Einführung in I/O und File-Handling in C++

Florian Adamsky, B. Sc. (PhD cand.)

florian.adamsky@iem.thm.de
http://florian.adamsky.it/

Softwareentwicklung im WS 2014/15

THM
TECHNISCHE HOCHSCHULE MITTELHESSEN

Outline

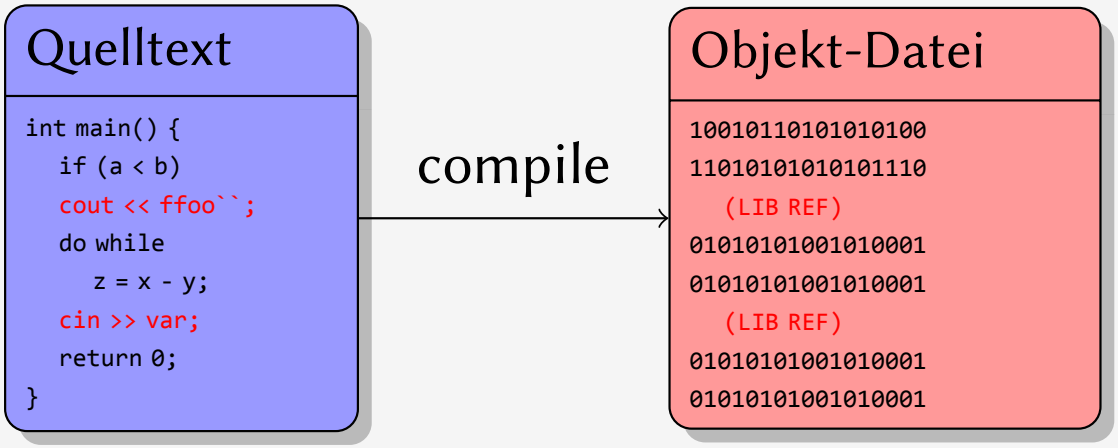
Inhaltsverzeichnis

Compilevorgang

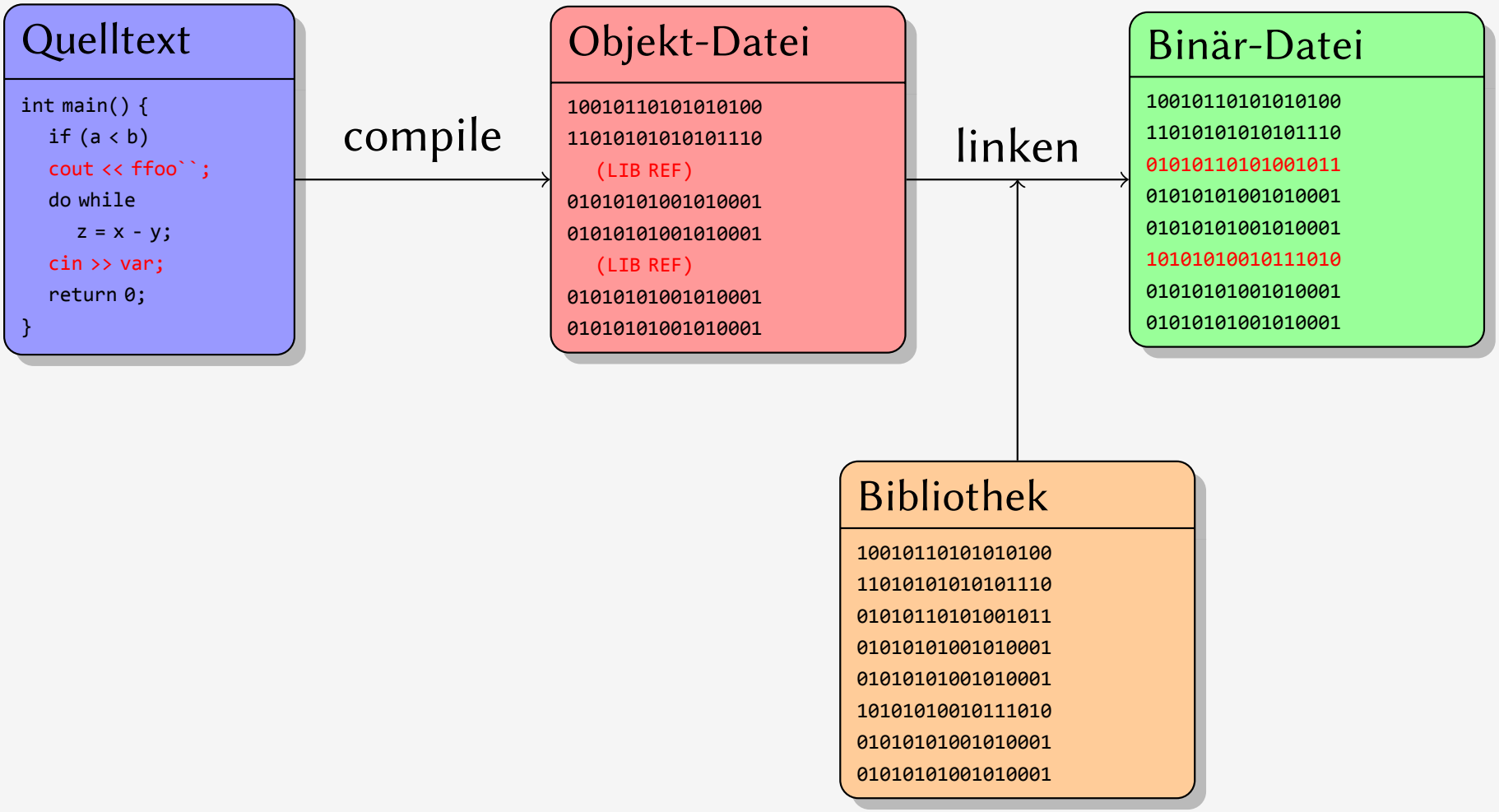
Quelltext

```
int main() {
   if (a < b)
   cout << ffoo``;
   do while
      z = x - y;
   cin >> var;
   return 0;
}
```

Compilevorgang



Compilevorgang



Entwicklungsumgebung

IDE

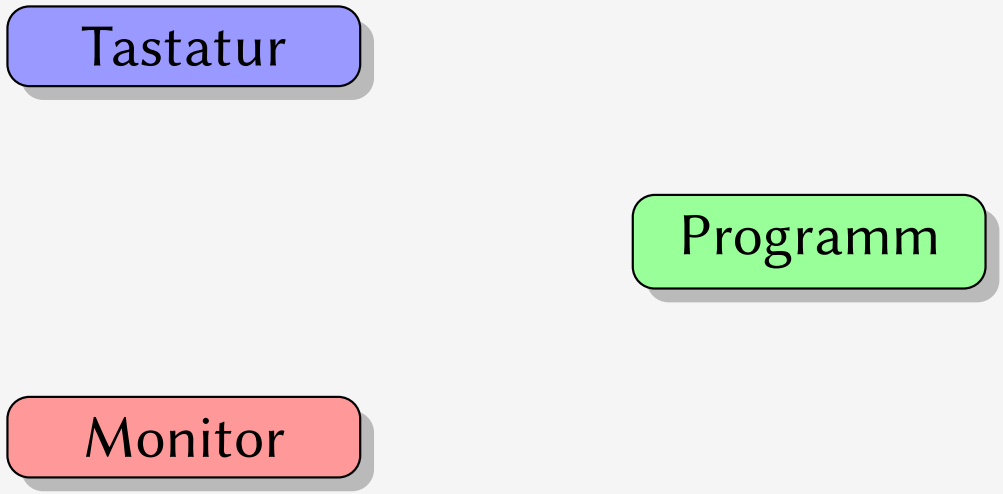
- Code::Blocks
  `http://www.codeblocks.org/`
- Visual C++ Express
  `http://www.microsoft.com/visualstudio/eng/products/visual-studio-2010-express`
- Eclipse CDT
  `http://www.eclipse.org/cdt/`

Texteditor

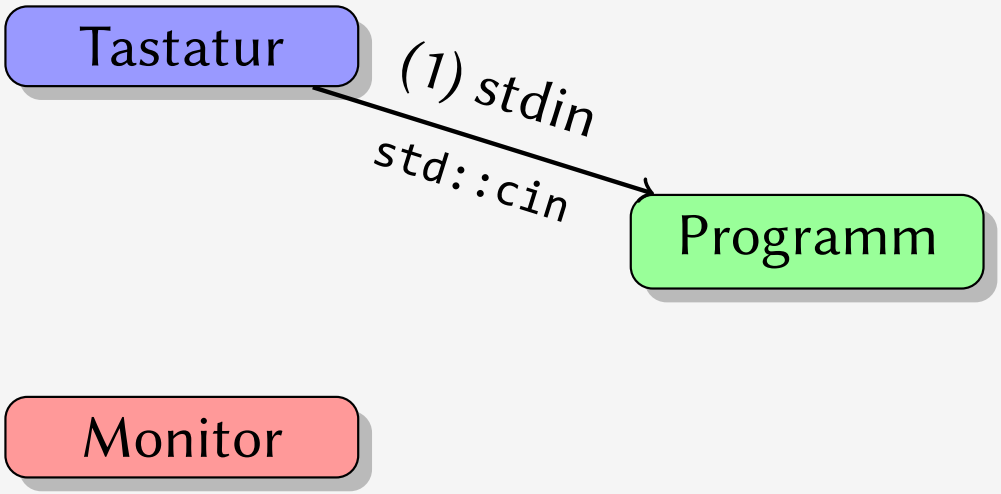
- Sublime Text 2
  `http://www.sublimetext.com/2`
- Notepad++
  `http://notepad-plus-plus.org/`
- Emacs
  `http://www.gnu.org/software/emacs/`

Inhaltsverzeichnis

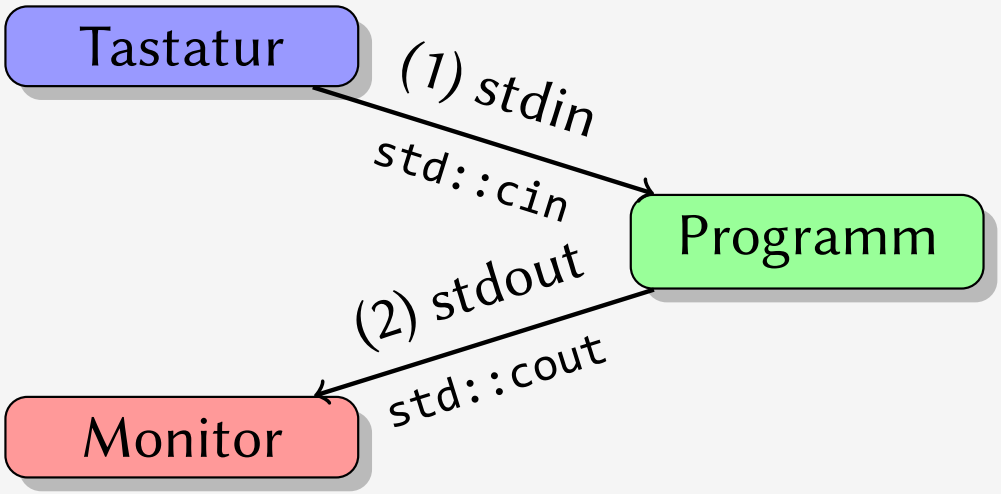
Standard-Datenströme



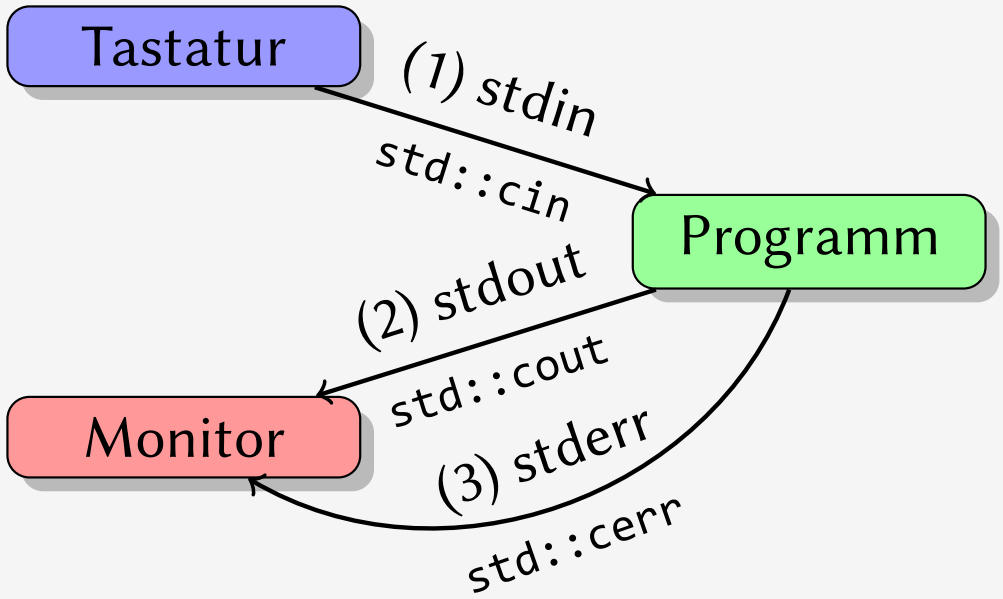
Standard-Datenströme



Standard-Datenströme



Standard-Datenströme



Ausgabeoperator <<

- Um die Standard Ein- und Ausgabefunktionen von C++ zu nutzen, ist es notwendig die folgende Datei zu inkludieren:

```
#include <iostream>
```

- Mit der Methode `cout` weisen Sie C++ an, Informationen auszugeben:

```
cout << "Ihre Eingabe: " << eingabe << endl;
```

- Der Ausgabeoperator (engl. *Insertation Operator*) << zeigt in die Richtung in die die Daten fließen und *verkettet* mehrere Daten miteinander

Manipulatoren

- Manipulatoren dienen dazu die Ein- und Ausgabe zu formatieren
- Der Manipulator `endl` fügt eine neue Zeile in den Ausgabestrom ein
- Über die Header-Datei `iomanip` können weitere Manipulatoren geladen werden:

Manipulator	Bedeutung
`endl`	Fügt neue Zeile ein
`setw(n)`	Abstand mit `n` Stellen
`left`	linksbündig
`right`	rechtsbündig
`setfill(c)`	`c` als Füllzeichen benutzen
`dec`	Stellt Zahl in dezimal dar
`oct`	Stellt Zahl in oktal dar
`hex`	Stellt Zahl in hexadezimal dar

Fließkomma-Manipulatoren

Manipulatoren	Bedeutung
`showpoint`	Zeige Dezimalpunkt mit Nachkommastellen an
`noshowpoint`	Nur relevante Stellen werden ausgegeben
`scientific`	Ausgabe in wissenschaftlicher Darstellung
`fixed`	Festgelegte Nachkommastelle
`setprecision(n)`	Genauigkeit mit n Stelle

Beispiel

Quelltext

```
#include <iostream>

using namespace std;
```

Beispiel

Quelltext

```
#include <iostream>

using namespace std;

int main(void) {
    // Ausgabe
    cout << "Hello World" << endl;

    int zahl = 42;
    cout << "Sinn des Leben ist "
         << zahl << endl;
}
```

Beispiel

Quelltext

```
#include <iostream>

using namespace std;

int main(void) {
    // Ausgabe
    cout << "Hello World" << endl;

    int zahl = 42;
    cout << "Sinn des Leben ist "
         << zahl << endl;
}
```

compiliert

Ausgabe

```
$ ./foo
Hello World
Sinn des Leben ist 42
```

Beispiele für Fließkommazahlen

Quelltext

```
// [...]
double irratio = sqrt(2);




// [...]
```

Ausgabe

```
// [...]




// [...]
```

Beispiele für Fließkommazahlen

Quelltext

```
// [...]
double irratio = sqrt(2);
cout << irratio;

// [...]
```

Ausgabe

```
// [...]

// [...]
```

Beispiele für Fließkommazahlen

Quelltext

```
// [...]
double irratio = sqrt(2);
cout << irratio;




// [...]
```

Ausgabe

```
// [...]


1.41421




// [...]
```

Beispiele für Fließkommazahlen

Quelltext

```
// [...]
double irratio = sqrt(2);
cout << irratio;

cout << setprecision(2) << irratio;




// [...]
```

Ausgabe

```
// [...]

1.41421




// [...]
```

Beispiele für Fließkommazahlen

Quelltext

```
// [...]
double irratio = sqrt(2);
cout << irratio;

cout << setprecision(2) << irratio;



// [...]
```

Ausgabe

```
// [...]

1.41421

1.4



// [...]
```

Beispiele für Fließkommazahlen

Quelltext

```
// [...]
double irratio = sqrt(2);
cout << irratio;

cout << setprecision(2) << irratio;

cout << fixed << setprecision(2)<< irratio;


// [...]
```

Ausgabe

```
// [...]

1.41421

1.4



// [...]
```

Beispiele für Fließkommazahlen

Quelltext

```
// [...]
double irratio = sqrt(2);
cout << irratio;

cout << setprecision(2) << irratio;

cout << fixed << setprecision(2)<< irratio;


// [...]
```

Ausgabe

```
// [...]

1.41421

1.4

1.41


// [...]
```

Beispiele für Fließkommazahlen

Quelltext

```
// [...]
double irratio = sqrt(2);
cout << irratio;

cout << setprecision(2) << irratio;

cout << fixed << setprecision(2)<< irratio;

cout << scientific<< setprecision(5) << irratio;
// [...]
```

Ausgabe

```
// [...]

1.41421

1.4

1.41


// [...]
```

Beispiele für Fließkommazahlen

Quelltext

```
// [...]
double irratio = sqrt(2);
cout << irratio;

cout << setprecision(2) << irratio;

cout << fixed << setprecision(2)<< irratio;

cout << scientific<< setprecision(5) << irratio;
// [...]
```

Ausgabe

```
// [...]

1.41421

1.4

1.41

1.41421e+00
// [...]
```

Fehlerausgabe

- Um Fehler auszugeben verwendet man die Methode `cerr`:

Example (Fehlermeldungen)

```
if (success)
    cout << "Alles in Ordnung" << endl;
else
    cerr << "Fehler" << endl;
```

- Ausgabe fließt in den Kanal `stderr (2)`, der jedoch standardmäßig mit `stdout (1)` verbunden ist
- Durch die verschiedenen Kanäle kann man Fehlermeldungen umlenken:

```
$ ./programm 2>fehlermeldungen.txt
```

Eingabeoperator >>

- Um die Standard Ein- und Ausgabefunktionen von C++ zu nutzen, ist es notwendig die folgende Datei zu inkludieren:

```
#include <iostream>
```

- Um eine Eingabe in einer Variable zu speichern, benutzt man die Methode `cin`:

```
cin >> eingabe;
```

- Der Eingabeoperator >> berücksichtigt den Datentyp

Beispiel

Quelltext

```
#include <iostream>

using namespace std;
```

Beispiel

Quelltext

```
#include <iostream>

using namespace std;

int main(void) {
    int    zahl1;
    double zahl2;
    char   zeichen;

    cin >> zahl1;
    cin >> zahl2;
    cin >> zeichen;
}
```

Inhaltsverzeichnis

Streams

- Stream: Verbindungsstück mit zwei Enden um Daten zu transferieren (Analogie: Schlauch)
  - Eingang: Festplatte, Tastatur oder ein anderes Programm
  - Ausgang: Festplatte, Bildschirm, oder Drucker
- Vorteil:
  - Flexibilität
- Streams sind effizienter mit Buffers

Ein Wort zu Buffer

Definition (Buffer)

Buffer: ist ein zusammenhängender Speicherbereich der als temporärer Speicher genutzt wird

- Möchte man 512 Bytes in eine Datei speichern, wäre es ineffizient 512 mal ein Byte zu speichern
- Besser ist es, die Daten in einem Buffer zu sammeln und mit einer Operation zu schreiben
- Einen Buffer zu leeren nennt man *flushing* und der dazugehörige Manipulator heißt `flush()`

```
cout << "foobar" << flush;
```

In eine Datei schreiben

1. Erstellen eines `ofstream` Objektes
2. Verbinden des Objektes mit einer bestimmten Datei
3. Das erstellte Objekt kann dann genauso benutzt werden wie `cout` oder `cerr`
4. Schließen der Verbindung

In eine Datei schreiben (Beispiel)

Example (Beispiel)

```
#include <fstream>

int main(void) {
    // Erstellen des Objektes
    std::ofstream fout;
```

In eine Datei schreiben (Beispiel)

Example (Beispiel)

```
#include <fstream>

int main(void) {
    // Erstellen des Objektes
    std::ofstream fout;

    // Verbinden mit einer Datei
    fout.open("test.txt");
```

In eine Datei schreiben (Beispiel)

Example (Beispiel)

```
#include <fstream>

int main(void) {
    // Erstellen des Objektes
    std::ofstream fout;

    // Verbinden mit einer Datei
    fout.open("test.txt");

    // Benutzen wie cout
    fout << "Lorem ipsum dolor sit amet";
```

In eine Datei schreiben (Beispiel)

Example (Beispiel)

```
#include <fstream>

int main(void) {
    // Erstellen des Objektes
    std::ofstream fout;

    // Verbinden mit einer Datei
    fout.open("test.txt");

    // Benutzen wie cout
    fout << "Lorem ipsum dolor sit amet";

    // Schliessen der Verbindung
    fout.close();
}
```

Vorsicht

Obacht

Das Öffnen einer Datei im default mode löscht den vorherigen Inhalt der Datei.

Aus einer Datei lesen

1. Erstelle ein `ifstream` Objekt
2. Verbinde das Objekt mit einer bestimmten Datei
3. Das erstellte Objekt kann genauso benutzt werden wie `cin`
4. Schließen der Verbindung

Aus einer Datei lesen (Beispiel 1)

Example (Beispiel)

```
#include <fstream>

int main(void) {
    // Erstellen des Objektes
    std::ifstream fin;
```

Aus einer Datei lesen (Beispiel 1)

Example (Beispiel)

```
#include <fstream>

int main(void) {
    // Erstellen des Objektes
    std::ifstream fin;

    // Verbinden mit einer Datei
    fin.open("test.txt");
```

Aus einer Datei lesen (Beispiel 1)

Example (Beispiel)

```
#include <fstream>

int main(void) {
    // Erstellen des Objektes
    std::ifstream fin;

    // Verbinden mit einer Datei
    fin.open("test.txt");

    // Benutzen wie cin
    char buf[80];
    fin >> buf;
```

Aus einer Datei lesen (Beispiel 1)

Example (Beispiel)

```
#include <fstream>

int main(void) {
    // Erstellen des Objektes
    std::ifstream fin;

    // Verbinden mit einer Datei
    fin.open("test.txt");

    // Benutzen wie cin
    char buf[80];
    fin >> buf;

    // Schliessen der Verbindung
    fin.close();
}
```

Aus einer Datei lesen (Beispiel 2)

Example (Einzelnes Zeichen einlesen)

```
char zeichen;
fin >> zeichen;
```

Aus einer Datei lesen (Beispiel 2)

Example (Einzelnes Zeichen einlesen)

```
char zeichen;
fin >> zeichen;
```

Example (String einlesen)

```
char buf[80];
fin >> buf;
```

Aus einer Datei lesen (Beispiel 2)

Example (Einzelnes Zeichen einlesen)

```
char zeichen;
fin >> zeichen;
```

Example (String einlesen)

```
char buf[80];
fin >> buf;
```

Example (Ganze Zeile einlesen)

```
string zeile;
getline(fin, zeile);
```

Datei Modi

Datei Modus	Bedeutung
`ios_base::in`	Datei lesend öffnen
`ios_base::out`	Datei schreibend öffnen
`ios_base::app`	Datei öffnen und Inhalt dranhängen
`ios_base::trunc`	Inhalt der Datei vorher löschen
`ios_base::ate`	Datei öffnen und ans Ende springen
`ios_base::binary`	Datei als Binärdatei öffnen
`ios_base::nocreate`	Datei nur öffnen falls vorhanden
`ios_base::noreplace`	Datei nur öffnen falls nicht vorhanden

Methode: open()

Example (Open)

```
ofstream myFile;
string filename = "test.txt";

myFile.open(filename.c_str(), ios::out | ios::app);
```

- Modi verbinden mit dem bitweise Oder |
- Die Methode open() benötigt einen C-String, daher muss C++-String erst umgewandelt werden

Methode: `eof()`

- Jede Datei endet mit einem speziellen `EOF`-Zeichen
- EOF ist ein Akronym und steht für *End Of File*
- Die Methode `eof()` liefert `true` zurück falls das Ende der Datei erreicht ist, ansonsten `false`

Example (Datei zeilenweise einlesen und ausgeben)

```
// ...
while (!fin.eof()) {
    string zeile;
    getline(fin, zeile);
    cout << zeile << endl;
}
// ...
```

Serialisierung

- ist die Abbildung von Daten oder Objekten in einer sequenziellen Form
- diese Form ermöglicht es die Daten abzuspeichern, zu laden oder über Netzwerk zu übertragen
- Einfachste Form der Serialisierung ist die Speicherung Byte für Byte

Obacht

Diese Form ist nicht kompatibel mit anderen Computersystemen (Little Endian, Big Endian)

Beispiel: Serialisierung von Objekten

Example (Philosoph abspeichern)

```
Student student;
student.name    = "Schopenhauer";
student.vorname = "Arthur";
student.mnr     = 1234;
```

Beispiel: Serialisierung von Objekten

Example (Philosoph abspeichern)

```
Student student;
student.name    = "Schopenhauer";
student.vorname = "Arthur";
student.mnr     = 1234;

ofstream file;
```

Beispiel: Serialisierung von Objekten

Example (Philosoph abspeichern)

```
Student student;
student.name    = "Schopenhauer";
student.vorname = "Arthur";
student.mnr     = 1234;

ofstream file;
file.open("testfile.bin", ios::binary);
```

Beispiel: Serialisierung von Objekten

Example (Philosoph abspeichern)

```
Student student;
student.name    = "Schopenhauer";
student.vorname = "Arthur";
student.mnr     = 1234;

ofstream file;
file.open("testfile.bin", ios::binary);
file.write((char*) (&student), sizeof(student));
```

Beispiel: Serialisierung von Objekten

Example (Philosoph abspeichern)

```
Student student;
student.name    = "Schopenhauer";
student.vorname = "Arthur";
student.mnr     = 1234;

ofstream file;
file.open("testfile.bin", ios::binary);
file.write((char*) (&student), sizeof(student));
file.close();
```

Beispiel: Deserialisierung von Objekten

Example (Philosph einlesen)

```
Student student;
```

Beispiel: Deserialisierung von Objekten

Example (Philosph einlesen)

```
Student student;



ifstream file;
```

Beispiel: Deserialisierung von Objekten

Example (Philosph einlesen)

```
Student student;



ifstream file;
file.open("testfile.bin", ios::binary);
```

Beispiel: Deserialisierung von Objekten

Example (Philosph einlesen)

```
Student student;



ifstream file;
file.open("testfile.bin", ios::binary);
file.read((char*) (&student), sizeof(student));
```

Beispiel: Deserialisierung von Objekten

Example (Philosph einlesen)

```
Student student;



ifstream file;
file.open("testfile.bin", ios::binary);
file.read((char*) (&student), sizeof(student));
file.close();

cout << student.name << endl;
```

What could possibly go wrong?

- keine Rechte um Datei zu lesen oder zu schreiben
- Datei oder Verzeichnis existiert nicht
- die zu lesende Datei ist gar keine Datei (z.B. ein Ordner)
- Zu viele Dateien sind gleichzeitig geöffnet
- ...

Fehlerbehandlung

Example (Fehlerbehandlung bei Datei-Operationen)

```
//...
if (!fin.is_open())  {
    cerr << "Konnte Datei " << filename
         << " nicht oeffnen" << endl;
    exit(1);
}

// code für file handling
```

Fehlerbehandlung

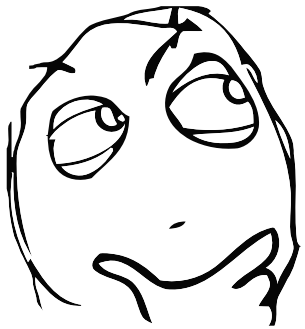
Example (Fehlerbehandlung bei Datei-Operationen)

```
//...
if (!fin.is_open())  {
    cer

    ex:
}

// code
```

Aber warum?



Aber warum konnte die Datei nicht geöffnet werden?

Fehlerbehandlung mit Grund

Example (Quelltext)

```
#include <cstring>
#include <cerrno>

// ...

if (!fin.is_open())  {
    cerr << "Konnte Datei " << filename
         << " nicht oeffnen: " <<  strerror(errno) << endl;
    exit(1);
}

// code für file handling
```

Fehlerbehandlung mit Grund

Example (Quelltext)

```
#include <cstring>
#include <cerrno>

// ...

if (!fin.is_open())  {
    cerr << "Konnte Datei " << filename
         << " nicht oeffnen: " <<  strerror(errno) << endl;
    exit(1);
}

// code für file handling
```

Inkludieren der Bibliothek <cerrno> zur Übersetzung der Fehlercodes in eine Fehlermeldung

Fehlerbehandlung mit Grund

Example (Quelltext)

```
#include <cstring>
#include <cerrno>

// ...

if (!fin.is_open())  {
    cerr << "Konnte Datei " << filename
         << " nicht oeffnen: " << strerror(errno) << endl;
    exit(1);
}

// code für file handling
```

Funktion `strerror` übersetzt anschließend den Fehlercode

Fehlerbehandlung mit Grund

Example (Quelltext)

```
#include <cstring>
#include <cerrno>

// .

if
        << " nicht oeffnen: " <<  strerror(errno) << endl;
    exit(1);
}

// code für file handling
```

Ausgabe

```
$ ./programm
Datei 'foobar' konnte nicht oeffnen: No such file or directory
```

Danke

Fragen?

florian.adamsky@iem.thm.de
http://florian.adamsky.it/