HAW HAMBURG

Programmiertechniken 1

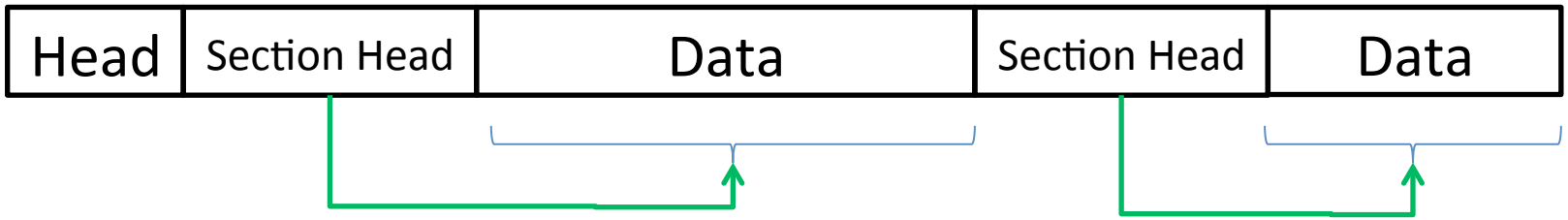
Lesen und Schreiben von Dateien

Dateien

- Dateien (File) sind aus Sicht des Computers eine Folge von Bytes, die nacheinander gelesen oder geschrieben werden (vergleichbar einem Tonband).
- Typischerweise sind Dateien persistent.
- Dateien werden in einem Dateisystem (File System) organisiert und über einen Pfad + Name identifiziert.
  Beispiele:
  c:\temp\auftrag.txt
  /var/spool/messages.log

Binärformate

- Die Position der Bytes in der Datei definiert die Interpretation



Header hat bekannte Größe und Struktur
Section Head hat bekannte Größe und Struktur
Data ist variable in der Länge (Elementeanzahl, bekannte Datentypen), Länge in Section Head gespeichert.
Daten können eigenen interne Struktur haben
…
Es kann auch eine ganz andere interne Strukturierung vorliegen.

Text-Datein

- Text-Dateien enthalten Byte-Folgen, die als ASCII-Zeichen (oder andere Codierung) sichtbar gemacht werden und für den Menschen „lesbar“ sind.

Byte-Folge:

```
5345 5031 202d 2050
7261 6b74 696b 756d
7374 6572 6d69 6e20
310a 5365 6872 2067
6565 6872 7465 2053
7475 6469 6572 656e
6465 2c0a 6265 6920
```

ASCII-Encoded:

```
SEP1 - Praktikumstermin 1
Sehr geehrte Studierende,
bei der stichprobenartigen
Durchsicht Ihrer Abgaben
ist mir noch folgendes
aufgefallen:
```

Trennsymbole

- Trennsymbole in den Dateien ermöglichen einfach Strukturierungen, die von der Position unabhängig sind.
- Beispiel: CSV – Comma-Separated Values
  Datensatz: Zeilenumbruch
  Datenfelder: Semikolon, Doppelpunkt oder Tabulator

```
Stunde;Montag;Dienstag;Mittwoch;Donnerstag;Freitag
1;Mathe;Deutsch;Englisch;Mathe;Kunst
2;Sport;Französisch;Geschichte;Sport;Geschichte
3;Sport;"Religion ev;kath";Kunst;;Kunst
```

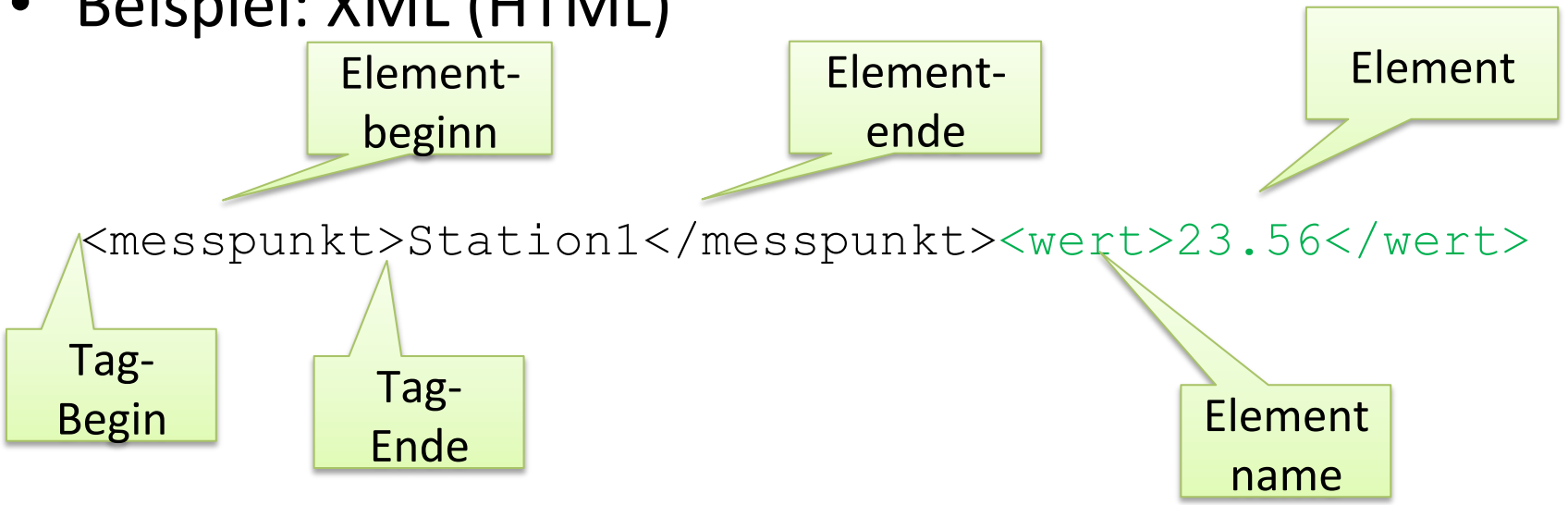
Ini-Dateien

- Bei Ini-Dateien werden Werte als Schlüssel-Werte-Paare (Key-Value-Pairs) in Sektionen abgelegt. Wird oft für Konfigurationen eingesetzt.
- Die Paare sind oft in Sektionen unterteilt.

```
[Tables]
Loadable=true
MaxRange=30
[GUI]
Language=EN
```

Markup-Datein

- Zeichenfolgen klammern Datenfelder (Elemente).
  Die Zeichenfolgen unterliegen wieder einem Schema.
- Beispiel: XML (HTML)



- Hierarchische Schachtelung ist erlaubt.

Dateibearbeitung

- Schreiben:
  - Daten müssen entsprechend der definierten Struktur als Datenstrom geschrieben werden.
  - Alte Daten werden beim Schreiben überschrieben.
- Lese:
  - Datenstrom wird gelesen und muss entsprechend der definierten Struktur zerlegt (parsen) werden.
- Navigation:
  - Datenstrom wird Byte-weise oder Block-weise verarbeitet.
  - Vor- und zurückspulen ist beim Lesen möglich.
  - Zurückspulen ist beim Schreiben möglich

FILE-Pointer

Eine Datei muss vor dem Zugriff geöffnet werden. Das Betriebssystem legt intern eine Verwaltungsstruktur an. Auf die Verwaltungsstruktur wird über den FILE-Pointer zugegriffen:

```
FILE *textfile = NULL;
```

Eine geöffnete Datei wird über die Variable innerhalb des Programms identifiziert. Der Inhalt wird nicht direkt verändert.

Ist der Wert NULL, so wurde die Datei nicht erfolgreich geöffnet.

Dateizugriff in C - Öffnen

Öffnen der Datei:

```
FILE * fopen ( const char * filename, const char * mode );
```

- Modes: r – read, w – write, a – append, …
  Beispiel:

```
FILE *outfile = fopen ( "c:\\tmp\\test.text", "a");
```

- Ist der Rückgabewert NULL, so wurde die Datei nicht erfolgreich geöffnet.

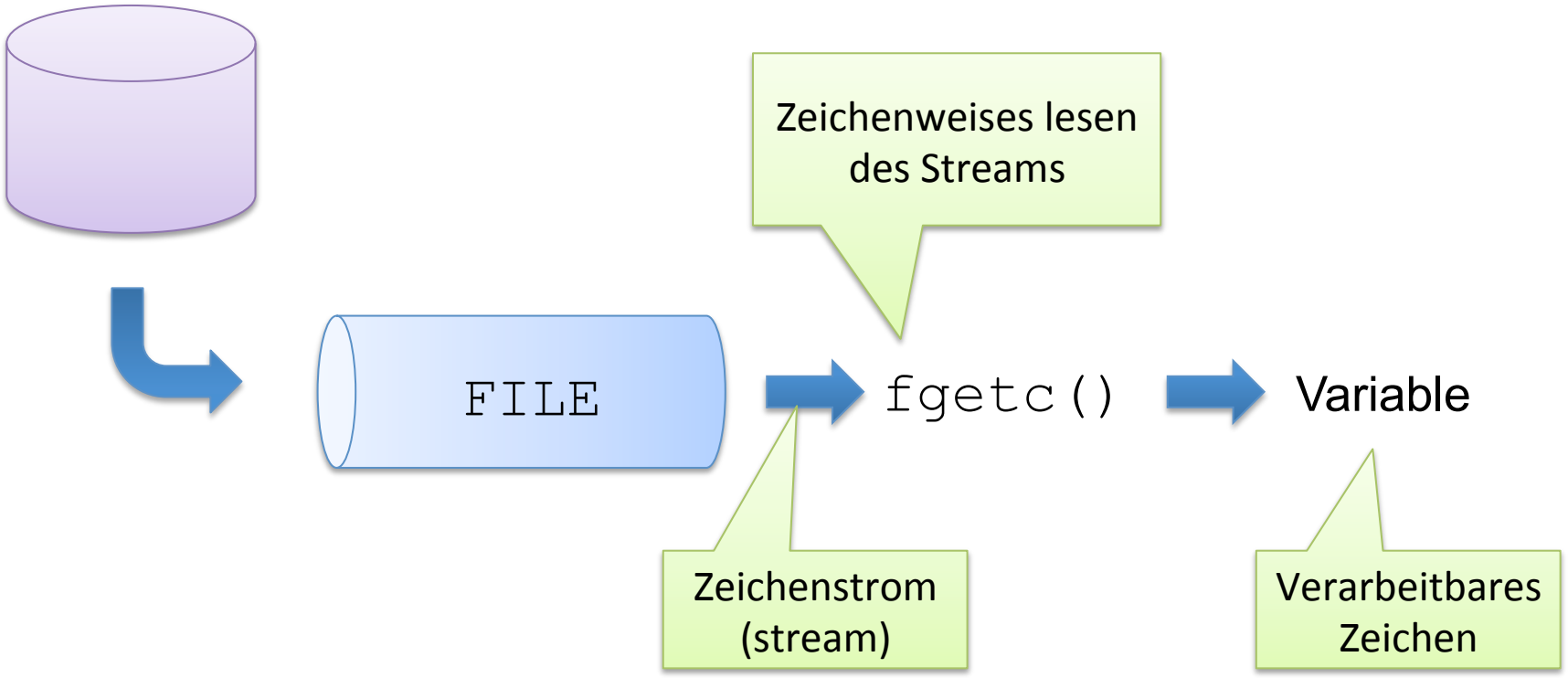
Schließen der geöffneten Datei:

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

Mehr Informationen: Offizielle Manuals zur Funktion.

Dateien einlesen mittels fgetc()

Datei

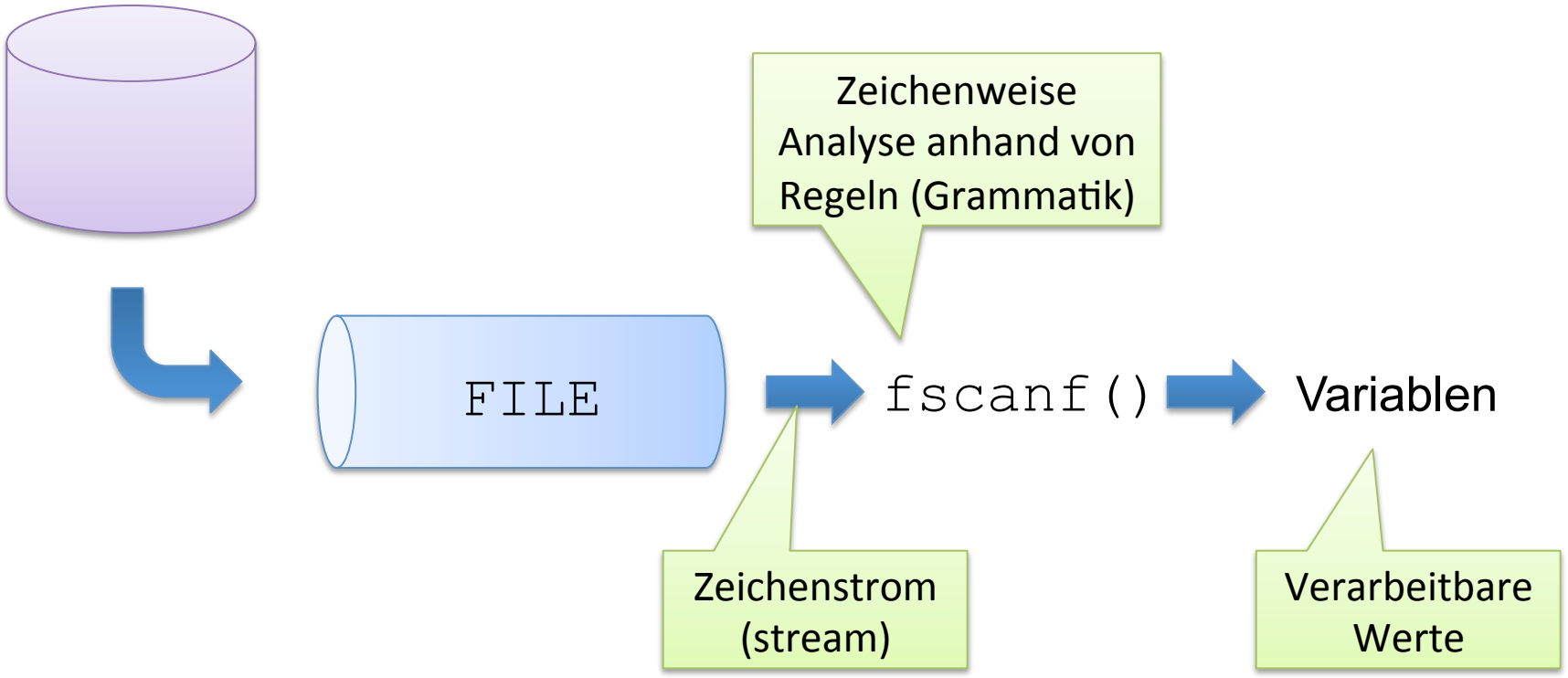


Hinweis: Der Kanal zu einer bestimmten Datei muss zuvor geöffnet werden.

Dateien einlesen mittels fscanf()

Datei

Analyseregeln werden anhand der Format Specifier ausgewählt.



Hinweis: Der Kanal zu einer bestimmten Datei muss zuvor geöffnet werden.

Dateizugriff in C - Lesen

Lesezugriffe erfolgen auf formatiertem Text mittels fscanf()

```
int fscanf ( FILE * stream, const char * format, ... );
```

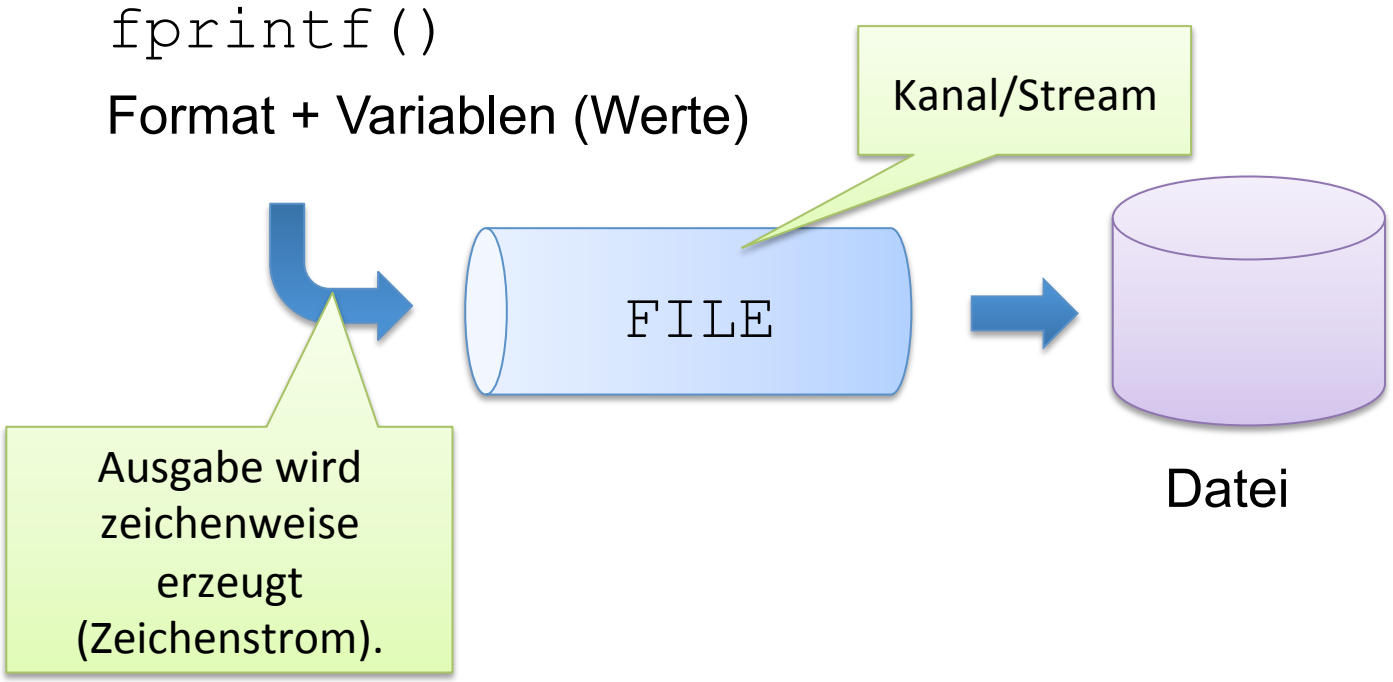
Ein Dateiende wird mittels `feof()` erkannt.

Beispiel:

```
FILE *exampleFile = 0;
int x = 0;
int y = 0;
exampleFile = fopen("c:\\tmp\\numbers.txt", "r");
if (NULL != exampleFile) {
    while (!feof(exampleFile)) {
        fscanf(exampleFile, "%i:%i\n", &x, &y);
        printf("%i: %i\n", x, y);
    }
    fclose(exampleFile);
}
```

Textdateien schreiben mittels fprintf()

Formatierung der Daten wird im Format Specifier festgelegt.



Hinweis: Der Kanal zu einer bestimmten Datei muss zuvor geöffnet werden.

Dateizugriff in C - Schreiben

Schreibzugriffe erfolgen binär mittels fwrite() oder formatiert mittels fprintf()

```
int fprintf ( FILE * stream, const char * format, ... );
```

Beispiel:

```
FILE *exampleFile = 0;
exampleFile = fopen("c:\\tmp\\numbers.txt", "w");

if (NULL != exampleFile) {
    for (int i = 0; 5 > i; i++) {
        fprintf(exampleFile, "%i:%i\n", i, i * i);
    }
    fclose(exampleFile);
}
```

Dateizugriff in C – File-Position

- Aktuelle Position innerhalb der Datei bestimmen:

```
long int ftell ( FILE * stream );
```

- Aktuelle Position innerhalb der Datei neu setzen:

```
int fseek ( FILE * stream, long int offset, int origin );
```

- Buffer des Streams leeren
  (Daten wirklich auf Datenträger ablegen)

```
int fflush ( FILE * stream );
```