File Pointer

>> Die Möglichkeit, Daten aus Dateien zu lesen und in Dateien zu schreiben, ist das wichtigste Mittel zur Speicherung **persistenter** Daten – Daten, die nicht verschwinden, wenn Ihr Programm beendet wird.

Die Abstraktion von Dateien, die C bereitstellt, wird in einer Datenstruktur namens **FILE** implementiert.

Fast immer arbeiten wir mit Pointern auf diese Dateien, **FILE***.

File Pointer

Alle wichtigen Funktionen zur Verarbeitung von Dateien befinden sich in **stdio.h**.

Diese Funktionen akzeptieren FILE* als einen ihrer Parameter – außer der Funktion **fopen()**, die verwendet wird, um überhaupt erst einen File Pointer zu erhalten.

Die häufigsten Ein-/Ausgabe-Funktionen für Dateien sind:

fopen() fclose()
fgetc() fputc()
fread() fwrite()

fopen Öffnet eine Datei und gibt einen File Pointer darauf zurück.

Überprüfen Sie immer den Rückgabewert, um sicherzustellen, dass Sie nicht NULL zurückbekommen.

```
FILE* ptr = fopen(<filename>, <operation>);
```

```
FILE* ptr1 = fopen("file1.txt", "r");
FILE* ptr2 = fopen("file2.txt", "w");
FILE* ptr3 = fopen("file3.txt", “a");
```

fclose Schließt die Datei, auf die der übergebene File Pointer zeigt.

```
fclose(<file pointer>);
```

fgetc Liest und gibt das nächste Zeichen aus der Datei zurück.

Hinweis: Der File Pointer muss im Modus "r" für Lesen geöffnet sein, sonst gibt es einen Fehler.

```
char ch = fgetc(<file pointer>);
```

Mit fgetc() können wir, in einer Schleife verwendet, alle Zeichen aus einer Datei lesen und auf dem Bildschirm ausgeben.

Dies entspricht dem Linux-Befehl "cat".

```
char ch;
while((ch = fgetc(ptr)) != EOF)
    printf("%c", ch);
```

fputc Schreibt oder hängt das angegebene Zeichen an die Datei an.

Hinweis: Der File Pointer muss im Modus „w“ für Schreiben oder „a“ für Anhängen geöffnet sein, sonst gibt es einen Fehler.

```
fputc(<character>, <file pointer>);
```

```
fputc('A', ptr2);
```

```
fputc('!', ptr3);
```

```
char ch;
while((ch = fgetc(ptr)) != EOF)
    fputc(ch, ptr2);
```

fread Liest <qty> Einheiten der Größe <size> aus der Datei und speichert sie im Puffer (meist ein Array), auf den <buffer> zeigt.

Hinweis: Der File Pointer muss im Modus „r“ geöffnet sein.

```
fread(<buffer>, <size>, <qty>, <file pointer>);
```

```
int arr[10];
fread(arr, sizeof(int), 10, ptr);

double* arr2 = malloc(sizeof(double) * 80);
fread(arr2, sizeof(double), 80, ptr);

char c;
fread(&c, sizeof(char), 1, ptr);
```

fwrite Schreibt <qty> Einheiten der Größe <size> in die Datei, indem sie aus dem Puffer (meist ein Array) gelesen werden.

Hinweis: Der File Pointer muss im Modus „w“ oder „a“geöffnet sein.

```
fwrite(<buffer>, <size>, <qty>, <file pointer>);
```

```
int arr[10];
fwrite(arr, sizeof(int), 10, ptr);
```

```
double* arr2 = malloc(sizeof(double) * 80);
// ... arr2 befüllen ...
fwrite(arr2, sizeof(double), 80, ptr);
```

Weitere nützliche Funktionen aus stdio.h

Funktion	Beschreibung
fgets	Liest einen kompletten String aus einer Datei.
fputs	Schreibt einen kompletten String in eine Datei.
fprintf	Schreibt einen formatierten String in eine Datei.
fseek	Ermöglicht Vor- und Zurückspulen innerhalb einer Datei.
ftell	Gibt die aktuelle (Byte-)Position in der Datei an.
feof	Zeigt an, ob das Dateiende erreicht wurde.
ferror	Zeigt an, ob ein Fehler aufgetreten ist.

EXTRAS IN 3 MINUTEN

FRAGEN – ANTWORTEN – RÄTSEL
UND KURZE ZUSAMMENFASSUNG

Was fehlt hier?

```c
FILE* fp = fopen("daten.txt", "r");
char c;
while((c = fgetc(fp)) != EOF) {
    printf("%c", c);
}
```

Warum ist das effizienter als einzelne Zeichen zu lesen?

```c
FILE* fp = fopen("large_data.txt", "r");
if (fp == NULL) return 1;


char buffer[1024];
size_t bytes_read;
while ((bytes_read = fread(buffer, 1, sizeof(buffer), fp)) > 0) {
    // Verarbeite buffer[0] bis buffer[bytes_read-1]
    process_data(buffer, bytes_read);
}


fclose(fp);
```

File Pointer
ermöglichen persistente
Datenspeicherung.

Buffer-Größe dem
Anwendungsfall anpassen.

Wichtigste Funktionen:

fopen(), fclose(),
fgetc(), fputc(),
fread(), fwrite()

Auf NULL-Pointer und
korrektes Schließen achten