

Hydroinformatik - SoSe 2024

HyBHW-S1-01-V6a: Input/Output (I/O)

Olaf Kolditz

¹Helmholtz Centre for Environmental Research – UFZ, Leipzig

²Technische Universität Dresden – TUD, Dresden

³Center for Advanced Water Research – CAWR

⁴TUBAF-UFZ Center for Environmental Geosciences – C-EGS, Freiberg / Leipzig

Dresden, 17.05.2024

<https://www.ufz.de/index.php?de=40416>

<https://bildungsportal.sachsen.de/opal/auth/RepositoryEntry/32518209537?10>

Zeitplan: Hydroinformatik I

Sommersemester 2024

Nr.	KW	Datum	ID	Thema	Format
01	16	19.04.2024	HyBHW-1-01-01	Hydroinformatik - Einführung	O
02		19.04.2024	HyBHW-1-01-02	Werkzeuge (Compiler, github)	
03		19.04.2024	HyBHW-1-01-03	Jupyter, Python	
04	18	03.05.2024	HyBHW-1-01-04	Datentypen	P
05		03.05.2024	HyBHW-1-01	Installation: Compiler/Python	
06	19	10.05.2024	HyBHW-1-01-05	Klassen	O
07		10.05.2024	HyBHW-1-01-06	Input-Output (I/O)	
08	20	17.05.2024	HyBHW-1-01-07	Strings - Textverarbeitung	O
09		17.05.2024	HyBHW-1-01-08	Pointer Container	
10	22	31.05.2024	HyBHW-1-01-09	Hydrologische Modellierung	P
11		31.05.2024	HyBHW-1-01-10	BigData Water 4.0	
12	23	07.06.2024	HyBHW-1-01-11	Neuronale Netzwerke	P
13		07.06.2024	HyBHW-1-01-12	ANN / Bayes'sche Netzwerke	
14		07.06.2024	HyBHW-1-01-13	BN / Maschinelles Lernen	

Übung: Klassen::Konstruktor (EX05c-classes-constructor.cpp)

```
1 #include <iostream>
2 using namespace std;
3 #include "student.h"
4 int main()
5 {
6     CStudent* m_std = new CStudent(); // instance using constructor
7     cout << "EX05c: Constructor of class CStudent" << endl;
8     cout << "What have we created? m_std\t : " << m_std << endl;
9     cout << "What size has it?\t\t : " << sizeof(m_std) << endl;
10    cout << "What have we created? &m_std\t : " << &m_std << endl;
11    cout << "What size has it?\t\t : " << sizeof(&m_std) << endl;
12
13    cout << "First name:\t " << m_std->name_first << endl;
14    cout << "Last name:\t " << m_std->name_last << endl;
15    return 0;
16 }
```

Listing: Klassen::Konstruktor

Übung: Klassen::Konstruktor (student.h)

```
1 class CStudent
2 {
3     private:
4         long bank_account;
5     public:
6         string name_first;
7         string name_last;
8         string course;
9         string birthday;
10        long id;
11        string email;
12        string address;
13    public:
14        CStudent(); // constructor
15        ~CStudent(); // destructor
16        ios::pos_type Read(ifstream&);
17 };
```

Listing: Klassen::Deklaration

Übung: Klassen::Konstruktor (student.cpp)

```
1 #include "student.h"
2
3 CStudent::CStudent()
4 {
5     name_first = "Max";
6     name_last = "Mustermann";
7 }
8
9 CStudent::~~CStudent()
10 {
11 }
```

Listing: Klassen::Konstruktor

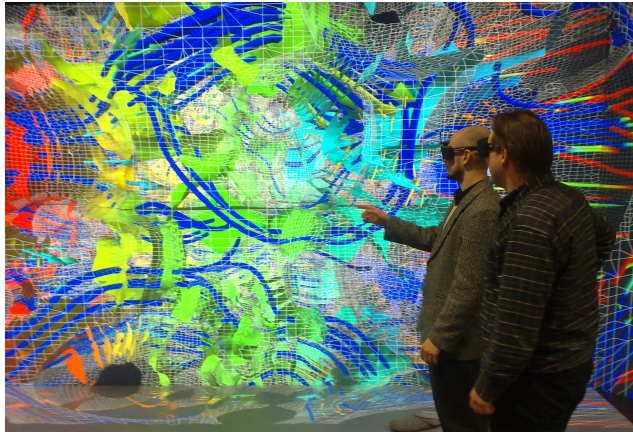
I/O Konzepte

Ein Programm (ob prozedural, modular, objekt-orientiert) ist eigentlich nichts weiter als eine Datenverarbeitung zwischen einer Eingabe (Input) und einer Ausgabe (Output). I und O können mehr oder weniger schick gemacht sein:

1. I/O Standardgeräte,
2. I/O Dateien,
3. Datenbanken (I) und Visualisierung (O).

UFZ – VISLab

Aus didaktischen Gründen müssen wir leider mit dem Langweiligsten - I/O Standardgeräte - anfangen. Spannend wird's, wenn Daten durch die Visualisierung 'lebendig' werden. Die Abb. 1 zeigt eine professionelle Datenaufbereitung einen Porenraummodells in unserem Labor für wissenschaftliche Visualisierung (TESSIN-VISLab) am UFZ in Leipzig.



Die `iostream` Klasse

Die Klasse `iostream` geht durch Mehrfachvererbung aus den Klassen `istream` und `ostream` hervor. `iostream` stellt damit die Funktionalität beider I/O Klassen zu Verfügung.

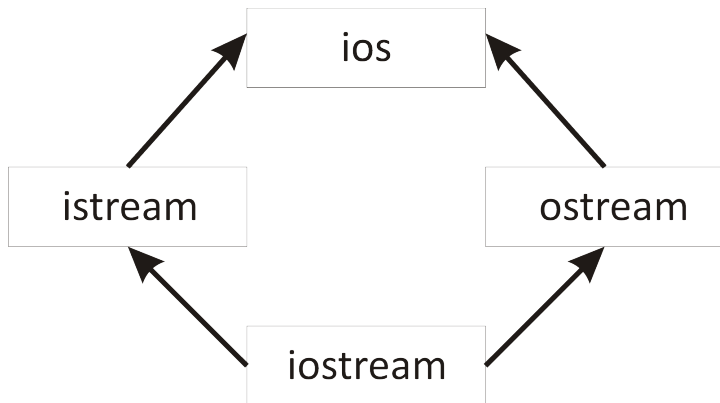


Figure: I/O stream Klassen

Die Standard-Streams

Es gibt vier Standard-Streams:

- ▶ `cin`: Standard-Eingabe über die Tastatur, Objekt der Klasse `istream`
- ▶ `cout`: Standard-Ausgabe auf dem Bildschirm, Objekt der Klasse `ostream`
- ▶ `cerr` und `clog`: zwei Objekte der Klasse `ostream` für die Fehlerausgabe.

Die Ein- `>>` und Ausgabeoperatoren `<<` transportieren die Ströme von und zu den Eingabe- bzw. Ausgabegeräten. Dabei formatieren sie die Datentypen (z.B. `int` in der Übung E3.1) entsprechend den Einstellungen der Klasse `ios`. Diese Einstellungen können durch Flags verändert werden (siehe nächsten Abschnitt).

Übung EX06a-io

```
1 #include <iostream>
2 using namespace std;
3 int main()
4 {
5     int zahl;
6     cout << "Bitte eine ganze Zahl eingeben: ";
7     cin >> zahl;
8     cout << zahl << endl;
9     return 0;
10 }
```

Formatierte Ausgaben

Wir beschäftigen uns mit der Gestaltung, d.h. Formatierung, von Ausgaben, wir wollen die Bildschirmausgabe schick machen, z.B. in Tabellenform, dass alles schön untereinander steht. Der zweite Aspekt der Formatierung ist die Genauigkeit von ausgegebenen Zahlenwerten.

Formatierte Ausgabe von Ganzzahlen

In der nachfolgenden Übung E3.2.1 beschäftigen wir uns mit den verschiedenen Ausgabemöglichkeiten von ganzen Zahlen.

```
1 #include <iostream>
2 using namespace std;
3 int main()
4 {
5     int zahl;
6     cout << "Bitte eine ganze Zahl eingeben: ";
7     cin >> zahl;
8     cout << uppercase // fuer Hex-Ziffern
9         << " oktal \t\t dezimal \t hexadezimal \n "
10        << oct << zahl << " \t\t "
11        << dec << zahl << " \t\t "
12        << hex << zahl << endl;
13     return 0;
14 }
```

Formatierte Ausgabe von Gleitpunktzahlen

In der nachfolgenden Übung beschäftigen wir uns mit den verschiedenen Ausgabemöglichkeiten von realen Zahlen.

Methoden	Wirkung
<code>int precision(int n)</code>	Genauigkeit wird auf n gesetzt

Formatierte Ausgabe von Gleitpunktzahlen

Übung EX06b-io-double

```
1 #include <iostream>
2 using namespace std;
3 int main()
4 {
5     double zahl;
6     cout << "Bitte eine Gleitkommazahl eingeben: ";
7     cin >> zahl;
8     cout.precision(7); // auf sieben Stellen genau
9     cout << "Standard: \t"          << zahl << endl;
10    cout << "showpoint: \t"         << showpoint << zahl << endl;
11    cout << "fixed: \t\t"           << fixed << zahl << endl;
12    cout << "scientific: \t"       << scientific << zahl << endl;
13    return 0;
14 }
```

Ausgabe von Speicherbedarf

In dieser Übung E3.2.3 benutzen wir den `sizeof` Operator, um den Speicherbedarf von Standard Daten-Typen zu bestimmen.

```
1 #include <iostream>
2 using namespace std;
3 int main()
4 {
5     cout << "Type\tNumber of bytes\n";
6     cout << "-----\n";
7     cout << "bool\t\t" << sizeof(bool) << endl;
8     cout << "char\t\t" << sizeof(char) << endl;
9     cout << "short\t\t" << sizeof(short) << endl;
10    cout << "int\t\t" << sizeof(int) << endl;
11    cout << "long\t\t" << sizeof(long) << endl;
12    cout << "float\t\t" << sizeof(float) << endl;
13    cout << "double\t\t" << sizeof(double) << endl;
14    cout << "long double\t" << sizeof(long double) << endl;
15    return 0;
16 }
```

Hydroinformatik - SoSe 2024

HyBHW-S1-01-V6b: Input/Output (I/O) - Files

Olaf Kolditz

¹Helmholtz Centre for Environmental Research – UFZ, Leipzig

²Technische Universität Dresden – TUD, Dresden

³Center for Advanced Water Research – CAWR

⁴TUBAF-UFZ Center for Environmental Geosciences – C-EGS, Freiberg / Leipzig

Dresden, 17.05.2024

<https://www.ufz.de/index.php?de=40416>

<https://bildungsportal.sachsen.de/opal/auth/RepositoryEntry/32518209537?10>

fstream Klasse

Abb. 2 zeigt die Hierarchie der fstream Klassen.

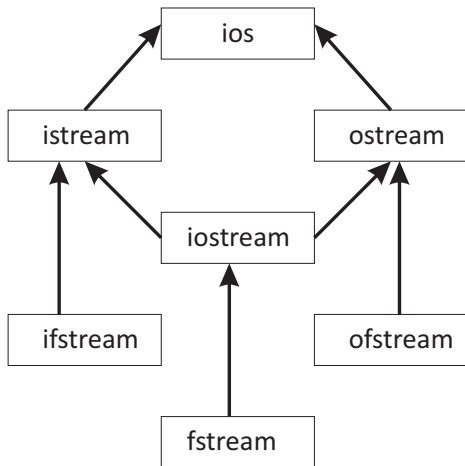


Figure: Die fstream Klassen

fstream Klasse

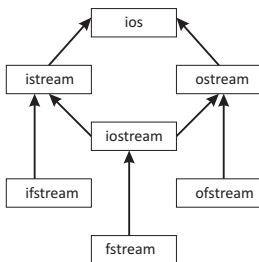


Figure: Die fstream Klassen

Diese sind von den bereits bekannten ios stream Klassen abgeleitet.

- ▶ `ifstream`: ist von `istream` abgeleitet für das Lesen von Dateien.
- ▶ `ofstream`: ist von `ostream` abgeleitet für das Schreiben von Dateien.
- ▶ `fstream`: ist von `iostream` abgeleitet für das Lesen und Schreiben von Dateien.

File streams anlegen

Eröffnungsmodus: Um eine Datei benutzen zu können, muss diese zunächst geöffnet werden.

Flag	Funktion
<code>ios::in</code>	Eine (existierende) Datei wird zum Lesen geöffnet.
<code>ios::out</code>	Eine Datei wird zum Schreiben geöffnet. Existierende Inhalte werden überschrieben.
<code>ios::app</code>	Die neuen Inhalte werden an die existierenden angehängt.
<code>ios::trunc</code>	Eine bestehende Datei wird beim öffnen auf die Länge 0 gekürzt.
<code>ios::ate</code>	Schreib- und Leseoperation werden auf das Dateiende gesetzt.
<code>ios::binary</code>	Schreib- und Leseoperationen werden im Binärmodus ausgeführt.

Table: Eröffnungsmodi für Dateien

Die default Werte sind:

- ▶ `ios::in` für `ifstream`
- ▶ `ios::out` | `ios::trunc` für `ofstream`

File-Streams schließen

Wir wissen schon, dass es bei objekt-orientierten Sprachen immer zwei passende Dinge gibt, z.B. Klassen-Konstruktoren und -Destruktoren. So ist zu erwarten, dass es zu einer Methode 'Datei öffnen' (`open()`) auch eine Methode 'Datei schließen' gibt (`close()`) (siehe Tabelle 6.4)

A simple copy function EX06c-io-files

```
1 #include <iostream> // for using cout
2 #include <fstream> // for using ifstream / ofstream
3 #include <string> // for using string
4 using namespace std; // namespace for std functions
5
6 int main()
7 {
8     //-----
9     ifstream input_file; // Instance of class ifstream
10    input_file.open("input_file.txt"); // Open file "text_file.txt"
11    string my_string; // Instance of class string
12    input_file >> my_string; // Reading a string from file
13    cout << my_string.data() << endl; // Output of string to screen
14    //-----
15    ofstream output_file; // Instance of class ifstream
16    output_file.open("output_file.txt"); // Open file "text_file.txt"
17    output_file << my_string; // Writing a string to a file
18    //-----
19    return 0;
20 }
```

Die Ein- `>>` und Ausgabeoperatoren `<<` formatieren die Datentypen (z.B. `int` in der Übung E623) entsprechend den Einstellungen der `fstream` Klasse. Diese Einstellungen können durch Flags verändert werden (siehe nächsten Abschnitt).

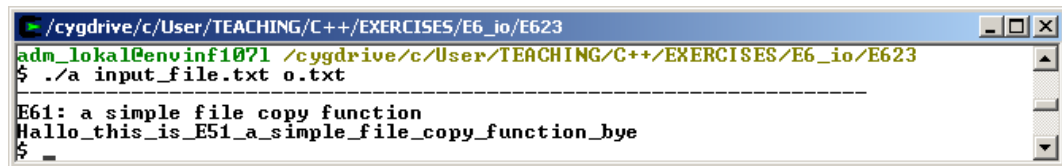
6.2.3 main() Function with parameters

Die main() Funktion kann auch mit einer Parameterliste (int argc, char *argv[]) versehen werden. Die Anzahl der Parameter (argc) wird automatisch erkannt. Mit jedem Leerzeichen in der Tastatureingabe entsteht ein neuer Eingabeparameter (Abb. 3).

```
1 int main(int argc, char *argv[])
2 {
3     ifstream input_file;           // Instance of class ifstream
4     input_file.open(argv[1]);      // Open file, name from cin
5     ofstream output_file;         // Instance of class ifstream
6     output_file.open(argv[2]);    // Open file, name from cin
7     return 0;
8 }
```

6.2.3 main() Function with parameters

Die Benutzung der main Funktion mit Eingabeparametern ist in der folgenden Abbildung zu sehen.



```
/cygdrive/c/User/TEACHING/C++/EXERCISES/E6_io/E623
adm_lokal@envinf1071 /cygdrive/c/User/TEACHING/C++/EXERCISES/E6_io/E623
$ ./a input_file.txt o.txt
-----
E61: a simple file copy function
Hallo_this_is_E51_a_simple_file_copy_function_bye
$ _
```

Figure: Die main Funktion mit Parametern

Daten-Konverter

Ihre Frage nach dem Sinn der Übung EX06d-io-converter ist vollkommen berechtigt, wozu ein Programm schreiben, um eine Datei zu kopieren. Das kann ich doch auch mit dem Windows-Explorer oder mit `cp file1 file2` machen. Richtig, aber genauso funktionieren Kopierprogramme, Windows-Explorer ruft 'unser' Kopierprogramm auf. Wenn wir auch nur kleine Änderungen in unserer Datei vornehmen wollen (z.B. eine laufende Nummer in jede Zeile einfügen), kann uns der Windows-Explorer nicht mehr weiter helfen. Dies ist insbesondere dann ärgerlich, wenn die Datei ziemlich groß ist ... Auch hier sagen sie zu Recht, eine Nummerierung in eine größere Datei einfügen, das kann ich z.B. auch mit EXCEL machen. In der nächsten Übung schreiben wir einen kleinen Konverter, also genau was EXCEL im Hintergrund macht, wenn wir eine neue Spalte einfügen.

Windows Explorer window showing the contents of the folder **DRESDEN** located at **C:\User\TEACHING\C++\LECTURES\DRESDEN**.

The left pane shows the folder structure:

- GEOSYS
- PAPER
- PICTURES
- PRESENTATIONS
- PROJECTS
- PVT
- REVIEWS
- TEACHING
 - C++
 - Copy of SCRIPT
 - EXERCISES
 - http%3a%2f%2fftp.inf.tu-dresden.
 - LECTURES
 - DRESDEN**
 - TUEBINGEN
 - SCRIPT
 - FIGURES
 - WEB
 - OMEFM

The right pane shows a list of files:

Name	Size	Type	Date Modified
L0.ppt	12.113 KB	Microsoft PowerPo...	17.04.2009 08:31
L1.ppt	1.734 KB	Microsoft PowerPo...	17.04.2009 08:26
L2.ppt	1.503 KB	Microsoft PowerPo...	24.04.2009 08:53
L3.ppt	2.066 KB	Microsoft PowerPo...	17.04.2009 08:10
L4.ppt	1.248 KB	Microsoft PowerPo...	08.05.2009 08:50
L5.pdf	332 KB	Adobe Acrobat Doc...	07.05.2009 17:03
L5.ppt	1.176 KB	Microsoft PowerPo...	08.05.2009 11:01
L6.ppt	732 KB	Microsoft PowerPo...	14.05.2009 17:19
L7.ppt	279 KB	Microsoft PowerPo...	24.04.2009 08:55

The image shows a screenshot of the Microsoft Excel application window titled "Microsoft Excel - Book1". The window includes a menu bar with "File", "Edit", "View", "Insert", "Format", "Tools", "Data", "Window", and "Help". A search bar on the right of the menu bar contains the text "Type a question for help". Below the menu bar is a ribbon with various icons for file operations, editing, and data analysis. The main area of the window is a spreadsheet grid with columns labeled A through K and rows numbered 1 through 19. Cell A1 is currently selected and highlighted with a black border. The formula bar above the grid shows "fx" and "A1".

6.2.4 Daten-Konverter

```
1 int main()
2 {
3     //-----
4     ifstream input_file;           // Instance of class ifstream
5     input_file.open("input.txt");  // Open file "text_file.txt"
6     ofstream output_file;         // Instance of class ifstream
7     output_file.open("output.txt"); // Open file "text_file.txt"
8     //-----
9     char line[80];
10    int i=0;
11    while(input_file.getline(line,80)) // Loop condition
12    {
13        output_file << i << " " << line << endl;
14        i++;                          // Incrementor (+1)
15    }
16    //-----
17    return 0;
18 }
```

C++ news tables

Was ist neu bei dieser Übung.

C++ Ding	Was tut's
<code>while()</code>	eine Kontrollstruktur für Schleifen (solange der Ausdruck in () wahr (true) ist wird die Schleife ausgeführt)
<code>i++</code>	der Inkremetor (zählt Eins hoch)

Table: C++ news

File-Streams und Klassen

Wir wollen eine Lesefunktion für die Klasse `CStudent` schreiben. Bevor wir damit beginnen, müssen wir uns Gedanken über eine geeignete Struktur für die Darstellung eines `CStudent` Datensatzes in einer Datei machen. Der Vorschlag für die Strukturierung von Datensätzen ist die Benutzung von Schlüsselwörtern zur Trennung von Datenblöcken, z.B.

```
#STUDENT
  $NAME_FIRST
    James
  $NAME_LAST
    Bond
  . . .
#STOP
```

File-Streams und Klassen

Wir benutzen zwei verschiedene Symbole für Schlüsselwörter:

- ▶ keyword `#` : zur Trennung von Datensätzen für eine Instanz von `CStudent`,
- ▶ subkeyword `$` : zur Identifizierung der einzelnen Daten für die `CStudent` Instanz.
- ▶ `#STOP` zeigt das Ende der Datenbasis an. (Eigentlich wäre dies nicht nötig, da das Dateiende auch mit dem Parameter `eof` (end-of-file) abgefragt werden kann. Wir werden aber sehen, dass mit `#STOP` einiges einfacher zu programmieren ist.)

6.3 CStudent::Read() E63

```
1 ios::pos_type CStudent::Read(istream& input_file)
2 {
3     //-----
4     string input_line;
5     char buffer[256]; // MAX_LINE
6     ios::pos_type position;
7     //-----
8     while(true)
9     {
10        position = input_file.tellg();
11        input_file.getline(buffer,256);
12        input_line = buffer;
13        if(input_line.size()<1) // empty line
14            continue;
15        // Dealing with keywords
16        if(input_line.find('#')!=string::npos) // keyword found
17            return position;
18        ...}
```

Listing: Title

6.3 main() E63

```
1 #include <iostream> // for using cout
2 #include <fstream> // for using ifstream / ofstream
3 #include <string> // for using string
4 #include "student.h" // for using CStudents
5 using namespace std; // for std functions
6
7 int main()
8 {
9     //-----
10    // File handling
11    ifstream input_file; // ifstream instance
12    input_file.open("data_set.txt");
13    if(!input_file.good()) // Check is file existing
14    {
15        cout << "! Error in STD::Read: file could not be opened" << endl;
16        return 0;
17    }
18    input_file.seek(0L,ios::beg); // Rewind file
19    //-----
20    CStudent* m_std = new CStudent(); // CStudent instance
21    m_std->Read(input_file);
22    //-----
23    input_file.close();
24    return 0;
25 }
```

C++ news tables

Die main Funktion besteht aus zwei Teilen, dem File-Handling und dem Aufruf der Lesefunktion. Beim File-Handling wird der stream aus der Datei data_set.txt geöffnet, anschließend erfolgt der Test, ob das File erfolgreich geöffnet werden konnte; wenn nicht, wird die main Funktion sofort beendet.

Was ist neu bei dieser Übung.

C++ Ding	Was tut's
<code>ifstream& input_file</code>	eine Reference auf ein Objekt wird in Kapitel 7 ausführlich abgehandelt

Table: C++ news

fstream Methoden

Methode	Funktion
<code>open()</code>	öffnet die Datei
<code>good()</code>	testet erfolgreiche Öffnung der Datei
<code>seekg(pos, ios::beg)</code>	geht zur Position pos in der Datei
<code>seekg(0L, ios::beg)</code>	spoolt zum Dateianfang zurück
<code>tellg()</code>	merkt sich die aktuelle Position im stream
<code>getline(buffer, 256)</code>	holt eine Zeile der Länge 256 (Zeichen) aus dem stream und kopiert diese in buffer
<code>close()</code>	schließt Datei
<code>>></code>	Eingabeoperator für Dateien
<code><<</code>	Ausgabeoperator für Dateien

Table: ifstream Methoden

Die string Auswertung spielt bei der Lesefunktion eine wichtige Rolle, daher werden wir uns in der nächsten Vorlesung mit der string Klasse beschäftigt.

Testfragen

1. Was ist die Basis-Klasse für alle Ein- und Ausgaben in C++ ?
2. Was sind die C++ Klassen für das Lesen und Schreiben von Dateien ?
3. Welchen Include benötigen wir für das Arbeiten mit I/O File-Klassen ?
4. Was sind die Standard-Flags für File-Streams (Lesen und Schreiben) ?
5. Mit welchem Flag können wir zu schreibende Daten an eine existierende Datei anhängen ?
6. Was ist der Unterschied zwischen ASCII- und Binär-Formaten ?
7. Mit welchem Flag können wir Daten in einem Binär-Format schreiben ?
8. Mit welcher Anweisung wird ein File geöffnet ?
9. Mit welcher Anweisung wird ein File geschlossen ?
10. Was bewirken die Stream-Operatoren << und >> ?
11. Wie können wir mit Dateinamen in unserem Hauptprogramm `main(...)` arbeiten ?
12. Welche Anweisung benötigen wir für die Erzeugung einer Instanz für einen Eingabe-Strom ?
13. Welche Anweisung benötigen wir für die Erzeugung einer Instanz für einen Ausgabe-Strom ?
14. Für die Erstellung einer Datenbank ist es wichtig einzelnen Datensätze zu trennen. Wie können wir soetwas in der Datenbank-Datei bewerkstelligen ?
15. Ist es wichtig das Ende einer Datenbank-Datei, z.B. mit einem Schlüsselwort #STOP, zu markieren ?
16. Mit welcher Abfrage können wir prüfen, ob die Öffnung einer Datei erfolgreich war ?
17. Mit welcher Anweisung können wir die aktuell gelesene Position in einer geöffneten Datei abfragen ?
18. Mit welcher Anweisung können wir zu einer bestimmten Position in einer geöffneten Datei springen ?
19. Mit welcher Anweisung können wir eine komplette Zeile aus geöffneten Datei auslesen ?

Quellcode und Compiler

Hier wartet schon die erste Überraschung auf uns: `pwd` wird nicht erkannt und wir werden aufgefordert eine Zeit einzugeben (Abb. 4).



```
c:\user\teaching\c++\exercises\e6_io\msvcp\debug\msvcp1.exe
'pwd' is not recognized as an internal or external command,
operable program or batch file.
The current time is: 12:48:40,49
Enter the new time: _
```

Figure: Das Debug Verzeichnis

Was ist passiert ?

- ▶ `pwd` ist ein Linux-Befehl, den kennt der Windows-Compiler nicht.
- ▶ `time` gibt es auch als DOS-Befehl, hat aber eine ganz andere Bedeutung: nicht Ausgabe der Zeit sondern Zeit ändern.

Wir sehen also, dass unser Quellcode von verschiedenen Compilern unterschiedlich interpretiert wird.