Technische Universität Braunschweig

Institut für Flugführung

Dateieingabe und -ausgabe

Prof. Dr.-Ing. Peter Hecker, Dipl.-Ing. Paul Frost, Andreas Dekiert M. Sc.,
28. Mai 2019

Agenda

09. April	Einführung
16. April	Softwareprojektmanagement
23. April	Entwicklungstools
30. April	GitHub
07. Mai	Software-Dokumentation und Bug-Reporting
14. Mai	Einführung Arduino
21. Mai	Frei
28. Mai	**Dateieingabe und -ausgabe**
4. & 11. Juni	Tag der Lehre und Exkursionswoche
18. Juni	Einführung von Qt
25. Juni	GUI-Erstellung mit Qt
02. Juli	Serielle Kommunikation
09. Juli	API-Anleitung und Projektarbeit
16. Juli	Vorbereitung der Abgabe und Fragen
12. August 10:00	Abgabe

Lehrziele

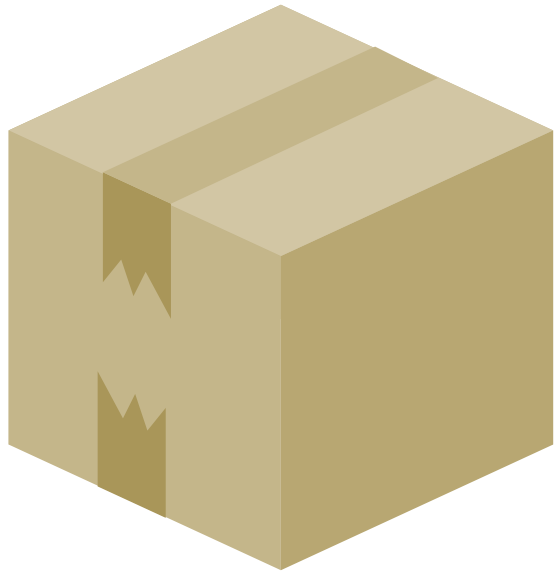
Dateieingabe und -ausgabe

Als Teilnehmer soll ich am Ende dieser Übung...

- ☐ verbreitete Dateiformate kennen
- ☐ Daten aus Dateien einlesen können
- ☐ Daten und Protokolle in Dateien schreiben können
- ☐ Nutzereingaben erfragen und verarbeiten können

Icon mady by DinosoftLabs from flaticon.com

Schreiben von Daten



Schreiben von Daten

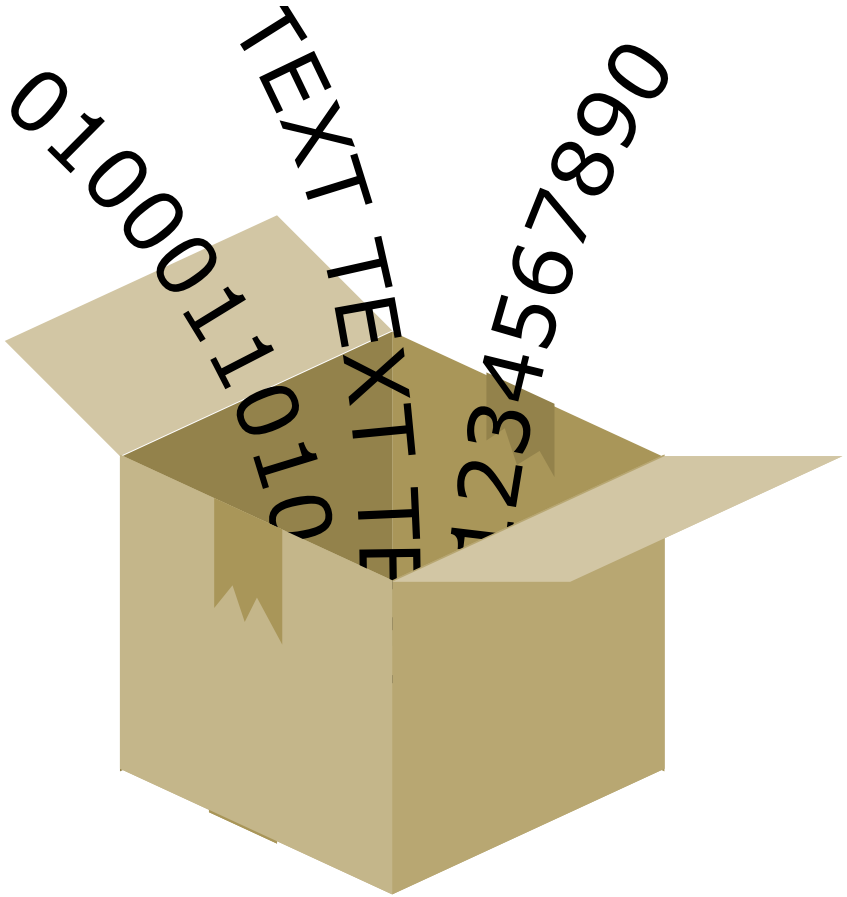
1. Datei zum Schreiben öffnen

   Falls die Datei noch nicht existiert, wird diese automatisch erstellt.

Schreiben von Daten

1. Datei zum Schreiben öffnen

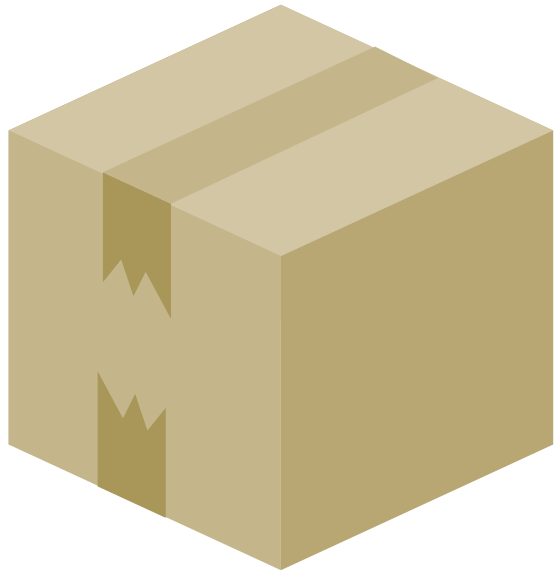
   Falls die Datei noch nicht existiert, wird diese automatisch erstellt.
2. Inhalt in Datei schreiben



Schreiben von Daten

1. Datei zum Schreiben öffnen

   Falls die Datei noch nicht existiert, wird diese automatisch erstellt.
2. Inhalt in Datei schreiben
3. Datei schließen



Schreiben von Daten

Beispiel

Listing 1: In neue Datei schreiben oder überschreiben

```
#include <fstream>

int main () {
   std::ofstream myFile; // output-file-stream
   myFile.open("hello.txt");
   myFile << "Hello World" << std::endl;
   myFile.close();
   return 0;
}
```

Achtung

Die Datei wird vollständig überschrieben.

Schreiben von Daten

Ergebnis

Anzeige im Texteditor

```
Hello World
```

Hexadezimale Werte (ASCII-Codierung)

H	e	l	l	o		W	o	r	l	d
48	65	6C	6C	6F	20	57	6F	72	6C	64

Binäre Werte (1B pro Zeichen)

H	e	l	l	o	
0100 1000	0110 0101	0110 1100	0110 1100	0110 1111	0010 0000

W	o	r	l	d
0101 0111	0110 1111	0111 0010	0110 1100	0110 0100

ASCII-Tabelle

HEX	...0	...1	...2	...3	...4	...5	...6	...7	...8	...9	...A	...B	...C	...D	...E	...F
0...	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1...	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2...	SP	!	"	#	$	%	&	’	(	)	*	+	,	-	.	/
3...	0	1	2	2	4	5	6	7	8	9	:	;	<	=	>	?
4...	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5...	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6...	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7...	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Anfügen von Daten

Beispiel

Listing 2: Inhalt von bestehender Datei erweitern

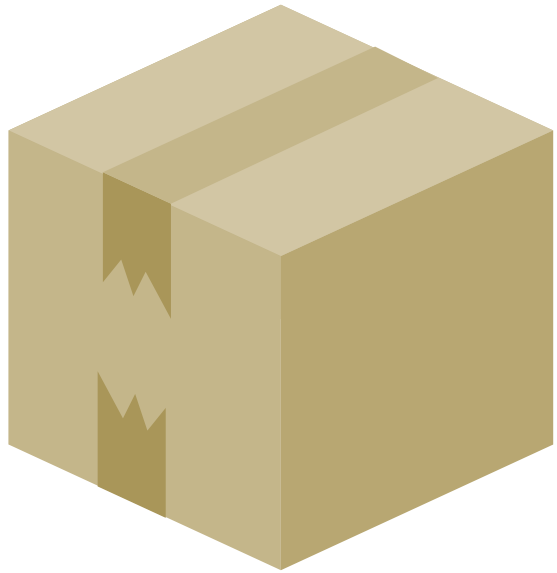
```
#include <fstream>
#include <ctime>

int main () {
   time_t currentTime = time(nullptr);
   struct tm tC{};
   localtime_s(&tC, &currentTime);

   std::ofstream myFile;
   myFile.open("log.txt", std::ios::app);
   myFile << "Current Time: " << tC.tm_hour << "
      hr " << tC.tm_min << "min\n";
   myFile.close();
   return 0;
}
```

Gibt es Fragen oder Anmerkungen zu dem Unterthema
Schreiben von Dateien?

Lesen von Daten



Lesen von Daten

1. Datei zum Lesen öffnen

Lesen von Daten

1. Datei zum Lesen öffnen
2. Prüfen, ob die Datei geöffnet werden konnte

Lesen von Daten

01000110101

1. Datei zum Lesen öffnen
2. Prüfen, ob die Datei geöffnet werden konnte
3. Datei zeilenweise auslesen

Lesen von Daten

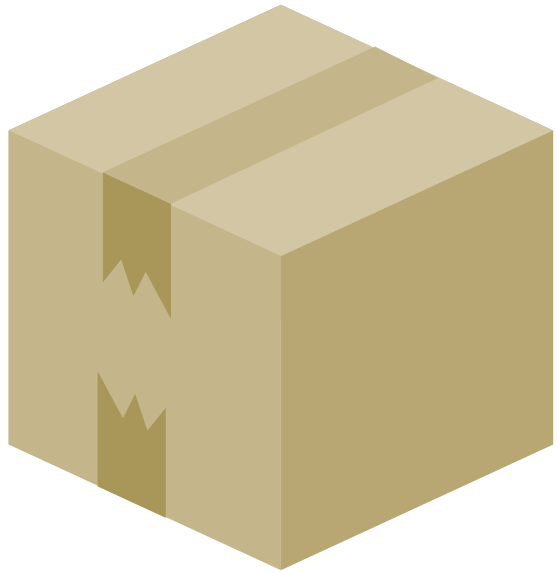
1. Datei zum Lesen öffnen
2. Prüfen, ob die Datei geöffnet werden konnte
3. Datei zeilenweise auslesen

Lesen von Daten

1. Datei zum Lesen öffnen
2. Prüfen, ob die Datei geöffnet werden konnte
3. Datei zeilenweise auslesen

Lesen von Daten

1. Datei zum Lesen öffnen
2. Prüfen, ob die Datei geöffnet werden konnte
3. Datei zeilenweise auslesen
4. Datei schließen



Lesen von Daten

Beispiel 1/2

Die Funktion `std::getline(STREAM, STRING)` liest zeilenweise aus einem `Stream` und speichert den Text in der übergebenen `String`-Variable.
Sind keine Zeilen mehr vorhanden, gibt die Funktion `false` zurück.

Listing 3: Aus einer Datei lesen

```
#include <string>
#include <iostream>
#include <fstream>
//..
```

Lesen von Daten

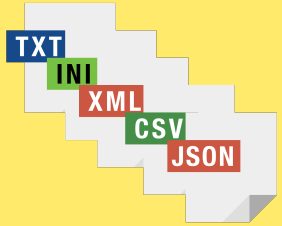
Beispiel 2/2

Listing 4: Aus einer Datei lesen

```
int main () {
   std::ifstream myFile; // input-file-stream
   myFile.open("log.txt");
   if (myFile.is_open())
   {
      std::string line;
      while (std::getline(myFile, line))
      {
         std::cout << line << std::endl;
      }
      myFile.close();
   }
   std::cin.get(); // Auf Enter warten...
   return 0;
}
```

Gibt es Fragen oder Anmerkungen zu dem Thema
Datei I/O?

Dateiformate



csv

ini

xml

json

Parser

Icons mady by Freepik from flaticon.com

Comma Separated Values

*.csv

Eigenschaften

- Werte sind tabellarisch angeordnet
- Eine Zeile in der csv-Datei entspricht einer Zeile der Tabelle
- Erste Zeile wird häufig zur Beschriftung der Spalten genutzt
- Trennzeichen separieren die Spalten

Verwendung

- Messwerte
- Datensammlungen & kleine Datenbanken

Comma Separated Values

Beispiel

Listing 5: Beispiel .csv-Datei

```
ID,Hersteller,Muster,MTOW
1,Airbus,A320neo,79000
2,Airbus,A380,575000
3,Cessna,172R,1111
4,Gulfstream,G650,45200
5,Bombardier,CS100,60781
6,Boeing,787-9,254011
7,Boeing,747-8,447700
8,Embraer,E175,40370
9,Schleicher,ASK-21,600
```

Comma Separated Values — Vor- und Nachteile

Vorteile

- Werte sind im Texteditor einsehbar und interpretierbar
- Sehr einfach zu verarbeiten
  → Bspw. auch in MS Excel importierbar
- Minimaler Speicher-Overhead zur Organisation der Daten

Nachteile

- Nur sinnvoll für tabellarisch organisierbare Daten
- Daten dürfen weder Trennzeichen noch Zeilenumbrüche enthalten
  → Kodierung dieser Zeichen wäre erforderlich (und möglich)

Initialisierungsdatei *.ini

Eigenschaften

- Schlüsseln werden Werte zugeordnet („Key-Value-Pairs“/ Wertepaare)
  `Schlüssel = Wert`
- Mehrere Schlüssel *können* gruppiert werden
  `[Gruppenname]`
- Schlüssel müssen innerhalb einer Gruppe eindeutig sein
- Gruppennamen müssen innerhalb der Datei eindeutig sein
- Pro Zeile ein Wertepaar oder eine Gruppendefinition

Verwendung

- Speicherung von Einstellungen und Parametern

Initialisierungsdatei Beispiel

Kommentarzeilen werden durch vorangestelltes ; gekennzeichnet.

Listing 6: Beispiel .ini-Datei

```
[SimConnect]
level=verbose
console=1
RedirectStdOutToConsole=1
OutputDebugString=1
;Nicht mehr benoetigte Einstellungen:
;file=c:\simconnect%03u.log
;file_next_index=0
;file_max_index=9
```

Initialisierungsdatei — Vor- und Nachteile

Vorteile

- Werte sind im Texteditor einsehbar und interpretierbar
- Relativ einfaches Format
- Wenig Overhead zum Speichern vieler verschiedener Parameter

Nachteile

- Inhalte können nur als Wertepaare gespeichert werden
- Nur eine Gruppenhierarchieebene vorgesehen

Extensible Markup Language `*.xml`

Eigenschaften

- Werte werden in Tags oder Attributen dargestellt

```
<Tag>Wert</Tag>
<Tag Attribut="WertAttribut">WertTag</Tag>
```

- Baumstruktur durch Verschachteln von Tags
- Inhärente Validierung der Datenstruktur

Verwendung

- Webinhalte, `HTML`
- Geographische Daten (OpenStreetMap)
- Datenaustausch, primär mit Online-Diensten

Extensible Markup Language — Beispiel

Listing 7: Beispiel .xml-Datei

```
<?xml version="1.0" encoding="UTF-8"?>
<aircraft Hersteller="McDonnell Douglas"
  Muster="MD-11F">
  <MTOW>285990</MTOW>
  <Reichweite>7242</Reichweite>
  <Nutzlast>
    <PAX>0</PAX>
    <Fracht>94922</Fracht>
  </Nutzlast>
  <Antriebsvarianten>
    <Variante TW="PW4460" Anzahl="3" />
    <Variante TW="PW4462" Anzahl="3" />
    <Variante TW="CF6-80C2D1F" Anzahl="3" />
  </Antriebsvarianten>
</aircraft>
```

Extensible Markup Language Vor- und Nachteile

Vorteile

- Werte sind im Texteditor einsehbar und interpretierbar
- Intuitives Format
- Flexibles Format innerhalb der Baumstruktur
- Validierung der Datenstruktur möglich

Nachteile

- Aufwendigeres Auslesen als von `csv`- oder `ini`-Dateien
  → Einsatz eines Parsers sinnvoll
- Erhöhter Speicheraufwand durch öffnende und schließende Tags

JavaScript Object Notation `*.json`

Eigenschaften

- Datenstruktur kombiniert Eigenschaften von `.ini` und `.xml`
- Vordefinierte Steuerzeichen gruppieren Wertepaare zu Objekten
- Trennung der Wertepaare und Unterobjekte erfolgt über ein Komma
- Objekte werden als Baumstruktur angelegt

Verwendung

- Einstellungen
- Datenaustausch, primär mit Online-Diensten
- Alternative zu `XML`

JavaScript Object Notation — Beispiel

Listing 8: Beispiel .json-Datei

```
{
   "Hersteller":"McDonnell Douglas",
   "Muster":"MD-11F",
   "MTOW":285990,
   "Reichweite":7242,
   "Nutzlast": {
      "PAX":null,
      "Fracht":94922
   },
   "Antriebsvarianten": [
         { "TW":"PW4460", "Anzahl":3 },
         { "TW":"PW4462", "Anzahl":3 },
         { "TW":"CF6-80C2D1F", "Anzahl":3 }
   ]
}
```

JavaScript Object Notation — Vor- und Nachteile

Vorteile

- Unterstützung von 6 Datentypen:

  `Null`, Boolean, Zahlen, Zeichenketten, Arrays und Objekte
- Ebenso flexibles Format innerhalb der Baumstruktur wie `XML`
- Geringerer Speicheroverhead als `XML`
- Native Unterstützung in `JavaScript` (Webentwicklung)

Nachteile

- Aufwendigeres Auslesen als von `csv`- oder `ini`-Dateien

  → Parser erforderlich
- Schwieriger lesbar als `XML`, `csv` und `ini`

Parser

Werkzeug zum Umwandeln, Ein- & Auslesen von Datenformaten

- Ermöglicht den einfachen programmatischen Zugriff auf die Daten
- Bekanntestes Beispiel für Parser: Webbrowser
  - Wandelt `XML`/`HTML`-Daten zur Anzeige um
- Jedes Daten ein- und/oder auslesende Programm agiert als Parser
- Bibliotheken stellen Parser für gängige Formate bereit

Für C++ verfügbare Parser — Für Profis

json-Parser `https://github.com/nlohmann/json`

xml-Parser `http://stackoverflow.com/questions/9387610/what-xml-parser-should-i-use-in-c`

Gibt es Fragen oder Anmerkungen zu dem Thema
Dateiformate?

Abgehakt

Dateieingabe und -ausgabe

Als Teilnehmer soll ich am Ende dieser Übung...

- [x] verbreitete Dateiformate kennen
- [ ] Daten aus Dateien einlesen können
- [ ] Daten und Protokolle in Dateien schreiben können
- [ ] Nutzereingaben erfragen und verarbeiten können

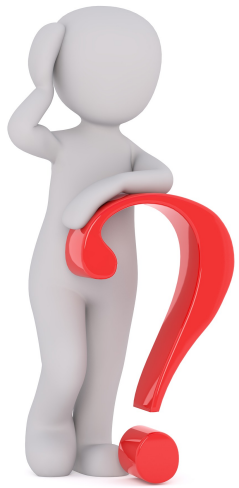
Übung

Es soll ein einfaches Programm geschrieben werden, welches den Nutzer begrüßt und ihm die Option gibt, den Begrüßungstext zu ändern. Alle Änderungen des Textes sollen protokolliert werden.

Aufgaben

1. Erstelle ein Programm, welches den Nutzer begrüßt
2. Lese den Begrüßungstext aus einer Datei ein
3. Erbitte die Eingabe eines neuen Begrüßungstextes durch den Nutzer
4. Speichere den neuen Begrüßungstext ab
5. Protokolliere alle eigegebenen Begrüßungstexte
6. Implementiere `Keywords` zum Beenden das Programmes und zum Zurücksetzen der Begrüßung

Wie erstelle ich ein Programm, welches den Nutzer begrüßt und sich nicht gleich beendet?



Begrüßung ausgeben

Der Ausgabe-Stream `cout` schreibt Text auf den Bildschirm.
Der Befehl `cin.get()` wartet, bis die Enter-Taste gedrückt wird.

Listing 9: Den Nutzer begrüßen

```
#include <iostream>

int main()
{
   std::cout << "Hello World!" << std::endl;
   std::cout << "Press Return to continue...";
   std::cin.get();
   return 0;
}
```

Wie lese ich einen Text aus einer Datei ein und verwende ihn im Code?

Daten einlesen

Listing 10: Begrüßung aus Datei einlesen

```
#include <iostream>
#include <fstream>
#include <string>

int main()
{
   std::string sWelcome = "Hello World!";

   std::ifstream readFile;
   readFile.open("welcome.conf");
   if(readFile.is_open())
   {
      std::string sLine;
      std::getline(readFile, sLine);
      readFile.close();
```

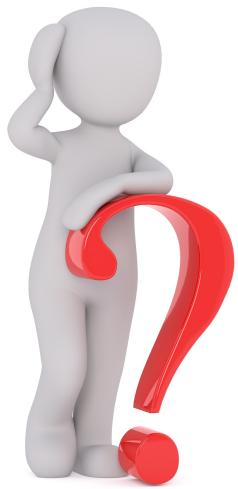
Daten einlesen

Listing 11: Begrüßung aus Datei einlesen

```
    if (sLine.length() > 0)
      sWelcome = sLine;
  }

  std::cout << sWelcome << std::endl;
  std::cout << "Press Return to continue...";
  std::cin.get();
  return 0;
}
```

Wie erfrage ich eine Nutzereingabe?



Nutzereingabe

Der Eingabe-Stream `cin` liefert alle Tastatureingaben des Nutzers. Er kann mittels `std::getline()` genauso ausgelesen werden wie eine Datei.

Listing 12: Nutzereingabe abfragen

```
// ...
   std::string newMsg;
   std::cout << "I am sure you would like to
      edit the welcome message..." << std::endl;
   std::cout << "Enter new welcome message: ";
   std::getline(std::cin, newMsg);
//...
```

Wie kann ich mit der Nutzereingabe den Begrüßungstext des nächsten Programmstarts aktualisieren?

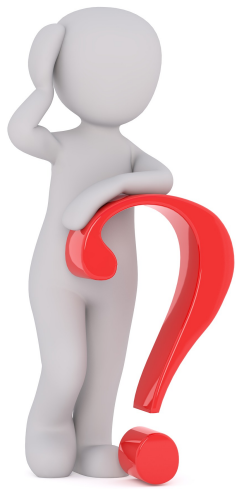
Begrüßung speichern

`ofstream` erstellt eine neue Datei, falls sie noch nicht vorhanden ist, und öffnet einen Output-Stream, um in die Datei zu schreiben.
Standardmäßig wird der gesamte Dateiinhalt überschrieben.
Der Stream kann genauso verwendet werden wie `cout`.

Listing 13: Neue Begrüßung abspeichern

```
//...
   if (newMsg.length() > 0)
   {
      std::ofstream myWriteFile;
      myWriteFile.open("welcome.conf");
      myWriteFile << newMsg << std::endl;
      myWriteFile.close();
   }
//...
```

Wie kann ich ein Protokoll, bspw. mit Nutzereingaben, schreiben?



Eingaben protokollieren

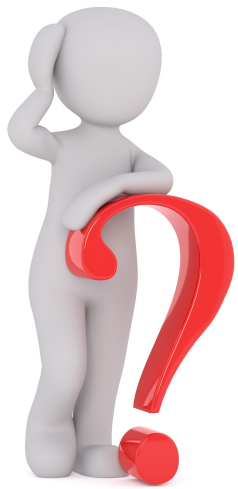
Um ein Protokoll aller eingegebenen Texte zu erstellen, darf die `log`-Datei nicht überschrieben werden, sondern der neue Inhalt muss an den bestehenden angehängt werden.

→ `ofstream::open()` wird die Option `std::ios::app`, für „append“ = anhängen, übergeben

Listing 14: Protokoll schreiben

```
//...
   if (newMsg.length() > 0)
   {
      std::ofstream logFile;
      logFile.open("log.txt", std::ios::app);
      logFile << newMsg << std::endl;
      logFile.close();
   }
//...
```

Wie prüfe ich die Nutzereingabe auf ein `Keyword`?



Keyword überprüfen

Die Nutzereingabe liegt als `String` vor.
`Strings` können ebenso wie Zahlen direkt auf Gleichheit geprüft werden.

Listing 15: Protokoll schreiben

```
//...
  if (newMsg == "exit") {
    return 0;
  } else if (newMsg == "reset") {
    std::cout << "Resetting..." << std::endl;
    newMsg = "Hello World!";
  }
//...
```

Gibt es Fragen oder Anmerkungen zu dem Unterthema
Datei I/O und Nutzereingaben?

Abgehakt

Dateieingabe und -ausgabe

Als Teilnehmer soll ich am Ende dieser Übung...

- ☑ verbreitete Dateiformate kennen
- ☑ Daten aus Dateien einlesen können
- ☑ Daten und Protokolle in Dateien schreiben können
- ☑ Nutzereingaben erfragen und verarbeiten können

Sprintmeeting 15 Min

Jetzt besteht die Möglichkeit, das Sprintmeeting durchzuführen.

Protokolliert bitte

- die bearbeiteten Aufgaben der Vorwoche.
- die Zwischenstände der geplanten Aufgaben.
- die in der kommenden Woche zu bearbeitenden Aufgaben.

Ende

Vielen Dank für eure Aufmerksamkeit!