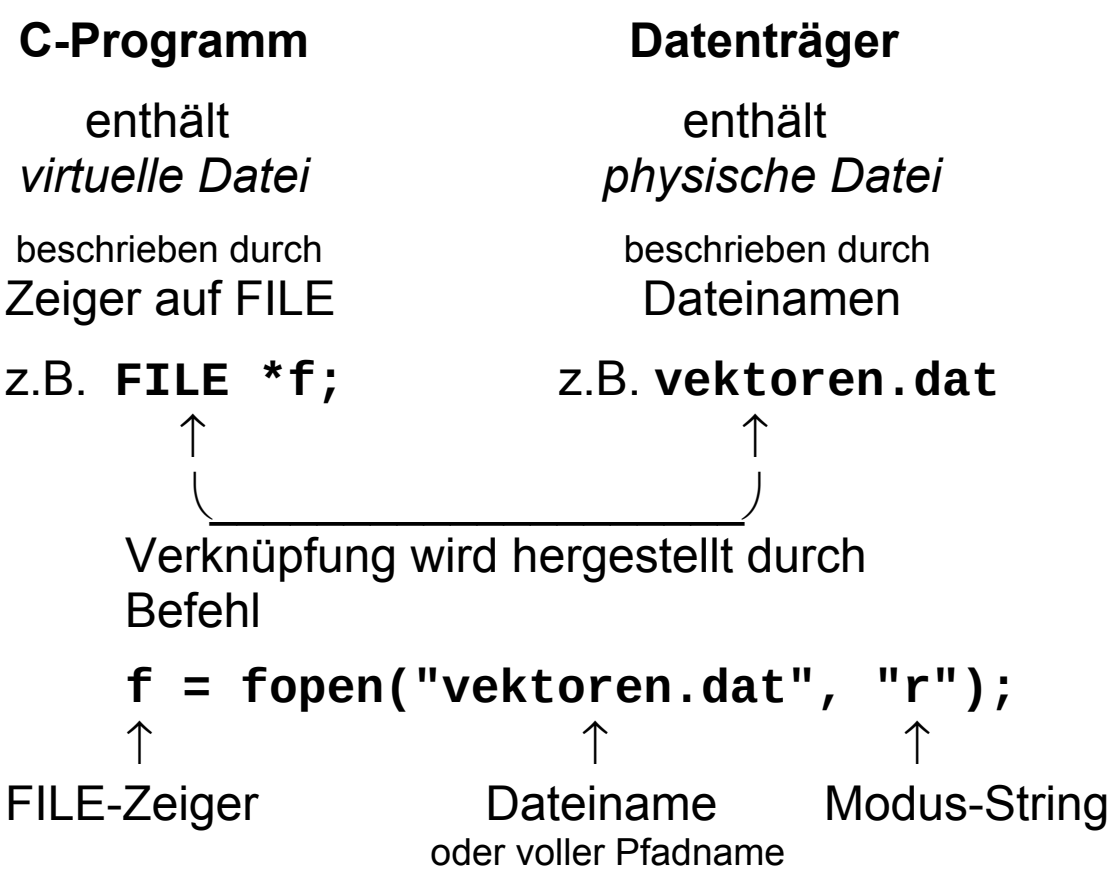
2. 7. Arbeiten mit Dateien



Wirkung:

1. Versuch, eine Datei mit dem angegebenen Namen zu öffnen
   - wenn nur Dateiname: im aktuellen Verzeichnis
   - wenn voller Pfadname: dort
2. gibt Zeiger auf Datei zurück, falls erfolgreich; sonst NULL

Wichtig: Bei Verwendung dieses Befehls testen, ob NULL zurückgegeben wurde (siehe Kap. 2.4.8, Fehlerbehandlung)!

Bedeutung des Modus-Strings:

`"r"` öffnet zum Lesen (*read*);
falls Datei nicht existiert: NULL

`"w"` öffnet zum Schreiben (*write*),
falls Datei schon existiert, wird Inhalt gelöscht
sonst wird Datei neu angelegt

`"a"` öffnet zum Schreiben, neuer Inhalt wird an alten angehängt (*append*)

`"r+"` öffnet zum Lesen *und* Schreiben;
Datei muss existieren, sonst NULL

`"w+"` öffnet zum Lesen und Schreiben;
alter Inhalt wird gelöscht

`"a+"` öffnet zum Lesen und Schreiben;
neuer Inhalt wird an alten angehängt.

2 Arten von Dateien:

- Textdatei: enthält Folge von Zeichen, Lesen und Schreiben textorientiert
  Befehle **`fprintf`**, **`fscanf`**, **`fgets`** (stdio.h)
- Binärdatei: Interpretation als Folge von Bytes, Lesen und Schreiben blockorientiert (ohne auf Inhalt zu achten)
  Befehle **`fwrite`**, **`fread`** (stdio.h)

`fprintf(FILE *f, ...)` (weitere Argumente wie bei **`printf`**),
`fscanf(FILE *f, ...)` (weitere Argumente wie bei **`scanf`**):
siehe Kap. 2.4.6.

Befehl zum *Schreiben in Binärdatei*: **`fwrite`**

```
size_t fwrite(const void *ptr, size_t size, size_t n, FILE *f)
```

Rückgabewert: Anzahl geschriebener Blöcke (spezieller Typ **`size_t`**, konvertierbar in **`int`**)	Zeiger auf beliebigen Typ!	Größe eines Blocks	Anzahl zu schreibender Blöcke	Dateizeiger

Öffnen von **`f`** vorher mit Modus **`"wb"`**.

Beispiel:

```
long x = 42;
file *f;
f = fopen("binaer.dat", "wb");
fwrite(&x, sizeof(long), 1, f);
```

Lesen aus Binärdatei: **`fread`**

```
size_t fread(const void *ptr, size_t size, size_t n, FILE *f)
```

analog aufgebaut.

Öffnen von **`f`** vorher mit Modus **`"rb"`**.

Bei Abschluss der Datei- Ein-/Ausgabe (Text oder binär):

```
fclose(f);
```

Grund: Ein-/Ausgabe ist gepuffert.

Befehl **`fflush(f);`** erzwingt Leeren des Puffers, Datei bleibt aber offen.

- bei Ausgabedateien: physikal. Schreiben des Pufferinhalts wird erzwungen
- bei Eingabedateien: Pufferinhalt wird gelöscht.

Befehle zur Navigation in Dateien:

`long ftell(FILE *f)` liefert momentane Position innerh. von **`f`** (relativ zum Dateianfang, Zählung in Bytes)

`int fseek(FILE *f, long offset, int m)` setzt Position innerh. **`f`** für folgende Lese- und/oder Schreib-operationen

vordefinierte Konstanten für **`m`** regeln den Modus:

`m` = **`SEEK_SET`** : offset wird relativ zum Datei-anfang interpretiert (erstes Byte = 0)

`m` = **`SEEK_CUR`** : relativ zur vorherigen Position

`m` = **`SEEK_END`** : relativ zum Dateiende

Rückgabewert 0 bei fehlerfreier Ausführung
(dies gilt auch für die folgenden Funktionen)

Arbeiten mit Verzeichnissen:

```
#include <dir.h>
```

`int chdir(const char *path)`
setzt das aktuelle Verzeichnis neu

`int mkdir(const char *path)`
erzeugt ein neues Verzeichnis

`int rmdir(const char *path)`
löscht ein Verzeichnis