ab"	Binärdatei zum Schreiben (<i>write</i>) öffnen , und Daten anfügen (<i>append</i>), gegebenenfalls die Datei erzeugen
"r+b"	bestehende Binärdatei zum Ändern (<i>read & write</i>) öffnen
"w+b"	Binärdatei zum Ändern (<i>read & write</i>) erzeugen , falls vorhanden, den alten Inhalt löschen
"a+b"	Binärdatei zum Ändern (<i>read & write</i>) öffnen , falls vorhanden, Daten anfügen (<i>append</i>)

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K12_Dateiooperationen.ppt, Seite 14

27.05.2004

Beispiel für `*fopen (..)`

- Das folgende Beispiel öffnet die Datei **TEST.C** im Pfad **H:\MSVC\work** zum Lesen.

Beispiel:

```
#include <stdio.h> /* Bibliothek stdio.h einbinden */
int main (void)    /* Hauptprogramm main (void) */
{
    FILE *p_file ; /* Zeiger auf Struktur FILE vereinbaren*/
    p_file = fopen ("H:\MSVC\work\TEST.C", "r") ;
                    /* Funktion fopen (..) aufrufen*/
    if (!p_file) printf ("Datei nicht gefunden!\n") ; /* Fehler: p_file ==0 */
    else printf ("Datei gefunden.\n") ; /* kein Fehler: p_file !=0 */
    return 0 ; /* Rücksprung ins Betriebssystem */
}
```

27.05.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K12_Dateioperationen.ppt, Seite 15



Aufgabe: Datei öffnen

1. Warum enthält der Pfadname der Datei im C-Quelltext im Unterschied zu den *MS-DOS*-Angaben statt des `\` den `\\` ?
2. Modifizieren Sie das vorherige Beispiel für das Öffnen einer Textdatei, indem Sie den Namen einer Datei einlesen, eine Binärdatei öffnen und als Parameter für `fopen (..)` - soweit möglich -
 - 2.1 Felder
 - 2.2 Zeiger auf Strings
 verwenden !

27.05.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K12_Dateioperationen.ppt, Seite 16

✓ Aufgabe: Datei öffnen (1)

1. Pfadangabe mit `\\` :

Der Backslash `\` leitet in Zeichenketten (*strings*) in C-Quelldateien immer eine Ersatzdarstellung (*escape sequence*) ein. Der Dateiname und Pfad wird als Zeichenkette (*string*) übergeben.

Deshalb muss für den Backslash `\` in Strings in C-Quelldateien immer die Ersatzdarstellung `\\` angegeben werden.

27.05.2004



✓ Aufgabe: Datei öffnen (2)

2.1 Binärdatei mit Feldparametern:

```
#include <stdio.h>          /* Bibliothek stdio.h einbinden */
int main (void)             /* Hauptprogramm main (void) */
{
    FILE *p_file ;          /* Zeiger auf Struktur FILE vereinbaren */
    char name [67], mode [ ] = "rb" ; /* max. MS DOS Pfadlänge = 66 */
    printf ("Dateinamen mit Pfad der Binaerdatei eingeben: ") ;
    scanf ("%s", name) ;     /* Bei der Eingabe kein \\ !! */
    p_file = fopen ( name, mode) ; /* Funktion fopen (..) aufrufen */
    if (!p_file) printf ("Datei %s nicht gefunden!\n", name) ; /* Fehler: */
    else printf ("Datei %s gefunden!\n", name) ; /* kein Fehler: */
    return 0 ;               /* Rücksprung ins Betriebssystem */
}
```

27.05.2004



✓ Aufgabe: Datei öffnen (3)

2.2 Binärdatei mit Stringparametern:

- ⚠ Achtung: Hier wäre die Definition `char *name` falsch:
`name` ist kein konstanter String!
 Die Länge des Strings ist dem Compiler unbekannt.

```
#include <stdio.h> /* Bibliothek stdio.h einbinden */

int main (void) /* Hauptprogramm main (void) */
{
    FILE *p_file ; /* Zeiger auf Struktur FILE vereinbaren*/
    char name [67], *mode = "rb" ; /* max. MS DOS Pfadlänge = 66 */
    printf ("Dateinamen der Binaerdatei eingeben: ") ;
    scanf ("%s", name) ; /* Bei der Eingabe kein \! */
    p_file = fopen ( name, mode) ; /* Funktion fopen (..) aufrufen*/
    if (!p_file) printf ("Datei %s nicht gefunden!\n", name) ; /* Fehler: */
    else printf ("Datei %s gefunden!\n", name) ; /* kein Fehler: */
    return 0 ; /* Rücksprung ins Betriebssystem */
}
```

27.05.2004

Hochschule für angewandte Wissenschaften Hamburg
 Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
 K12_Dateioperationen.ppt, Seite 19

Datei schließen: `fclose` (..)

- Die Bibliothek `stdio.h` stellt ebenfalls eine Funktionen zur Verfügung, um die

➡ **Verbindung zu einer Datei** zu trennen:

```
int fclose ( FILE *p_file ) ;
```

Die Funktion `fclose` (..) verwendet ebenfalls einen Zeiger auf die Struktur `FILE`.

Beim Schließen einer Datei werden **alle Puffer geleert**:
 Noch nicht gespeicherte Daten werden geschrieben
 und noch nicht gelesene Eingabedaten gelöscht

Auch hier ist meistens die **Fehlererkennung wichtig**:
 Bei **fehlerhafter Ausführung** liefert die Funktion den
Wert EOF zurück, **ansonsten** wird der **Wert NULL ('0')**
 zurückgegeben.

27.05.2004

Hochschule für angewandte Wissenschaften Hamburg
 Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
 K12_Dateioperationen.ppt, Seite 20

Funktionen für Textdateien: **fscanf (..)**

- Die Bibliothek **stdio.h** stellt ebenfalls eine Funktionen zur Verfügung, um

➡ **Eingabefelder formatiert aus eine Datei zu lesen:**
`int fscanf (FILE *p_file, char *format, char *in1, char *in2, ..);`

fscanf (..) benutzt den Zeiger auf die Struktur **FILE**.

Bei **fehlerhafter Ausführung** liefert die Funktion den Wert **EOF** zurück, ansonsten werden **Zeiger auf die gelesenen Eingabefelder**, sowie **deren Anzahl** als **Funktionsrückgabewert** zurückgegeben.

char *format: enthält normalerweise **Umwandlungsangaben** zur Interpretation der gelesenen Zeichenketten.

Die Strings werden unter **char *in1, char *in2, ...** abgelegt.

fscanf (..) verhält sich sinngemäß wie **scanf (..)**.



Beispiel für das Lesen von Textdateien (1)

- Das folgende Programm ist ein Beispiel

➡ zum **Einlesen von Stringzeilen aus einer Textdatei**
 ➡ und zur **Ausgabe auf dem Bildschirm**.

```
/* Datei TEST21.C */
#include <stdio.h> /* Bibliothek stdio.h einbinden */
int main (void)    /* Hauptprogramm main (void) */
{
    FILE *p_file;      /* Zeiger auf Struktur FILE vereinbaren*/
    char name [67], *mode = "r"; /* MS-DOS Pfad: max. 66 Zeichen */
    char zeile [81];    /* max. Zeilenlänge = 80 Zeichen */
    printf ("Dateinamen der Textdatei eingeben: "); scanf ("%s", name);
    /* Pfadeingabe hier ohne doppelte Backslashes!! */
    p_file = fopen ( name, mode); /* Funktion fopen (..) aufrufen*/
    ...
}
```



Beispiel für das Lesen von Textdateien (2)

```

***
if (!p_file)
{
    /* Fehler: */
    printf ("Datei %s nicht gefunden!\n", name);
    return 1;      /* Fehlerrücksprung ins Betriebssystem */
}
else printf ("\n\nAusgabe der Datei %s:\n\n", name); /* kein Fehler: */
while (1)        /* Endlosschleife */
{
    ➡ if (fscanf (p_file, "%80[^\n]", zeile) == EOF) break ; /* Datei bis zum
        Zeilenende (\n) lesen, max. 80 Zeichen; EOF ? : Zeilenabbruch */
    ➡ printf (zeile);      /* Zeile ausgeben */
    ➡ if (fscanf (p_file, "%1[\n]", zeile) == EOF) break ; /* Zeilenumbruch
        (\n) lesen, max. 1 Zeichen; EOF ? : Schleifenabbruch */
    ➡ printf (zeile);      /* Zeilenumbruch ausgeben*/
}
fclose (p_file);  /* Datei schließen */
return 0;         /* Rücksprung ins Betriebssystem */
}

```

27/05/2004

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K12_Dateioperationen.ppt, Seite 23

Beispiel für das Lesen von Textdateien (3)

- **test21.c**
erzeugt die
neben-
stehende
Bildschirm
ausgabe:

27/05/2004

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K12_Dateioperationen.ppt, Seite 24

Funktionen für Textdateien: **fprintf (..)**

- Die Bibliothek **stdio.h** stellt ebenfalls eine Funktionen zur Verfügung, um

➔ **Eingabefelder formatiert in eine Datei** zu schreiben:

```
int fprintf ( FILE *p_file, char *format, char *in1, char *in2, .. ) ;
```

fprintf (..) verwendet ebenfalls einen Zeiger auf **FILE**.

Bei **fehlerhafter Ausführung** wird als Funktionsrückgabewert **EOF** geliefert, **ansonsten** wird die **Anzahl der geschriebenen Bytes** zurückgegeben.

char *format: enthält normalerweise einen **Formatstring** zur Interpretation der Zeichenketten.

Die Strings werden unter **char *in1, char *in2, ...** abgelegt.

fprintf (..) verhält sich sinngemäß wie **printf (..)**.



Aufgabe: Textdatei mit **fprintf (..) schreiben**

1. Eine Textdatei soll in eine neue Datei kopiert werden.

1.1 Verwenden Sie **fscanf (..)** und **fprintf (..)**, um die Kopie der Textdatei zu erstellen!

Stellen Sie sicher, dass eine eventuelle schon vorhandene Datei gleichen Namens nicht überschrieben wird!

1.2 Überprüfen Sie die Lesbarkeit der Datei mit Hilfe eines Texteditors (z.B. **MS NOTEPAD**)



✓ Aufgabe: Textdatei mit **fprintf** (..) schreiben (1)

1.1 C-Quelltext (1):

```
/* Datei: test 22.c */

#include <stdio.h> /* Bibliothek stdio.h einbinden */

int main (void) /* Hauptprogramm main (void) */
{
    FILE *p_file_q, *p_file_z; /* Zeiger auf Struktur FILE vereinbaren */
    char name_q[67] = "I:\\MSVC\\TEST22.c", *mode_q = "r";
                                /* MS-DOS Pfad ist maximal 66 Zeichen lang */
    char name_z[67] = "h:\\msvc\\test22_n.c", *mode_z = "w";
    char zeile [81], jn ; /* max Zeilenlaenge = 80 Zeichen */
    printf ("Dateinamen der Quelltextdatei eingeben: ");
    fflush(stdin); /* Tastaturpuffer löschen */
    scanf ("%s", name_q); /* Quelldateinamen einlesen */
    p_file_q = fopen ( name_q, mode_q); /* Funktion fopen(..) aufrufen */

    ***
}
```

27.05.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K12_Dateiooperationen.ppt, Seite 27

✓ Aufgabe: Textdatei mit **fprintf** (..) schreiben (2)

1.1 C-Quelltext (2):

```
***
if (!p_file_q) /* Fehler : Quelldatei nicht vorhanden */
{
    printf ("Datei %s nicht gefunden!\n", name_q); /* Fehler: */
    return 1; /* Fehlerruecksprung ins Betriebssystem */
}
printf ("Dateinamen der Zieltextdatei eingeben: ");
fflush(stdin); /* Tastaturpuffer loeschen */
scanf ("%s", name_z); /* Zieldateinamen einlesen */
while (p_file_z = fopen ( name_z, mode_q)) /* existiert Zieldatei? */
{
    printf ("\n Die Datei %s existiert bereits.\n"
            "Ueberschreiben ja, nein, Abbruch=andere Taste (j,n): ", name_z );
    fflush(stdin); /* Tastaturpuffer loeschen */
    scanf ("%c",&jn); /* Schalter jn einlesen */

    ***
}
```

27.05.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K12_Dateiooperationen.ppt, Seite 28

✓ Aufgabe: Textdatei mit **fprintf (..)** schreiben (3)

1.1 C-Quelltext (3):

```

***
switch (jn)                                /* Datei ueberschreiben j/n/Abbruch?
*/
{
    case 'j': break;                        /* jn == j: Datei ueberschreiben */
    case 'n':                               /* jn == n: neuer Dateiname */
        {
            printf ("\nneuen Dateinamen eingeben: ");
            scanf ("%s", name_z);           /* neuen Zieldateinamen einlesen */
            break;                          /* switch abbrechen */
        }
    default :                               /* jn == andere Taste: Abbruch */
        {
            printf ("\n Programmabbruch!");
            return 1;                       /* Fehlerruecksprung ins Betriebssystem */
        }
}

```

27.05.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K12_Dateioperationen.ppt, Seite 29

✓ Aufgabe: Textdatei mit **fprintf (..)** schreiben (4)

1.1 C-Quelltext (4):

```

***
    if ( jn == 'j') break;
}
p_file_z = fopen ( name_z, mode_z); /* Datei zum Schreiben oeffnen */
while (1)
{
    if (fscanf (p_file_q, "%80[^\n]", zeile) == EOF) break ;
        /* Datei bis zum Zeilenende lesen ; EOF?: = Fehler */
    if (fprintf (p_file_z, "%s", zeile) == EOF)
        /* Zeile schreiben; EOF?: = Fehler */
    {
        printf ("Fehler beim Kopieren in die Zieldatei!");
        return 1; /* Fehlerruecksprung ins Betriebssystem */
    }
    if (fscanf (p_file_q, "%1[\n]", zeile) == EOF) break ;
        /* Zeilenumbruch (\n) lesen; EOF?: = Fehler */
}

```

27.05.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K12_Dateioperationen.ppt, Seite 30

✓ Aufgabe: Textdatei mit **fprintf** (..) schreiben (5)

1.1 C-Quelltext (5):

```

***
    if (fprintf (p_file_z, "%s", zeile) == EOF)
        /* Zeilenumbruch schreiben; EOF?: = Fehler */
        {
            printf ("Fehler beim Kopieren in die Zielfdatei!");
            return 1;          /* Fehlerruecksprung ins Betriebssystem */
        }
    }
    fclose (p_file_q);        /* Quelldatei schließen */
    fclose (p_file_z);        /* Zielfdatei schließen */
    printf("\n Das Duplikat %s der Datei %s\n"
           "wurde erfolgreich erstellt.\n\n", name_z, name_q);
    return 0;                 /* Ruecksprung ins Betriebssystem */
}

```

27.05.2004



✓ Aufgabe: Textdatei mit **fprintf** (..) schreiben (6)

1.1 Bildschirmausgabe:

1.2 Lesbarkeit: Die Datei ist mit einem Texteditor lesbar.

27.05.2004



Funktionen für Textdateien: `fgetc (..)`, `fputc (..)`

- Die Funktionen `fscanf (..)` und `fprintf (..)` sind häufig verwendete Funktionen, weil sie
 - ➔ **unterschiedliche Formate** bei Dateizugriffen erlauben
 - ➔ `scanf (..)` und `printf (..)` **ähnlich sind**.
- Wegen der aufwendigen Interpretation und Umsetzung der Formatinformation sind sie aber leider **zeitaufwendig** und **uneffektiv**.
- Deshalb ist es u.U. ratsam, auf **Zeichenketten direkt** mit Hilfe der Funktionen `fgetc (..)` und `fputc (..)` zuzugreifen.
`fgetc (..)` und `fputc (..)` funktionieren ähnlich wie `getc (..)` und `putc (..)`.

27/05/2004



Lesen und Schreiben von Binärdateien (1)

- Beim Zugriff auf **Binärdateien**
 - ➔ erfolgt der **Zugriff vergleichsweise schnell**
 - ➔ sind **keine Umcodierungen** notwendig.
- Die Bibliothek `stdio.h` stellt für Binärdateien folgende Funktionen zur Verfügung, um
 - ➔ **Daten unformatiert** aus einer Datei zu **lesen**:
`unsigned int fread ( void *ptr, unsigned int size, unsigned int n, FILE *p_file ) ;`
 - ➔ **Daten unformatiert** in eine Datei zu **schreiben**:
`unsigned int fwrite ( void *ptr, unsigned int size, unsigned int n, FILE *p_file ) ;`

27/05/2004



Lesen und Schreiben von Binärdateien (2)

- Die **Parameter** von **fread (..)** bzw. **fwrite (..)** :
(void *ptr, unsigned int size, unsigned int n, FILE *p_file)
 bei **fread (..)** bzw. **fwrite (..)** haben folgende Bedeutung:
 - ➔ **ptr** Zeiger auf den Speicherbereich, in den die Daten geschrieben bzw. aus dem sie gelesen werden
 - ➔ **size** Länge der zu lesenden bzw. schreibenden Datenblöcke in Bytes
 - ➔ **n** Anzahl der zu lesenden bzw. schreibenden Datenblöcke in Bytes
 - ➔ **pfile** Zeiger auf die Struktur **FILE**

Jedem Zeiger auf void - auch **ptr** – darf beim Aufruf ein **beliebiger anderer Datentyp** zugeordnet werden.



Funktionen für den Zugriff auf Binärdateien (1)

- C und die Bibliothek **stdio.h** stellen weitere nützliche Funktionen zum Zugriff auf Binärdateien zur Verfügung:
 - ➔ **sizeof (Typ)**: Diese Schlüsselfunktion in C liefert den von **Typ benötigten Speicherplatz in Bytes** als Funktionsrückgabewert.
Typ darf ein **beliebiger Variablen- oder Strukturtyp** sein.
 - ➔ **rewind (File *p_file)**: Die Funktion in **stdio.h** setzt den **Zeiger auf den Anfang** der über **p_file** angesprochenen Datei **zurück**.
 Die Funktion kann auch auf Textdateien angewendet werden.



Beispiel: Schreiben von Binärdateien (1)

```

/* Datei: test23.c */
#include <stdio.h> /* Bibliothek stdio.h einbinden */

typedef struct      /* Struktur DATUM vereinbaren*/
{
    int tag ; char *monat ; int jahr ;
} DATUM;

int main (void)     /* Hauptprogramm main (void) */
{
    FILE *p_file;    /* Zeiger auf Struktur FILE vereinbaren*/
    DATUM dat1 = {20, "Oktober", 2000 }, dat2;
    /* Strukturen dat1, dat2 vereinbaren, dat1 initialisieren*/
    printf (" Eingabedatum: %d.%s %d \n\n", dat1.tag, dat1.monat,
            dat1.jahr) ; /* Eingabedatum ausgeben */
    p_file = fopen ( "i:\\msvc\\test23.bin", "w+b") ; /* fopen (..) aufrufen*/
    /* test23.bin muss existieren!! */
    ...

```

27.05.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K12_Dateiooperationen.ppt, Seite 37

Beispiel: Schreiben von Binärdateien (2)

```

...
if (!p_file)        /* Fehlerabfrage bei fopen (..) */
{
    printf ("Datei nicht vorhanden!"); /* Fehlermeldung */
    return 1;        /* Fehlerrücksprung ins Betriebssystem */
}
fwrite (&dat1, sizeof (dat1), 1, p_file);
/* Struktur dat1 als Binärdatei schreiben */
rewind (p_file);    /* Dateizeiger auf den Anfang zuruecksetzen */
fread (&dat2, sizeof (dat2), 1, p_file);
/* Struktur dat1 aus Binärdatei zuruecklesen */
printf ("Datum: %d.%s %d\n", dat2.tag, dat2.monat, dat2.jahr);
/* zurueckgelesenes Datum ausgeben */
return 0;           /* Ruecksprung ins Betriebssystem */
}

```

27.05.2004

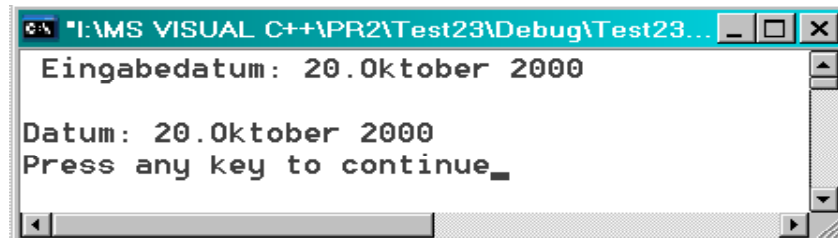
Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K12_Dateiooperationen.ppt, Seite 38

Beispiel: Schreiben von Binärdateien (3)

- Bildschirmausgabe von **test23.exe**



27.05.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K12_Dateioperationen.ppt, Seite 39

Sequentieller und wahlfreier Zugriff

- Bisher haben wir **nacheinander** auf die Daten einer Datei zugegriffen.
Man nennt diese Zugriffsart:
➡ **sequentieller Zugriff** (*sequential access*)
- Der **sequentielle Zugriff** ist **zeitaufwendig**, wenn man beispielsweise nur auf Daten am Ende der Datei zugreifen möchte.
 - Effizienter ist in diesem Fall der
➡ **wahlfreie Zugriff** (*random access*).

27.05.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K12_Dateioperationen.ppt, Seite 40

wahlfreier Zugriff: `fseek (..)` (1)

- Die Bibliothek `stdio.h` beinhaltet Funktionen für

➔ den **wahlfreien Dateizugriff** (*random access*):

```
int fseek ( FILE *p_file, long offset, int org ) ;
```

`fseek (..)` verwendet ebenfalls den Zeiger auf `FILE`.

Bei **fehlerfreier Ausführung** wird **0** als Funktionsrückgabewert geliefert, **ansonsten** ein **Wert $\neq 0$** .

➔ **offset** `fseek (..)` **setzt** den zu `p_file` gehörenden **Datenzeiger auf eine neue Position**, die **offset Bytes** von der durch `org` gegebenen Dateiposition entfernt ist.
Im Textmodus sollte **offset = 0** sein oder ein Wert, der von `ftell (..)` geliefert wurde.

27.05.2004



wahlfreier Zugriff: `fseek (..)` (2)

```
int fseek ( FILE *p_file, long offset, int org ) ;
```

➔ **org** Die Art der Positionierung des Zeigers wird mit Hilfe von 3 Konstanten festgelegt, die in `stdio.h` definiert sind:

org = SEEK_SET relativ zum Dateianfang

org = SEEK_CUR relativ zur vorherigen Position

org = SEEK_END relativ zum Dateiende

Beispiel: `#include <stdio.h>` */* Einbinden von `stdio.h` */*

```
...  
FILE *p_file ; /* Definition von p_file */  
...
```

```
ok = fseek (p_file, 100L, SEEK_SET) ;  
/* Setze Dateizeiger in der Struktur FILE 100 Bytes  
hinter den Dateianfang */
```

27.05.2004



wahlfreier Zugriff: `ftell (..)`

- Die Bibliothek **`stdio.h`** beinhaltet Funktionen für
 ➔ **wahlfreie Dateizugriffe** (*random access*):
`int ftell ( FILE *p_file ) ;`
- **`ftell (..)`** verwendet ebenfalls den Zeiger auf **`FILE`**.
 - Bei **fehlerfreier Ausführung** wird **die aktuelle Zeigerposition in Byte** als Funktionsrückgabewert geliefert,
ansonsten der Wert -1L.
- **`ftell (..)`** kann auf Binär- und Textdateien angewendet werden.

27.05.2004

**Beispiel für wahlfreien Zugriff: `ftell (..)` (1)**`int ftell ( FILE *p_file ) ;`

liefert bei wahlfreiem Dateizugriff auf Text- oder Binärdateien die **aktuelle Zeigerposition**.

Beispiel:

```

/* Datei: Test23_1.c */
#include <stdio.h>          /* Einbinden von stdio.h */

int main (void)            /* Hauptprogramm main (void) */
{
    FILE *p_file ;         /* Definition von p_file */
    p_file = fopen ( "i:\\msvc\\datei.txt", "w+" ) ; /* datei.txt oeffnen */
    fprintf ( p_file , "Dies ist ein Test." ) ;
    /* Dies ist ein Test. in DATEI.TXT schreiben */
    printf ( "Der Dateizeiger steht bei Byte %ld.\n", ftell (p_file) ) ;
    fclose (p_file) ;      /* Zeigerposition ausgeben; Datei schliessen */
    return 0 ;             /* Rücksprung ins Betriebssystem */
}

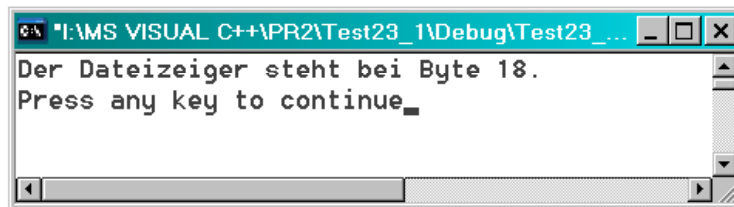
```

27.05.2004



Beispiel für wahlfreien Zugriff: `ftell (..)` (2)

- Bildschirmausgabe von `test23_1.exe`



27.05.2004

