Iostream - Ein/Ausgabe

(vgl. RRZN Handbuch C++ für C-Programmierer S.47 ff.)

- C++ - Standardbibliothek stellt die notwendigen Mechanismen für die Ein- und Ausgabe von **Grunddatentypen** zur Verfügung und erlaubt die Konstruktion eigener Ein- und Ausgabe-funktionen für selbst definierte Klassen.

- Auf unterster Ebene wird ein **stream** als Strom oder Folge von **Bytes** aufgefasst.

- Das **Byte** ist die **Dateneinheit** des Stroms.

- Andere Datentypen, wie **int**, **char ***, **string**, **double**, **float** oder **vector** erhalten erst durch die Bündelung und Interpretation der Bytesequenzen auf höherer Ebene ihre Bedeutung.

- Die Basisklasse heißt **ios_base**, aus ihr werden die anderen Klassen abgeleitet.

- Die mit **basic** beginnenden Klassen sind **Templates**, die für beliebige Zeichentypen geeignet sind, z.B. Unicode-Zeichen, die den Typ **wchar_t** haben.

- Für den am häufigsten benötigten Datentyp **char** sind die Klassen durch Typdefinitionen, wie
  typedef basic_ofstream <char> ofstream;
  spezialisiert worden.

Iostream - Ein/Ausgabe

(vgl. RRZN Handbuch C++ für C-Programmierer S.47 ff.)

- Zusätzlich zur E/A von C ueber **#include <*stdio.h*>** (*printf*, *scanf*, ...) existiert in C++ die Stream-E/A über ***#include <iostream>*** und ***#include <iomanip>***

- **Stream** spiegelt wider, dass Zeichenfolgen bei der E/A als "Datenstrom" behandelt werden.

- Die Stream-E/A in C++ ist **typsicherer** und **weniger fehleranfällig** als die E/A in C

- **Iostream**-Klassen:

Name der Klasse	Verwendung	Header
ostream	Ausgabe (Basisklasse)	<iostream>
istream	Eingabe (Basisklasse)	<iostream>
iostream	Ein-/Ausgabe (Basisklasse)	<iostream>
ofstream	Datei-Ausgabe	<fstream>
ifstream	Datei-Eingabe	<fstream>
fstream	Datei-Ein-/Ausgabe	<fstream>
ostringstream	Ausgabe in string - Objekt	<sstream>
istringstream	Eingabe aus string - Objekt	<sstream>
stringstream	string - Ein-/Ausgabe	<sstream>

Iostream - Ein/Ausgabe

(vgl. RRZN Handbuch C++ für C-Programmierer S.47 ff.)

- Standardströme:

Typ	Name	Bedeutung
istream	**cin**	Standardeingabe
ostream	**cout**	Standardausgabe
ostream	**cerr**	Standard-Fehlerausgabe
ostream	**clog**	Log-Ausgabe (ungepuffert)

- Alle in der C++ - Standardbibliothek definierten globalen Bezeichner gehören zum Namensbereich ***std***, Zugriff über ***std::cout*** oder ***using namespace std;*** und nur ***cout***

- Regeln beim Einlesen über ***cin >>***
  - für die einfachen Datentypen (double, int, float, char, ...) ist ***cin*** >> in C++ vordefiniert.

  - führende Zwischenraumzeichen (Whitespace-Zeichen, z.B. Leerzeichen, Enter, Tabulatoren) werden überlesen

  - **Abbruch** der Eingabe, wenn ein Zeichen des Stroms **nicht** zum Typ der eingelesenen Variable paßt. Im Fehlerfall wird ein **Fehler-Bit** (**ios::failbit**) gesetzt und die **interne Streamposition** verbleibt beim fehlerhaften Zeichen. Eine **Folgeeingabe** funktioniert in dem Zustand nicht !

Iostream - Ein/Ausgabe

(vgl. RRZN Handbuch C++ für C-Programmierer S.47 ff.)

- E/A-Fehler werden in der Klasse **ios_base** bzw. **ios** in einem **Statuswort** vom Aufzählungstyp **iostate** gespeichert. Die tatsächlichen **Bitwerte** sind **implementationsabhängig**.

```
enum iostate { goodbit = 0x00,  // Eingabe ohne Fehler
               eofbit = 0x01,   // eof
               failbit = 0x02,  // letzte E/A mit Fehlern
               badbit = 0x04    // unfertige Operation, grober Fehler
};
```

- Im Fehlerfall werden nachfolgende E/A-Operationen solange blockiert, bis **ios::goodbit** gesetzt wird

- Aufrufe Bits des Statuswortes: **ios::eofbit** // Datei-Ende (EOF)
  ios::failbit // Fehler bei Ausführung
  ios::badbit // Fehler mit Datenverlust

  Mehrere dieser Bits können gesetzt sein !

- Zugehörige logische Abfragefunktionen für Objekte **cin**, **cout**, **cerr**, **clog** der Klasse **ios**:

bool fail(); bool eof(); bool bad(); bool good()

Iostream - Ein/Ausgabe

(vgl. RRZN Handbuch C++ für C-Programmierer S.47 ff.)

- Lesen bis Eingabeende **eof**: **int wert = 0; while (cin >> wert) {** /* verarbeite wert */ **}**

- **Statuswort** für Objekte cin, cout, cerr, clog insgesamt: **int rdstate()**

- Setzen des **Statuswortes**: **void clear(int status = ios::goodbit)**

- Aufruf: **cin.clear()**; ist äquivalent zu **cin.clear(ios::goodbit)**;

- Das **erste** Whitespace-Zeichen des Eingabestroms **beendet** die Eingabe, z.B **Enter-Taste**, **Leerzeichen**, **Tabulatoren**.

Achtung: Auch Zeichenketten (char *) werden beim ersten Whitespace-Zeichen beendet, z.B. **char s[128]; cin>>s;** mit ***HTW Dresden*** im Strom liest nur ***"HTW"*** nach **s**

- Zeichenketten (**char ***) werden bei **cin>>s;** automatisch immer mit **'\0'** abgeschlossen.

- Rezept: Nach jeder **cin** >> - Eingabe sollte **cin.clear(); cin.ignore(INT_MAX, '\n');** gerufen werden, um ein evtl. gesetztes Fehler-Bit zurückzusetzen und die aktuelle Position des Stromes auf die **Enter**-Taste (**'\n'**) zu setzen. Nur dadurch funktionieren weitere **cin** >> - Eingaben fehlerfrei.

Iostream - Ein/Ausgabe

(vgl. RRZN Handbuch C++ für C-Programmierer S.47 ff.)

- Mittels **cin.peek()** kann das nächste Zeichen des Eingabestroms erhalten werden, ohne es aus dem Eingabestrom zu übernehmen (vgl. S. 55). Falls **cin.peek() == '\n'** gilt, dann war die letzte Eingabe mittels **cin** >> fehlerfrei, vgl. Beispiel **robust_ea1.cpp**

- Um einen Pufferüberlauf zu vermeiden, sollten Zeichenketten bei der Eingabe mit **setw** und **sizeof** limitiert werden, z.B. werden hier maximal 79 Zeichen gelesen, Abschluß mit '\0':

```
char s[80]; cout << "Eingabe Zeichenkette: "; cin >> setw(sizeof  s) >> s;
```

- In der Klasse **ostream** sind überladene Operatoren für eingebaute Datentypen enthalten.

- Der Operator << dient dazu, ein Objekt eines internen Datentyps in eine Folge von ASCII - Zeichen zu verwandeln:

```
ostream &operator<<(const char *);  // C-Strings
ostream &operator<<(const char);
ostream &operator<<(const int); ostream &operator<<(const float); u.s.w.
```

- Wegen Rückgabetyp "**Referenz auf ostream**" ist Hintereinanderschaltung von Ausgabe-operatoren möglich: **cerr<<"x = "<<x;** wird als **(cerr.operator<<("x = ")).operator<<(x);** interpretiert. Weitere Ausgabemöglichkeiten: **ostream &put(char);** // gibt Zeichen aus
ostream &write(const char *, size_t); // binäre Ausgabe