Programmieren in C++

Stream I/O

Inhalt

- Standard Input and Output
- Datenströme (Streams)
- Formatierte Ein- und Ausgabe
- Fehlerbehandlung beim Einlesen
- Zustände von Streams
- Stream-Manipulatoren
- Unformatierte Ein- und Ausgabe
- Textdatei öffnen, schreiben und lesen
- Binärdatei öffnen, schreiben und lesen

Standardeingabe und -ausgabe in C

- Standardeingabe
  - int scanf(const char * format, ...);
  - liest Daten gemäss dem angegebenen Format von der Standardeingabe und speichert die eingelesenen Werte in die zusätzlichen Parameter
- Standardausgabe
  - int printf(const char * format, ...);
  - schreibt den C-String format auf die Standardausgabe und ersetzt dabei allfällige Formatspezifikatoren durch die zusätzlichen Parameter
- Beispiel

```
constexpr size_t len = 20;

struct S { char m_name[len]; uint32_t m_flags; double m_value; } s;
scanf_s("%s %u %lf", s.m_name, len, &s.m_flags, &s.m_value);
printf("s: name = %s, flags = %u, value = %lf\n",
   s.m_name, s.m_flags, s.m_value);
```

Standardeingabe und -ausgabe in C++

- Standardeingabe
  - Lesen eines Bytestroms von der Tastatur
  - Verwendung eines Objekts der Klasse istream (z.B. cin)
- Standardausgabe
  - Schreiben eines Bytestroms auf den Bildschirm
  - Verwendung eines Objekts der Klasse ostream
    - cout: Standardausgabe
    - cerr: Standardfehlerausgabe
    - clog: gepufferte Standardfehlerausgabe
- Beispiel

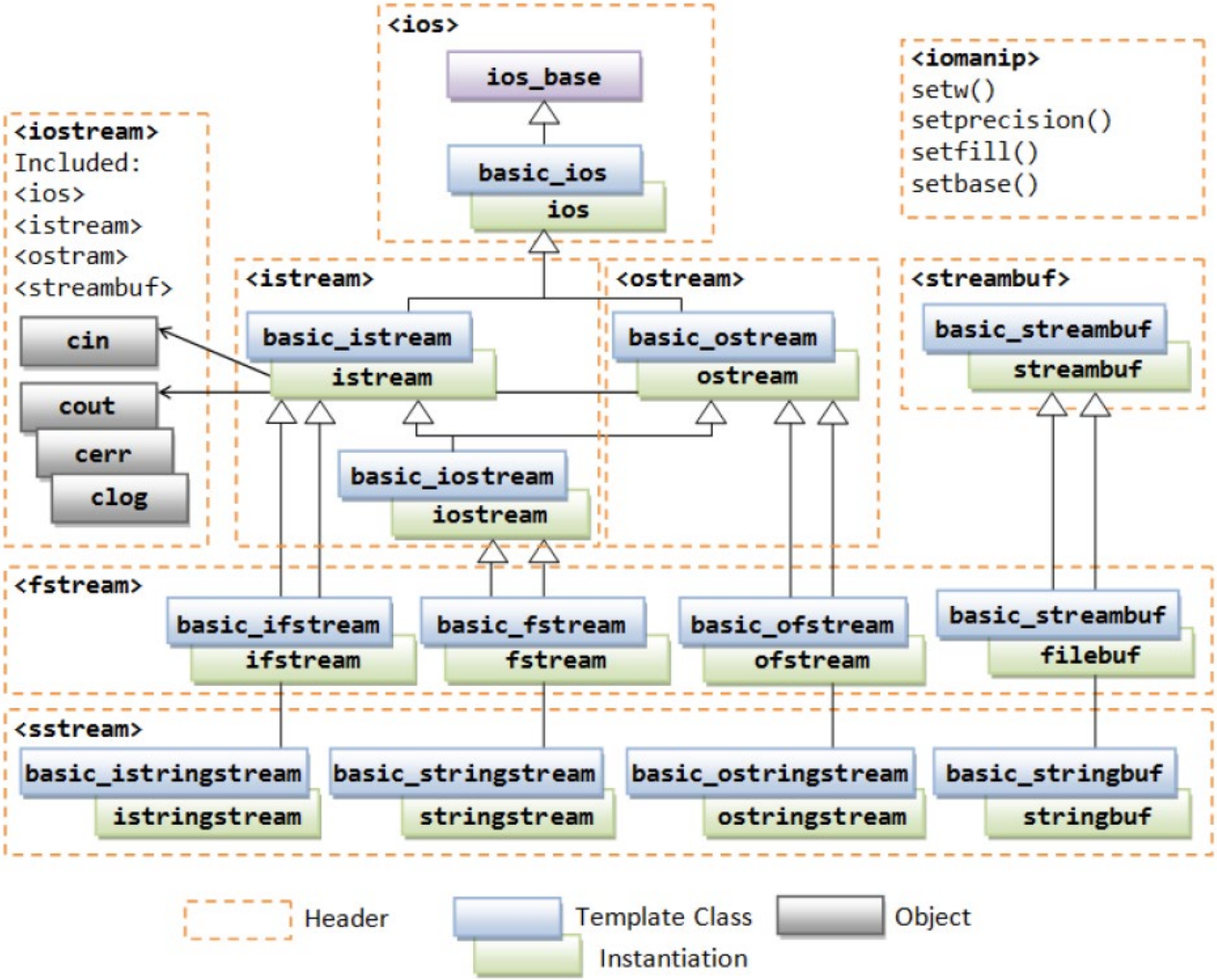
```
struct S { string m_name; uint32_t m_flags; double m_value; } s;
cin >> s.m_name >> s.m_flags >> s.m_value;
cout << "s: name = " << s.m_name << ", flags = " << s.m_flags
     << ", value = " << s.m_value << endl;
```

Datenströme (Streams)

- Was ist ein Datenstrom?
  - geordnete Folge von Datenbytes mit unbekannter Länge (Anzahl von Bytes)
- Eingabestrom (input stream)
  - Datenstrom, der aus einer Datenquelle kommt
  - Beispiel: Zeichen, die über die Tastatur eingegeben werden
- Ausgabestrom (output stream)
  - Datenstrom, der zur einer Datensenke gesendet wird
  - Beispiel: Zeichen, die auf den Bildschirm geschrieben werden

- Wo finde ich Infos dazu?
  - Streams sind Teil der Standard-Bibliothek
  - C++ Standard library

Klassenhierarchie <iostream>



Ein- und Ausgabe

- Formatierte Ein- und Ausgabe
  - Ausgabe: bei der formatierten Ausgabe wird ein Wert/Objekt als Zeichenkette in einen Ausgabestrom geschrieben
    - es wird der operator<<(…) verwendet
  - Eingabe: bei der formatierten Eingabe wird eine Zeichenkette aus einem Eingabestrom gelesen, die Zeichenkette geparst und ein Wert/Objekt des gewünschten Datentyps mit Daten abgefüllt
    - es wird der operator>>(…) verwendet
    - falls der Parser einen Fehler feststellt, wird der Wert/Objekt nicht abgefüllt und der Eingabestrom wird in einen Fehlerzustand (failbit) gesetzt
- Unformatierte Ein- und Ausgabe
  - Ausgabe: Daten werden mit write(…) als Zeichenfolge in den Ausgabedatenstrom geschrieben
  - Eingabe: Daten werden mit read(…) als Zeichenfolge aus dem Eingabestrom gelesen

Formatierte Ausgabe: ASCII-Tabelle

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

int main() {
    constexpr int nColumns = 4;
    cout << "ASCII-Tabelle" << endl << endl;

    for (int i=32; i < 128; i++) {
        cout.width(3);                  // Zahlenbreite: 3
        cout.fill('0');                 // mit führenden Nullen auffüllen
        cout << i << " = 0x";

        cout.setf(ios::hex, ios::basefield); // Zahlenbasis auf 16 setzen
        cout.setf(ios::uppercase);    // Hexzahlen mit Grossbuchstaben
        cout << i << ": ";
        cout.unsetf(ios::hex);        // wieder auf Dezimal umstellen

        cout << (char)i << '\t';      // Zeichenausgabe und zur nächsten
                                      // Tabulatorposition springen
        if (i%nColumns == nColumns - 1)
            cout << endl;             // Zeilenumbruch nach 4 Spalten
    }
}
```

Formatierte Ein- und Ausgabe (1)

```
class Person {
    std::string m_name;
    std::string m_givenName;
    int m_age;
    bool m_female;

public:
    Person(…) {}

    friend std::ostream& operator<<(std::ostream& os, const Person& p) {
        return os << p.m_givenName << " " << p.m_name << std::boolalpha
            << (p.m_female ? " (female)" : " (male)") << " is "
            << p.m_age << " years old";
    }

    friend std::istream& operator>>(std::istream& is, Person& p) {
        return is >> p.m_givenName >> p.m_name >> p.m_age
            >> std::boolalpha >> p.m_female;
    }
};
```

Formatierte Ein- und Ausgabe (2)

```
#include "Person.h"
#include <string_view>
#include <sstream>

using namespace std;

int main() {
   constexpr std::string_view s1 = "Anja Keller 21 true";

   stringstream ss(s1.data());
   Person p1, p2;

   ss >> p1;
   cout << p1 << endl;

   cin >> p2;
   cout << p2 << endl;
}
```

Fehlerbehandlung beim Einlesen

- Was passiert in folgender Situation?

```
cout << "Bitte ganze Zahl eingeben: ";
cin >> i; // die Benutzerin tippt "hallo" ein und schliesst mit Return ab
cout << "Die eingegebene Zahl ist: " << i << endl;
```

- Mit Fehlerbehandlung

```
…
if (cin.good()) {      // Kurzform if (cin) {
   cout << "Die eingegebene Zahl ist: " << i << endl;
} else {
   cin.clear();
   getline(cin, s);    // unformatierte Eingabe (ganze Zeile einlesen)
   stringstream ss(s);
   cout << "Sie haben keine ganze Zahl eingegeben, sondern die Strings: ";
   while (ss.good()) {
      ss >> s;
      cout << '[' << s << "] ";
   }
   cout << endl << "Alle eingegebenen Zeichen wurden verarbeitet." << endl;
}
```

Zustände von Datenströmen

- Zustand eines Datenstromes ist eine Zahl, `iostate`, welche mit `rdstate()` ausgelesen werden kann:
  - 0 bedeutet, dass alles in Ordnung ist.
  - alle anderen Zahlen bedeuten, dass sich der Strom in einem Fehlerzustand befindet, wobei eines oder mehrere Fehlerbits gesetzt sind
- Abfragen einzelner Bits von `iostate` mit
  - `good()`
  - `eof()`
  - `fail()`
  - `bad()`
- Manuelles Setzen des fail Bits von `iostate` mit
  - `setstate(std::ios::failbit)`

Bits in iostate

Bsp.: Öffnen einer Datei schlägt fehl oder es soll ein int eingelesen werden und der Benutzer gibt «hallo» ein oder EOF wurde erreicht (dann werden eof und fail bit gesetzt)

Bsp.: Hardware-Fehler oder Fehler im Betriebssystem oder der Stream Library
→ ziemlich mühsam, Programmierer kann nicht viel tun!

iostate	bedeutet	Funktionen, die iostate prüfen			
		good()	eof()	fail()	bad()
goodbit	Keine Fehler (iostate = 0)	true	false	false	false
eofbit	Ende der Datei erreicht (bei Input)	false	true	false	false
failbit	logischer Fehler bei i/o Operation	false	false	true	false
badbit	Lese/Schreib-Fehler auf Stream-Buffer	false	false	true	true

Stream-Manipulatoren

- Idee
  - anstatt dem mühsamen Setzen von Flags (z.B. mit setf(..))
    werden Stream-Manipulatoren gezielt in den Datenfluss integriert
- einfaches Beispiel
  - vorher

```
cout.setf(ios::hex, ios::basefield);      // Zahlenbasis auf 16 setzen
cout.setf(ios::uppercase);                // Hexzahlen mit Grossbuchstaben
cout << i << endl;
```

  - nachher

```
cout << hex << uppercase << i << endl;
```

- weiteres Beispiel
  - vorher

```
cout.width(3);                            // Zahlenbreite: 3
cout.fill('0');                           // mit füllenden Nullen auffüllen
cout << i << endl;
```

  - nachher

```
cout << setw(3) << setfill('0') << i << endl;
```

- was steckt dahinter?

Manipulatoren ohne Parameter

- Beispiel
  - cout << hex << uppercase << i << endl;
  - hex, uppercase und endl sind Zeiger auf Funktionen (Funktionszeiger)

- Ausschnitt aus der Klasse ostream

```
ostream& operator<<(ostream& (*fp) (ostream&) ) {
    return fp(*this);       // Funktionsaufruf
}
ostream& endl(ostream& os){
    os.put('\n');
    os.flush();
    return os;
}
```

Manipulatoren mit Parameter

```
cout << setw(3) << setfill('0') << i << endl;
-----
template<class T> class Omanip {
   using Func = std::function<std::ostream& (std::ostream&, T)>;
   Func m_fp;
   T m_arg;
public:
   Omanip(Func f, const T& arg)
   : m_fp(f)
   , m_arg(arg) {}

   friend std::ostream & operator<<(std::ostream& os, Omanip<T>& o) {
      return o.m_fp(os, o.m_arg);
   }
};

std::ostream& width(std::ostream& os, int arg) { … }

Omanip<int> setw(int w) {
   return Omanip<int>(&width, w);
}
```

Unformatierte Ein- und Ausgabe

- Unformatierte Eingabe
  - peek gibt Vorschau auf das nächste Zeichen im Zeichenstrom
  - get liest ein Zeichen vom Zeichenstrom
  - read liest *n* Zeichen vom Zeichenstrom
  - getline liest eine ganze Zeile oder bis zu einem angegebenen Trennzeichen

    (beim Übergang von formatierter zu unformatierter Eingabe können mit is >> ws nicht konsumierte Whitespaces vorgänging konsumiert werden)
  - ignore überliest und ignoriert Zeichen im Zeichenstrom
  - gcount gibt Anzahl verarbeitete Zeichen der letzten unformatierten Eingabe zurück
  - unget macht das zuletzt gelesene Zeichen im Zeichenstrom nochmals verfügbar
- Unformatierte Ausgabe
  - put schreibt ein Zeichen in den Zeichenstrom
  - write schreibt *n* Zeichen in den Zeichenstrom

Textdatei öffnen und unformatiert lesen

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

void main(){
    ifstream inData;
    string fileName;

    cout << "Geben Sie den Dateinamen ein: ";
    cin >> fileName;

    inData.open(fileName, ios::in);                    // Datei öffnen
    if (!inData) {
        cerr << "Datei konnte nicht geoeffnet werden!" << endl;
    } else {
        while (! inData.eof()) {
            cout << static_cast<char>(inData.get());   // Datei zeichenweise
        }                                              // lesen und ausgeben
        inData.close();                                // Datei schliessen
    }
}
```

Textdatei öffnen, lesen und schreiben

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

int main() {
	// Datei anlegen
	fstream inOutFile("fstream.txt",
		ios::out | ios::trunc);
	inOutFile.close(); // leere Datei existiert nun

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

	// Datei formatiert schreiben
	for(int j = 1; j <= 20; ++j)
		inOutFile << j << ' ';
	inOutFile << endl;

	// zum Anfang der Datei springen
	inOutFile.seekg(0);

	// Sauberes Lesen mit read-ahead Logik
	int i;
	while(inOutFile >> i) {
		cout << i << ' '; // Kontrollausgabe
	}

	// Datei ab Pos. 25 lesen
	inOutFile.clear();   // EOF-Status löschen
	inOutFile.seekg(25); // Leseposition 25 suchen

	// zum Anfang der nächsten Zahl gehen
	while (inOutFile.get() != ' ');

	// restliche Zahlen lesen und ausgeben
	while (inOutFile >> i) {
		cout << i << ' '; // Kontrollausgabe
	}
}
```

Binärdatei zuerst schreiben, …

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

using namespace std;

struct S {
	int value;
	char text[20];
};

int main() {
	constexpr int Size = 10;

	S array[Size]{};
	ofstream ofs;
```

```
	// Array initialisieren
	for(int i = 0; i < Size; i++) {
		stringstream strs;
		array[i ].value = i;
		strs << setw(2) << setfill('0') << (i+1) << "._Array-Element“;
		strs >> array[i ].text;
	}

	// Ausgabedatei öffnen
	ofs.open("Ausgabe.dat", ios::out | ios::binary);
	if (!ofs) {
		cerr << "Datei konnte nicht geoeffnet werden!\n";
		return 1;
	}

	// Array in Datei schreiben
	ofs.write((char *)array, size*sizeof(S));

	// Datei schliessen
	ofs.close();
```

… dann lesen (Fortsetzung)

```
    ifstream ifs;
    S array2[Size]{};

    // Eingabedatei öffnen
    ifs.open("Ausgabe.dat", ios::in | ios::binary);
    if (! ifs) {
        cerr << "Datei konnte nicht geoeffnet werden!\n";
        return 1;
    }

    // Array einlesen und ausgeben
    ifs.read((char*)&array2, Size*sizeof S);

    for (S& s: array2) {
        cout << s.text << " = " << s.value << endl;
    }

    // Datei schliessen (optional)
    ifs.close();
}
```