Hochschule Esslingen
University of Applied Sciences

Objektorientierte Programmierung mit C++:

Ein- und Ausgabe mit Streams

Profs. M. Dausmann, D. Schoop, A. Rößler, A. Freymann

Fakultät Informationstechnik, Hochschule Esslingen

Agenda

- Ausgabe in C++
- Formatierung der (Ein- und) Ausgabe
- Streamkonzept
- Streams für Bildschirm und Tastatur
- Standard-Namensraum
- Schreiben und Lesen mit Dateien

Ausgabe in C++

```
// Basic IO
#include <iostream>

int main() {
   std::cout << "Hello World!" << std::endl;
   return 0;
}
```

Formatierung der Ein- und Ausgabe

- Der Operator "<<" akzeptiert als zweiten Operanden auch einen Strommanipulator.
- Die Wirkung dieser Operation ist die Manipulation des linken Operanden, also des Ausgabestroms.
- Ein Ausgabestrom kennt verschiedene Betriebsarten, die bestimmen, wie Werte dargestellt werden.
  - Dec, hex …

```
cout << Strommanipulator << “Text” << std::endl;
```

Formatierung der Ein- und Ausgabe

- Die Formatierung wird gesteuert über
  - Methoden, Beispiele: **`width()`** und **`fill()`**
  - Flags, Beispiele: **`left`**, **`right`**, **`internal`**, **`scientific`**
  - Manipulatoren ohne Parameter, Beispiele: **`endl`**, **`hex`**, **`oct`**
  - Manipulatoren mit Parameter, Beispiel: **`setbase(n)`**
- Bibliotheken sind **`<ios>`** und **`<iomanip>`**.
- Die komplette Liste ist umfangreich und bei Bedarf nachzuschlagen.
- Wichtig sind die Formatierungen hauptsächlich für die Ausgabe.

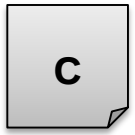
Beispiele

C	C++	Ausgabe
`printf("Hello");`	`cout << "Hello";`	`Hello`
`printf("%d", 117);`	`cout << dec << 117;`	`117`
`printf("%o", 117);`	`cout << oct << 117;`	`165`
`printf("%X", 117);`	`cout << hex << 117;`	`75`

`printf("%+9.5f", 1.23);`	`cout.precision(5);`
	`cout.width(9);`
	`cout << showpos << showpoint << fixed << 1.23;`
		`+1.23000`

`printf("\n");`	`cout << endl;`	*neue Zeile*

Manipulatoren für die Ausgabe



*formatting_*

showpos	positive Zahlen werden mit Vorzeichen ausgegeben
noshowpos	positive Zahlen werden ohne Vorzeichen ausgegeben
uppercase	bei der hexadezimalen Darstellung werden nur Großbuchstaben verwendet
nouppercase	bei der hexadezimalen Darstellung werden nur Kleinbuchstaben verwendet
showpoint	der Dezimalpunkt wird immer angezeigt.
noshowpoint	abschließende Nullen hinter einem Dezimalpunkt werden nicht angezeigt.
showbase	Integerwerte bekommen die Basis vorangestellt
noshowbase	Integerwerte bekommen die Basis nicht vorangestellt
fixed	Darstellung als Festpunktzahl (Gegenteil zu **scientific**)
scientific	Darstellung in exponentieller Notation
setprecision(n)	setzt die Anzahl der gezeigten Ziffern (Genauigkeit) von Fließkommazahlen auf n
boolalpha	boolesche Werte werden als „true“ oder „false“ anstelle von „0“ und „1“ ausgegeben.
setw(n)	setzt die Feldbreite auf n
left	linksbündige Ausgabe im Feld
right	rechtsbündige Ausgabe im Feld
internal	zentrierte Ausgabe im Feld
setfill(c)	c wird als Füllzeichen verwendet

Beispiel mit formatierter Ausgabe

```
#include <iostream>
#include <iomanip> // für setprecision, setw, setfill
using namespace std;

int main() {
  int i = 1234;
   cout << "Dec: " << i << " Hex: " << hex << i
        << " Oct: " << oct << i << endl;
  cout.precision(5);
  cout << setfill('*') << setw(12) << showpos << fixed
       << i / 100.0 << endl;
  cout << noshowpos << scientific << i / 100.0 << endl;
  return 0;
}
```

```
Ausgabe:
Dec: 1234 Hex: 0X4D2 Oct: 02322
***+12.34000
1.23400E+001
```

Das Streamkonzept

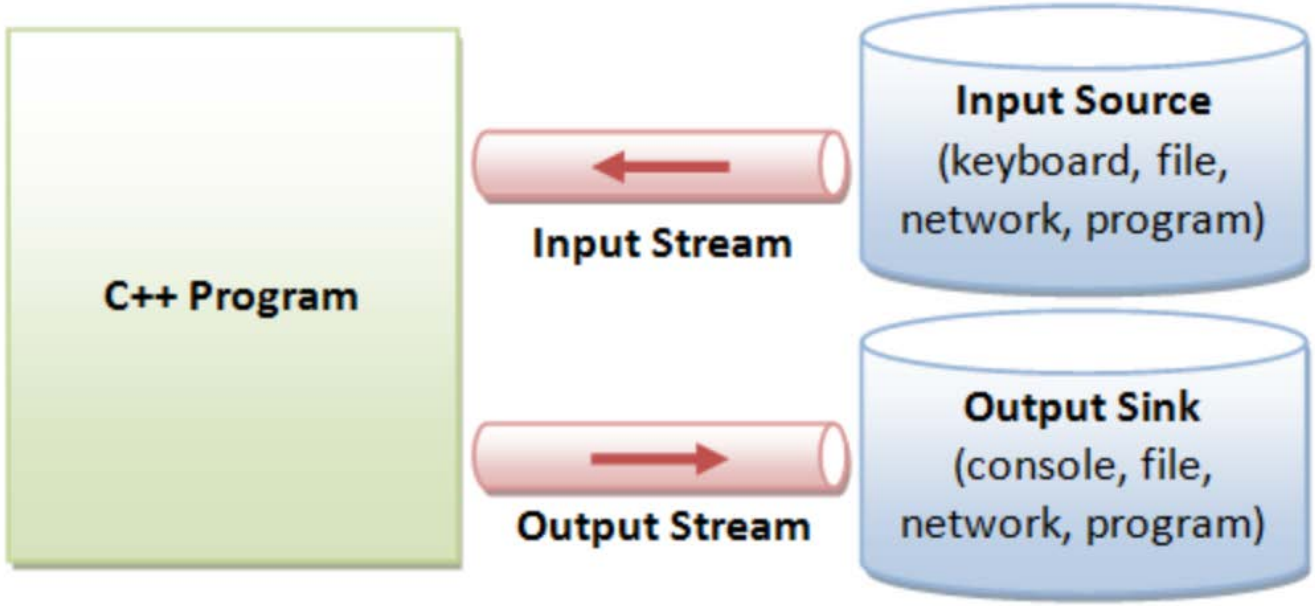
- Der Begriff Datei (File) ist nicht auf eine physikalische Datei auf einem Datenträger beschränkt, sondern steht auch für Ein- und Ausgabekanäle und Geräte.
- Datenstrom (Stream) bezieht sich auf Ein-/Ausgabe von allen Dateien, d.h. auch von Geräten.

- In C: **`stdout, stdin, sterr; <stdio.h>`**
  Daten werden mit **`printf`**, **`scanf`** etc. bearbeitet.

- In C++: **`cin, cout, cerr, clog; <iostream>`**
  Streams sind Objekte zur Eingabe und Ausgabe.

Das Streamkonzept

http://www.ntu.edu.sg/home/ehchua/programming/cpp/cp10_io.html



Internal Data Formats:

- Text: `char`, `wchar_t`
- `int`, `float`, `double`, etc.

External Data Formats:

- Text in various encodings (US-ASCII, ISO-8859-1, UCS-2, UTF-8, UTF-16, UTF-16BE, UTF16-LE, etc.)
- Binary (raw bytes)

Streams für Bildschirm und Tastatur

- („Bitshift-“)Operatoren >> und << für Ein- und Ausgabe

```
cin >> Eingabevariable;          // Standardeingabe
cout << Ausgabevariable;         // Standardausgabe
cerr << Ausgabevariable;         // ungepufferte Fehlerausgabe
clog << Ausgabevariable;         // gepufferte Fehlerausgabe
```

- Beispiele

```
std::cout << "Kalender";
std::cout << "Heute ist der " << v_tag;
std::cout << ". März 2013. " << std::endl;
```

Beispiel mit formatierter Eingabe

```
#include <iostream>
using namespace std;

int main() {
  int i = 1;
  cout << showbase << uppercase;
  cout << "Eingabe: ";
  cin >> hex >> i;
  cout << "Dec: " << dec << i << endl;
  cout << "Hex: " << hex << i << endl;
  cout << "Oct: " << oct << i << endl;
  return 0;
}
```

```
Durchlauf 1:

Eingabe: FFZ
Dec: 255
Hex: 0XFF
Oct: 0377

Durchlauf 2:

Eingabe: 0x10x1
Dec: 16
Hex: 0X10
Oct: 020
```

Stream-Status

- Jeder Stream hat einen Status für die Anzeige von Fehlern und außergewöhnlichen Zuständen.

```
eof()  // gibt true bei end-of-file (ios::eofbit gesetzt)
fail() // gibt true bei Fehler (ios::failbit oder ios::badbit)
bad()  // gibt true bei fatalem Fehler (ios::badbit gesetzt)
good() // gibt true, wenn Stream OK ist (ios::goodbit)
rdstate()     // gibt die gesetzten Flags zurück (readstate)
clear()       // löscht alle Flags
clear(flags) // löscht alle Flags und setzt flags als Zustand
setstate(flags)     // setzt zusätzlich flags als Zustand
```

Status-Bits

- Es gibt die folgenden Status-Bits (deklariert in **ios**):

```
ios::goodbit      // alles ok
ios::eofbit       // end of file
ios::failbit      // leichter Fehler, Fortsetzung möglich
ios::badbit       // keine Garantie für Fortsetzung

int s = cin.rdstate(); // Rückgabe Error-Flags
if      (s == ios::goodbit){ /* Alles OK */ }
else if (s == ios::failbit){ /* vielleicht Zeichen verloren
                                gegangen */}
else if (s == ios::badbit) { /* event. Formatierfehler */}
else if (s == ios::eofbit) { /* End-of-File */}
```

Status-Bits

http://www.cplusplus.com/reference/ios/ios/good/

iostate value (member constant)	indicates	functions to check state flags				
		good()	eof()	fail()	bad()	rdstate()
goodbit	No errors (zero value iostate)	**true**	**false**	**false**	**false**	goodbit
eofbit	End-of-File reached on input operation	**false**	**true**	**false**	**false**	eofbit
failbit	Logical error on i/o operation	**false**	**false**	**true**	**false**	failbit
badbit	Read/writing error on i/o operation	**false**	**false**	**true**	true	badbit

Status-Bits – Beispiel

Statusbits bei fehlgeschlagenem Einlesen:

```
void showstate(istream & s) {
   ios_base::iostate state = s.rdstate(); cout << boolalpha;
   cout << "eof  = " << (state != 0) << endl;
   cout << "fail = " << (state != 0 ) << endl;
   cout << "bad  = " << (state != 0) << endl;
}
```

```
int main() {
  int i = 1;
  showstate(cin);
  cout << "Eingabe: ";
  cin >> i;
  showstate(cin);  }
```

```
Ausgabe eines Durchlaufs:

eof = false
fail= false
bad = false

Eingabe: ZZ
eof = false
fail= true
bad = false
```

Standard-Namensraum

- Die Beispiele zeigten bisher den Gebrauch von **`std::cout`** und **`std::endl`**.

- C++ definiert die gesamte Standard-Bibliothek in einem eigenen Namensraum (namespace), der **`std`** genannt wird.
- Alle Bezeichner gehören zu diesem Namensraum und müssen außerhalb dieses Bereiches mit **`std::`** qualifiziert werden.
- Der Teil **`std::`** kann weggelassen werden, wenn zu Beginn des Programms der Namensraum std (Standard) ausgewählt wird.

Nutzen des Standard-Namensraumes (1)

```
#include <iostream>

int main()
{     // Mit Qualifizierung
      std::cout << "Hallo, Welt! " << std::endl;
      {
            using namespace std;
            // Jetzt geht es in diesem Block
            // ohne Qualifizierung
            cout << "Hallo, Welt! " << endl;
      }
      return 0;
}
```

Nutzen des Standard-Namensraumes (2)

```
#include <iostream>
// Namensraum gleich zu Beginn:
using namespace std;

int main()
{     // Jetzt geht alles ohne Qualifizierung
      cout << "Hallo, Welt! " << endl;
      {
            cout << "Hallo, Welt! " << endl;
      }
      return 0;
}
```

Files und Streams

- Klassen für die Arbeit mit Dateien:
  - **`ifstream`** zum Lesen aus einer Datei
  - **`ofstream`** zum Schreiben in eine Datei
  - **`fstream`** zum Lesen und Schreiben
- Dazu wird die Bibliothek **`<fstream>`** benötigt.
- Das Öffnen einer Datei entspricht dem Erzeugen eines Objektes dieser Klassen.
- Der Konstruktor erhält den Namen der Datei und ggf. den Öffnungsmodus:

```
ofstream OutFile1 ("output.txt");
ofstream OutFile2 ("test.txt", ios::app);
```

Öffnen einer Datei

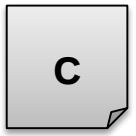
Modus	Bedeutung	Eröffnung zum
`in`	input	Lesen ab Dateianfang (Default für **`ifstream`**)
`ate`	at end	Positionieren auf Dateiende
`binary`	binary	Bearbeiten ohne Aufbereitung
`out`	output	Schreiben ab Dateianfang (Default für **`ofstream`**)
`app`	append	Schreiben hinter Dateiende
`trunc`	truncate	Überschreiben des alten Inhalts

Beispiele zum Öffnen

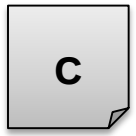
```
ofstream datei("output.tmp");
if (!datei) cerr << "Kann Datei nicht öffnen" << endl;


fstream datei ("output.txt", ios::out | ios::trunc);
datei.close (); // leere Datei jetzt vorhanden

datei.open ("output.txt", ios::in | ios::out);
                  // Datei zum Schreiben und Lesen öffnen
```

Beispiele zum Arbeiten mit Dateien

```
int main (void) {
    char ch;
    ifstream FromFile ("input.txt");
    ofstream ToFile ("output.tmp");
    while (FromFile.get(ch)){// einfache Kopierschleife
      ToFile.put(ch);
    };
    if ( !FromFile.eof () || ToFile.bad () )
        cerr << "Fataler Fehler aufgetreten" << endl;
}
```

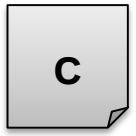


spielerschreiben



spielerschreiben

Text- und Binärdateien



readfrombinarfile

- Den bisherigen Beispiele lagen Textdateien zu Grunde.
- Textdateien können betrachtet und gedruckt werden. Sie können daher auch schön formatiert werden.
- Textdateien sind portabel.
- Bei jedem Schreiben und Lesen findet aber ein Umwandlung der binären Darstellung im Speicher in die Textdarstellung der Datei statt.
- Es gibt auch die Möglichkeit, die binäre Darstellung aus dem Speicher direkt in eine Datei zu schreiben und umgekehrt. Man spricht dann von Binärdateien.
- Binärdateien sind nicht portabel, da es unterschiedliche binäre Darstellungen auf unterschiedlichen Plattformen gibt.

Zusammenfassung

- Für die Ein- und Ausgabe wird in C++ ein objekt-orientierter Ansatz verwendet: Streams sind Objekte, man braucht keine File-Handles wie in C.
- Es gibt Standardstreams **`cin`**, **`cout`**, **`cerr`** und **`clog`**.
- Zur Ein- und Ausgabe stehen die Operatoren **`<<`** und **`>>`** zur Verfügung. Diese können für eigene Klassen überladen werden.
- Zur Formatierung gibt es Manipulatoren, Methoden und Flags.
- Für die byteweise (satzweise) Ein- und Ausgabe stehen die Methoden **`read()`** und **`write()`** zur Verfügung.
- Man kann in einer Datei an die Stelle positionieren, ab der gelesen bzw. geschrieben werden soll.

Zusammenfassung

- C++ nutzt ein Streamkonzept. Streams sind Objekte.
- Standard-Streams (u.a): `cin` und `cout`.
- Eingabeoperator: `>>` Ausgabeoperator: `<<`
- Bibliothek: `iostream` (in C++ bei `include` ohne `.h`)
- Formatierung mittels Manipulatoren, Methoden (und Flags)
- Elemente der Bibliothek gehören zum Namensraum `std`.
- Ein Namensraum wird einem Bezeichner mit dem Scope-Operator(`::`) vorangestellt, beispielsweise: `std::cout`.
- Der Namensraum-Zusatz kann entfallen nach einer `using`-Deklaration: `using namespace std;`