5. Ein- und Ausgabe

5.1.cout und cin

5.1.1. cout

Wie wir schon wissen, befindet sich der Ausgabestream *cout* in der Quelldatei <iostream>, im namespace (Namensbereich) *std*. Hier noch einmal ein kleines Beispiel, das schon allen vertraut aussehen müsste:

Beispiel 1

```
// Datei iostream einbinden
#include <iostream>
// using Directive
using namespace std;
int main ( )
{
    cout [<< ausgabe1...];
}
```

Man kann statt

Beispiel 2

```
using namespace std;
```

auch folgendes schreiben:

Beispiel 3

```
using std::cout;
```

Man sollte eigentlich die erste Variante vermeiden, falls man sich nicht ganz im klaren ist, was sie tut. Der Namensbereich *std* ist sehr gross und beinhaltet auch andere Klassen. Deshalb kann es zu Zweideutigkeiten führen. In unseren kleinen Beispielen ist es aber ungefährlich.

5.1.1.1. Ausgabe von Texten und Zeichen (Wiederholung)

Hier ein paar Beispiele, wie man „Hallo" ausgeben kann, auf verschiedene Weisen:

Beispiel 4

```
#include <iostream>
using std::cout;
void main() {
   cout << „Hallo! \n";
   cout << „Hallo!" << endl;
   cout << „Hall";
   cout << „o! \n";
   cout << ´H´ << ´a´ << ´l´ << ´l´ << ´o´ << „\n";
}
```

5.1.1.2. Spezialzeichen

Das Spezialzeichen „\n" haben wir schon gesehen. Hier sind noch ein paar oft benutzt Steuerzeichen (auch Escape-Sequenzen genannt):

Tabelle 1

ASCII	Kurz	Bezeichnung	Darstellung
7	BEL	Alarm (Warnton)	'\a'
8	BS	Rückschritt (Backspace))	'\b'
9	HT	Horizontal-Tabulator	'\t'
10	LF	Zeilenvorschub (Line Feed)	'\n'
11	VT	Vertikal-Tabulator	'\v'
12	FF	Seitenvorschub (Form Feed)	'\f'
13	CR	Wagenrücklauf (Carriage Return)	'\r'
34	"	Anführungszeichen	'\"'
92	\	Backslash	'\\'
96	'	Apostroph ("Hochkomma")	'\''

5.1.1.3. Ausgabe von Werten

Wir haben das stream *cout* schon auch für Werte benutzt. Hier ein kleiner Beispiel zur Erinnerung:

Beispiel 5

```
// IO-Stream Datei einbinden
#include <iostream>
// cout aus std Namensraum
using std::cout;

// Das Programm
int main ( )
{
    // Auszugebende Daten
    int anyVal = 10;
    char byteVal = 66;
    float floatVal = 1.23f;

    // Gemischte Ausgabe von Text und Daten
    cout << "int-Datum: " << anyVal <<
            " float-Datum: " << floatVal <<
            '\n';
    // Ausgabe von char-Daten
    cout << "char-Datum: " << byteVal << '\n';
    cout << "char-Datum als Wert: " <<
            (int)byteVal;
    // und fertig!
}
```

Aber damit sind die Ausgabemöglichkeiten von *cout* natürlich noch nicht ausgeschöpft. In manchen Fällen kann es durchaus sinnvoll sein, Ganzzahlen in einem anderen Zahlensystem als dem standardmäßigen Dezimalsystem darzustellen. Um Ganzzahlen als hexadezimal oder oktal Zahl auszugeben, stellt C++ so genannte Manipulatoren zur Verfügung, die einfach in den Ausgabestream

eingefügt werden. Folgende Manipulatoren stellen die Ausgabe von ***Ganzzahlen*** auf das hexadezimale, oktale oder dezimale Zahlensystem um:

Tabelle 2

Manipulator	Ausgabebasis
std::dec	dezimal (Basis 10)
std::hex	hexadezimal (Basis 16)
std::oct	oktal (Basis 8)

Und hier ein Beispiel:

Beispiel 6

```
// IO-Stream Datei einbinden
#include <iostream>
// cout aus std Namensraum
using std::cout;
// Das Programm
int main ( )
{
    // Auszugebende Daten
    int var = 10;
    // Zahlenbasis mit ausgeben
    cout << std::showbase;
    cout << " Dez: " << var;
    cout << " Hex: 0x" << std::hex << var;
    cout << " Okt: " << std::oct << var;
    cout << " Zahl: " << var;
    // und fertig!
}
```

Eine einmal eingestellte Zahlenbasis für die Ausgabe bleibt solange aktiv, bis sie explizit umgestellt wird.

Außer den vorhin vorgestellten Manipulatoren *dec*, *hex* und *oct* stehen noch die Manipulatoren *setbase(...),. setw(...), setfill(...)* und *setprecision(...)* zur Verfügung. Wenn Sie einen dieser Manipulatoren einsetzen, müssen Sie die Datei *iomanip* mittels *#include* zusätzlich einbinden.

Beginnen wir mit dem Manipulator *setbase(n).* Er bietet prinzipiell nichts Neues. *setbase(n)* dient, genauso wie *dec, hex* und *oct*, zur Einstellung des Zahlensystems für die Ausgabe. Der Parameter *n* gibt das entsprechende Zahlensystem an und kann die Werte 8, 10 und 16 für das Oktal-, Dezimal- und Hexadezimal-System annehmen. Alle anderen Werte stellen die Ausgabe wieder auf das Dezimalsystem zurück. Hier ein Beispiel:

Beispiel 7

```
// IO-Stream und Manipulatoren einbinden
#include <iostream>
#include <iomanip>

using std::cout;
using std::setbase;
// Das Programm
int main ( )
{
    // Auszugebendes Datum
    int var = 10;
    // Ausgabe als Hex-Zahl
    cout << setbase(16) << var << "->";
    // Ausgabe als Oktal-Zahl
    cout << setbase(8) << var << "->";
    // Ausgabe wieder als Dezimal-Zahl
    cout << setbase(10) << var;
}
```

Mit dem Manipulator *setw(n)* kann die minimale Breite des ***nächsten*** Ausgabefeldes eingestellt werden. Dieser Manipulator erhält als Parameter *n* die Anzahl der minimalen Ausgabestellen. Benötigt die Ausgabe mehr Stellen als angegeben wurden, so wird das Ausgabefeld entsprechend vergrößert, d.h. die Zahl wird immer vollständig dargestellt. Beispiel:

Beispiel 8

```
// IO-Stream Datei und
// Manipulatoren einbinden
#include <iostream>
#include <iomanip>

using std::cout;
using std::setw;

// Das Programm
int main ( )
{
    // Auszugebendes Datum
    int var = 10;
    // Datum ohne min. Feldbreite ausgeben
    cout << ":" << var << ":";
    // Min. Feldbreite für ':' auf 4 setzen
    cout << setw(4) << ":" << var << ":";
    // Min. Feldbreite für var auf 4 setzen
    cout << ":" << setw(4) << var << ":";
    // Ausgabe wieder ohne min. Feldbreite
    cout << ":" << var << ":";
}
```

Achtung! Die Angabe von *setw(n)* gilt nur für die unmittelbar nachfolgende Ausgabe (siehe Beispiel)!

Ist die Breite des Ausgabefeldes größer als tatsächlich Stellen benötigt werden, so können nicht belegte Stellen mit einem beliebigen Zeichen ausgefüllt werden. Die Festlegung des Füllzeichens erfolgt mit dem Manipulator *setfill(n).* Er erhält als Parameter *n* das zu setzende Füllzeichen. Beachten Sie bitte, dass Sie hier ein Zeichen übergeben müssen das in einfache Hochkomma eingeschlossen wird. Wie Sie vielleicht schon erraten haben, ist das Standard-Füllzeichen das Leerzeichen. Ein einmal eingestelltes Füllzeichen bleibt solange aktiv, bis es wieder umgesetzt wird. *setfill(n)* wirkt ebenfalls auf alle Datentypen.

Im Beispiel rechts wurde der *setfill(...)* Manipulator einmal mit der vollen Qualifikation *std::setfill(...)* verwendet.

Beispiel 9

```
// IO-Stream Datei und
// Manipulatoren einbinden
#include <iostream>
#include <iomanip>

using std::cout;

// Das Programm
int main ( )
{
    // Auszugebendes Datum
    int var = 10;
    // Ausgaben mit Füllzeichen '#'
    cout << ":" << std::setw(4) <<
            std::setfill ('#')  <<
            var << ":";
    cout << ":" << std::setw(3) <<
            var << ":";
    // Füllzeichen wieder zurückstellen
    cout << ":" << std::setw(4) <<
            std::setfill(' ') <<
            var << ":";
}
```

Für die Ausgabe von Gleitkommazahlen können Sie über den Manipulator *setprecision(...)* die Anzahl der auszugebenden Stellen (ohne einen eventl. Exponenten und Vorzeichen) einstellen. Überschreitet die Anzahl der Vorkommastellen die mit *setprecision(...)* eingestellte Stellenanzahl, so wird automatisch auf Exponentialdarstellung umgestellt (erste Ausgabe rechts). Wird über die Methode *setf(...)* (siehe Vertiefung unten) die Darstellung mit oder ohne Exponenten erzwungen, so legt *setprecision(...)* die Anzahl der Nachkommastellen fest.

Und auch die einmal eingestellte Anzahl der auszugebenden Stellen bleibt solange gültig, bis sie erneut umgesetzt wird. Standardmäßig beträgt die Genauigkeit für die Ausgabe 6 Stellen.

Beispiel 10

```
// IO-Stream Datei und
// Manipulatoren einbinden
#include <iostream>
#include <iomanip>

using std::cout;
using std::setprecision;
using std::ios;

// Das Programm
int main ( )
{
    // Auszugebende Daten
    double var1 = 40000.0/3.0;
    double var2 = 4.0/3.0;
    // Ausgabe auf 4 Stellen begrenzen
    cout << setprecision(4);
    // Normale Ausgabe
    cout << var1 << ' ' << var2 << "->";
    // Ausgabe immer ohne Exponenten
    cout.setf(ios::fixed,ios::floatfield);
    cout << var1 << ' ' << var2;
}
```

5.1.2. cin

Den Stream *cin* haben wir auch schon für Eingaben von der Seite des Benutzers angewandt. Hier ein kleiner Beispiel zur Erinnerung:

Beispiel 11

```
// Datei einbinden
#include <iostream>
using namespace std;
// Variablen definieren
short var1;
long var2;
// Hauptprogramm
int main ( )
{
    ....
    // 2 Daten einlesen
    cin >> var1 >> var2;
    ....
}
```

Standardmäßig werden alle numerischen Eingaben im Dezimalformat erwartet. Sie können jedoch mit den bekannten Manipulatoren *dec*, *hex* und *oct* eine andere

Zahlenbasis für die Eingabe einstellen. Die eingestellten Zahlenbasen für *cout* und *cin* arbeiten unabhängig voneinander. Beispiel:

Beispiel 12

```
short var;
cin >> hex >> var;
```

Da beim Eingabestream *cin* die einzelnen Eingaben durch Leerzeichen voneinander getrennt werden, können Eingaben die selbst Leerzeichen enthalten nicht direkt eingelesen werden. Soll eine komplette Zeile eingelesen werden, so kann hierfür die *cin*-Methode *getline(...)* verwendet werden. *getline(...)* erhält im ersten Parameter die Adresse des *char*-Feldes übergeben, in dem die Eingabe abgelegt werden soll. Der zweite Parameter definiert die maximale Anzahl der einzulesenden Zeichen plus 1. Wenn Sie z.B. maximal 10 Zeichen einlesen wollen, so müssen Sie hier den Wert 11 übergeben (und auch das *char*-Feld mit mindestens 11 Elementen definieren!). Der Grund für dieses Verhalten ist, dass *getline(...)* die Eingabe wiederum als String ablegt, und der wird bekanntermaßen immer mit einer binären 0 abgeschlossen.

Beispiel 13

```
#include <iostream>
using namespace std;
const int SIZE = 81;
int main ()
{
    // Feld zur Aufnahme der Eingabe def.
    char array[SIZE];
    // Hinweis ausgeben
    cout << "Bitte eine Zeile eingeben: ";
    // Zeile einlesen, aber max. SIZE Zeichen
    cin.getline(array, SIZE);
    // Zur Sicherheit 0 anfuegen
    array[SIZE-1] = 0;
    // Restliche Eingabe ueberspringen
    cin.seekg(0,ios::end);
    // Eingabe wieder ausgeben
    cout << "Eingabe war: " << array <<
             endl;
}
```

5.2.file streams

Bis jetzt haben wir nur Ein- und Ausgabe auf dem Dsiplay gesehen. Man kan aber natürlich sie umleiten, z.B. auf eine Datei. Das funktioniert wieder mit einem Stream. Es existieren noch andere Streams (ausser auf dem Standard I/O und Dateien), die wir aber im Kurs nicht besprechen werden. Sie funktionieren alle ähnlich.

Wir sehen uns zwei Arten von Dateien an: Textdateien und Binärdateien.

5.2.1. Textdateien

Textdateien bestehen aus ASCII Zeichen und können von jedem Texteditor bearbeitet werden (wie z.B. Notepad, UltraEdit usw.)

Achtung! Diese Dateiform ist die einzige Möglichkeit Daten zwischen unterschiedlichen Plattformen auszutauschen. Das Dateiformat der im Anschluss erwähnten Binärdatei ist nicht standardisiert.

5.2.2. Binärdateien

Im Gegensatz dazu sind Binärdateien Dateien, in denen Daten in binärer Form, also in der Form, wie sie im Speicher des Rechners liegen, abgelegt sind. Diese Daten können in der Regel nicht mit einem Editor bearbeitet werden, da Editoren nur Dateien mit ASCII-Zeichen sinnvoll darstellen können. Der Aufbau einer Binärdatei ist, wie bereits erwähnt, aber nicht standardisiert, d.h. unterschiedliche Systeme können Daten unterschiedlich ablegen.

Bemerkung. Wenn Sie viele numerische Daten verarbeiten müssen und **nicht** auf den Datenaustausch mit anderen System angewiesen sind, so legen Sie die Daten am besten innerhalb einer Binärdatei ab. Dies kann zu erheblichen Platzeinsparungen führen. Nehmen wir einmal an, Sie müssen einen 4-stelligen *short*-Wert 1000-mal in einer Datei ablegen. In einer Textdatei benötigen Sie dafür 1000*5 Bytes, gleich 5000 Bytes. Das zusätzliche 5. Byte wird als Trennzeichen zwischen den einzelnen Werten benötigt damit die Daten später auch wieder eingelesen werden können. Die gleiche Datenmenge innerhalb einer Binärdatei belegt dagegen nur 1000*2 Bytes, gleich 2000 Bytes, da ein *short*-Wert 2 Bytes benötigt und keine Trennzeichen notwendig sind (wie Sie nachher gleich sehen werden).

Einzelheiten über Binärdateien werden wir in unserem Kurs nicht behandeln.

5.2.3. Definition und Anwendung eines Stream-Objekts

Um einen Stream-Objekt zu benutzen, braucht man folgende Schritte:

- ein Stream- Objekt definieren
- das Objekt mit einer Datei verbinden
- die Datei bearbeiten (lesen/schreiben)
- die Datei-Verbindung aufheben
- das stream-Objekt löschen

5.2.3.1. Ausgabestream Objekt definieren und mit einer Datei verbinden

Die Definition des Objekts ist sehr einfach. Objekte behandeln wir später, jetzt kann man sich unter einem Objekt ertsmal eine Variable vorstellen:

Beispiel 14

```
ofstream myFile;
```

Die Verbindung zwischen der Datei und dem Stream findet mit der folgenden Zeile statt:

Beispiel 15

```
void ofstream::open(const char *pFName, ios::openmode mode=ios::out);
```

Dabei ist der openmode ein Modus des Objekts, der mehrere Zustände annehmen kann. Hier ist eine Übersicht:

Tabelle 3

Modus	**Bedeutung**
ios::out	Öffnen einer Datei zum Schreiben (default).
ios::trunc	Öffnet eine Datei, wobei der ursprüngliche Inhalt wird verworfen (default).
ios::ate	Öffnen einer Datei und positionieren des Schreibzeigers auf das Dateiende. Wird der Schreibzeiger neu positioniert (*seekp(...)*), so werden die Daten an der neuen Position eingefügt.
ios:app	Öffnen einer Datei und positionieren des Schreibzeigers auf das Dateiende. Die neuen Daten werden immer an die Datei eingefügt, unabhängig davon oder der Schreibzeiger inzwischen neu positioniert wurde.
ios::binary	Öffnet einer Datei im Binärmodus.

Man kann den Modus auch verwerfen – dann öffnet der Stream die Datei in einem Textmodus.

Man kann die Initialisierung und Anbindung auch in nur einer Zeile machen:

Beispiel 16

```
ofstream myFile(const char* pFName, ios::openmode mode = ios::out);
```

Und hier sind ein paar Beispiele dazu. Eine Datei im dafualt-Modus öffnen:

Beispiel 17

```
#include <fstream>
using namespace std;
....
int main ( )
{
    ofstream myFile;
    myFile.open ("d:\\tmp\\test.dat");
    ...
}
```

Eine Datei im Binärmodus öffnen:

Beispiel 18

```
#include <fstream>
using namespace std;
....
int main ( )
{
    ofstream myFile ("d:\\tmp\\test.dat",
                     ios::out|ios::binary);
    ...
}
```

Eine Datei öffnen und auf Fehler abfragen:

Beispiel 19

```
#include <fstream>
using namespace std;
....
int main ( )
{
    ofstream myFile;
    myFile.open ("d:\\tmp\\test.dat");
    if (!myFile)
    {
        ...  // Fehlerbehandlung
    }
}
```

Achtung! Beachten Sie den Doppel-Backslash im Namen der Datei!

5.2.3.2. Dateiverbindung lösen und Stream-Objekt löschen

Hier haben wir zwei Möglichkeiten. Falls wir den *Stream*-Objekt weiterbenutzen wollen, aber mit einer anderen datei (oder noch weiteren) müssen wir die Datei schliessen:

Beispiel 20

```
#include <fstream>
using namespace std;
....
int main ( )
{
    ofstream myFile;
    myFile.open ("d:\\tmp\\test.dat");
    ...
    myFile.close();
    ....
    // Stream erneut mit einer Datei verbinden
    myFile.open(...);
}
```

Wie im Beispiel zu sehen ist, kann man dann das *stream*-Objekt sofort für eine andere Datei benutzen.

Die andere Möglichkeit ist das *stream*-Objekt selbst zu zerstören (damit auch die Verbindung mit der Datei), z.B. mit einer begrnezten Lebensdauer in einem Block:

Beispiel 21

```
#include <fstream>
using namespace std;
....
int main ( )
{
    ....
    { // Blockbeginn
        ofstream myFile("d:\\tmp\\test.dat");
        ....
    } // Blockende, Stream-Objekt löschen!
}
```

5.2.3.3. Schreiben in eine Textdatei

Das Schreiben in einer Textdatei erfolgt auf genau dieselbe Art, wie mit cout. Hier ein kleiner Beispiel:

Beispiel 22

```
#include <fstream>
using namespace std;
int var1 = 10, var2 = 20;
int main ( )
{
    ofstream myFile;
    myFile.open ("d:\\tmp\\test.dat");
    // Hier Fehler abfangen!
    // Daten schreiben
    myFile << var1 << ' ';
    myFile << setw(5) << var2 << endl;
    // Datei schliessen
    myFile.close();
}
```

5.2.3.4. Eingabedateien öffnen

Man öffnet die Eingabedateien fast auf dieselbe Art, wie auch die Ausgabedateien:

Beispiel 23

```
ifstream myFile;
void istream::open(const char *pFName, ios::openmode mode=ios::in);
```

Die Moden sind aber nur zwei:

Tabelle 4

Modus	Bedeutung
ios::in	Öffnen einer Datei zum Lesen.
ios::binary	Öffnet einer binären Datei.

Natürlich kann man beide Zeilen in eine zusammenfassen:

Beispiel 24

```
ifstream myFile(const char* pFName,  ios::openmode mode = ios::in);
```

Beispiel 25

```
#include <fstream>
using namespace std;
....
int main ( )
{
    ifstream myFile ("d:\\tmp\\test.dat",
                     ios::in|ios::binary);
    if (!myFile)
        .... // Fehlerbehandlung
    ...
}
```

5.2.3.5. Lesen aus einer Datei

… erfolgt genauso wie beim cin.

Beispiel 26

```
#include <fstream>
using namespace std;
int var1, var2;
int main ( )
{
    ifstream myFile;
    myFile.open ("d:\\tmp\\test.dat");
    // Hier Fehler abfangen!
    // Daten lesen
    myFile >> var1 >> var2;
    // Datei schliessen
    myFile.close();
}
```

5.2.3.6. Öffnen einer Datei zum gleichzeitigen Lesen und Schreiben:

Beispiel 27

```
fstream( const char* pFName, int mode );
```

Die Moden sind hier:

Tabelle 5

Mode	Bedeutung
ios::in	Öffnen zum Lesen
ios::out	Öffnen zum Schreiben
ios::binary	Öffnen einer binären Datei
ios::app	Neue Daten werden immer an die Datei angehängt
ios::ate	Neue Daten werden an die Datei angehängt.
ios::trunc	Bisheriger Dateiinhalt wird verworfen.

Achtung! Wollen Sie eine Datei zum Lesen und Schreiben öffnen, die noch nicht existiert, so müssen Sie den Mode *ios::trunc* mit angeben. Selbstverständlich können Sie dann zu Beginn nur Daten schreiben.

5.3.Aufgaben

5.3.1. Compilieren Sie und führen Sie alle Beispiel-Programme aus dem kapitel aus. Sie befinden sich auf der Homepage des Kurses.

5.3.2. Schreiben Sie ein Programm, dass verschiedene float-Variablen mit unterschiedlicher Genauigkeit ausgibt. Dabei sind die Zahl und die Genauigkeit als Paramtere einzugeben.

5.3.3. Schreiben Sie ein Programm, dass aus der Standard-Eingabe Zahlen einliest und sie in eine ASCII Datei speichert.

5.3.4. Schreiben Sie ein Programm, dass die Zahlen aus der vorherigen Aufgabe einliest und miteinander addiert.

5.3.5. Schreiben Sie ein Programm, dass einen Text zeilenweise aus der Standard-Eingabe einliest und in eine ASCII-Datei speichert.