Programmiertechniken 2

Prof. Dr.-Ing. Zhen Ru Dai
zhenru.dai@haw-hamburg.de

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 (d.h. für eine längere Zeit verbleibend).
- Dateien werden in einem Dateisystem (File System) organisiert und über einen Pfad + Name identifiziert. Beispiele:

  c:\temp\auftrag.txt
  /var/spool/messages.log

Text-Dateien

- 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:
```

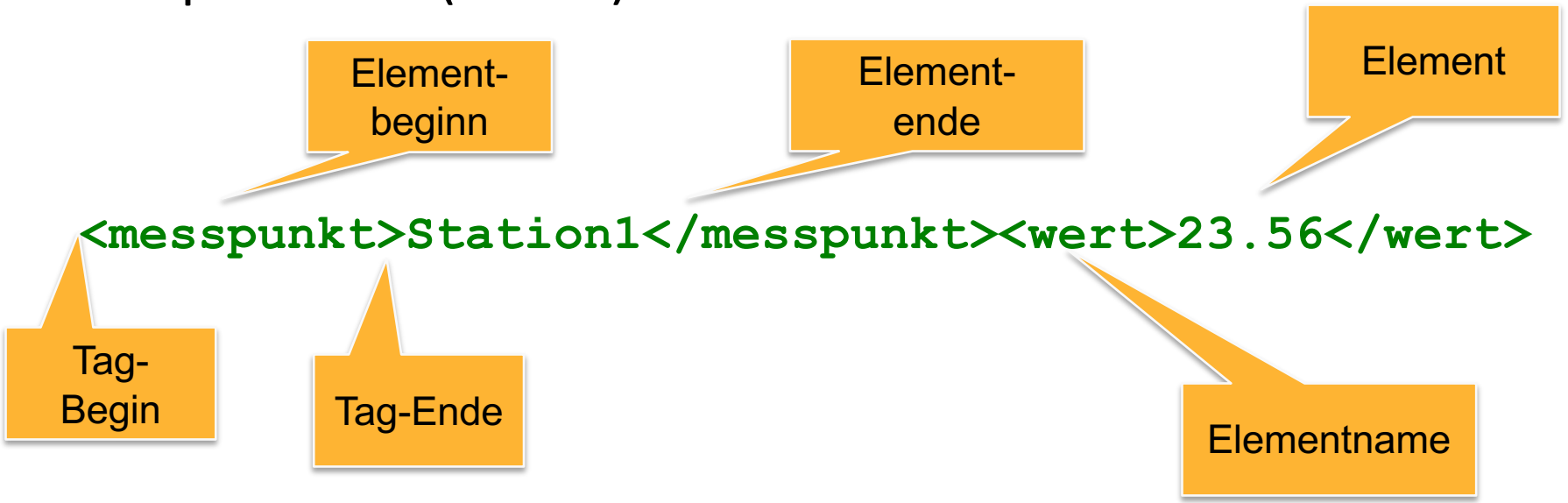
Trennsymbole

- 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
```

Markup-Datein

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



- Hierarchische Schachtelung ist erlaubt.

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:

```
#include<stdio.h>

FILE *textfile = NULL;
```

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

Dateizugriff - Öffnen

Öffnen der Datei:

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

- mode:
  - r: nur lesen. Kreiiert kein neues Datei, wenn nicht vorhanden. Alte Datei wird nicht geleert.
  - w: nur schreiben. Neue Datei kreiieren, wenn keine vorhanden. Immer am Dateianfang schreiben. Alte Datei wird zuerst geleert und dann beschrieben.
  - a: Daten ans Ende hinzufügen. Wenn Datei nicht existiert, dann wird eine erzeugt.
  - r+: lesen und Schreiben erlaubt. Anfang wird überschrieben, Rest bleibt bestehen. Wenn keine Datei vorhanden, dann Fehler.

Dateizugriff – Öffnen und Schliessen

Öffnen der Datei:

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

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

Schließen der Datei:

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

Dateizugriff - Ausgeben

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

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

- Beispiel:

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

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

Dateizugriff - Lesen

- Lesezugriffe erfolgen binär mittels ***fread()*** oder auf formatiertem Text mittels ***fscanf()***

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

-> Beispiel: fscanf (pFile, "%s", str);
```

- Ein Dateiende wird mittels ***feof()*** erkannt.

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

- Stream einfügen mittels ***fputs()***.

```
int fputs ( const char * str, FILE * stream );

-> Beispiel: fputs (sentence, pFile);
```

Dateizugriff - Lesen

- Beispiel:

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

ESCAPE-Sequence, liefert einfachen Backslash

Dateizugriff – 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 );
```

Beispiel:

```
int main () {
        FILE * pFile;
        pFile = fopen ( "example.txt" , "w" );
        fputs ( "This is an apple." , pFile );
        fseek ( pFile , 9 , SEEK_SET );
        fputs ( " sam" , pFile );
        fclose ( pFile );
        return 0;
}
```

Anfangsposition

Was ist das Ergebnis?

Ergebnis: „*This is a sample.*“

Verwendung mit Klassen

- Sollen Klassen Daten lesen oder schreiben, so wird oft eine geöffnete Datei der Methode übergeben:

```
void Point3D::writeToFile(FILE* outfile) {
    std::fprintf(outfile, "%f \t %f \t %f \n", x, y, z);
}
```

- Verantwortung Datei bereitzustellen: außerhalb
- Verantwortung der Ausgabe: Klasse