4 Ein- und Ausgabe

4.1 Einfache Ein- und Ausgabe

4.2 Manipulatoren skipws und noskipws

4.3 Allgemeine Dateien

4.4 Protokollierung

4.5 Header-Dateien und Klassenüberblick

4.6 Formatierung

4.7 Manipulatoren

4.8 Stringstream

4.9 Datenpuffer

4.10 Mit Dateien arbeiten

4.11 Internationalisierung

```
//
// Einfache Ein- und Ausgabe über Konsole und Bildschirm
//

#include <iostream>
#include <string>
using  std::cin;
using  std::cout;
using  std::endl;
using  std::flush;
//  using  std::getline;  //unnötig wegen Koenig lookup
using  std::hex;
using  std::showpoint;
using  std::string;

int main () {

        cout << "Bitte Ganzzahl Gleitpunktzahl (2x) "
                "Zeichen Text: "  << endl;

        int i;
        cin >> i;

        double d, dd;
        cin >> d >> dd;

        char c;
        cin >> c;

        string text;
        // möglicherweise fehlerträchtig!!
        cin >> text;
```

```
        // eventuell besser
        string text2;
        getline (cin, text2, '\n');

        cout << "Eingelesen wurden:" << endl
             << "i = " << i << endl
             << "d, dd = " << d << ",  " << dd << endl
             << showpoint
             << "nach Manipulator showpoint" << endl
             << "d, dd = " << d << ",  " << dd << endl
             << "c = " << c << endl
             << "text = " << text << endl
             << "text2 = " << text2 << endl << endl;

        cout << flush;                              //  Puffer leeren
        cout << "i (in Sedezimalform) = " << hex << "0x" << i
             << endl;
}//main


/*  Ein Beispiellauf:
Bitte Ganzzahl Gleitpunktzahl (2x) Zeichen Text:
34 45.125678 48 B Dies ist eine mehrteilige Zeichenkette.
Eingelesen wurden:
i = 34
d, dd = 45.1257,  48
nach Manipulator showpoint
d, dd = 45.1257,  48.0000
c = B
text = Dies
text2 =  ist eine mehrteilige Zeichenkette.

i (in Sedezimalform) = 0x22
*/
```

```
//
// Manipulatoren skipws und noskipws
//
#include <iostream>
using  std::cin;
using  std::cout;
using  std::endl;
using  std::skipws;
using  std::noskipws;

int main () {
      cout << "Leerzeichen werden ueberlesen." << endl;
      cout << "Bitte Folge von Zeichen mit Leerzeichen, "
            << "beendet mit Z: " << endl;

      char c;
      while (cin >> c)
            if  (c == 'Z')
                  break;
            else
                  cout << c;
      cout << endl;

      cin >> noskipws;
      while (c != '\n')  cin >> c;
      cout << "Nach Manipulator noskipws." << endl;
      cout << "Bitte Folge von Zeichen mit Leerzeichen: "
            << endl;

   while (cin >> c)
      if (c == '\n')
            break;
      else
            cout << c;
```

```
        cout << endl;

        cin >> skipws;
        cout << "Nach Manipulator skipws." << endl;
        cout << "Bitte Folge von Zeichen mit Leerzeichen, "
                "beendet mit Z: " << endl;

        while (cin >> c)
                if (c == 'Z')
                        break;
                else
                        cout << c;
        cout << endl;
}//main



/*  Ein Beispiellauf:

Leerzeichen werden ueberlesen.
Bitte Folge von Zeichen mit Leerzeichen, beendet mit Z:
a    s  d     f       gZ
asdfg
Nach Manipulator noskipws.
Bitte Folge von Zeichen mit Leerzeichen:
a    s  d     f       g
a    s  d     f       g
Nach Manipulator skipws.
Bitte Folge von Zeichen mit Leerzeichen,  beendet mit Z:
a  s    d     f  g    Z
asdfg

*/
```

```
//  Benutzung allgemeiner Dateien
//
//    istream& get      (char* s, streamsize sz, char delim = '\n')
//    istream& getline (char* s, streamsize sz, char delim = '\n')
//    Es werden höchstens sz-1 Zeichen nach (*s) übertragen,
//    '\0' wird hinzugefügt; getline liest auch das Trennzeichen,
//    get nicht.

#include <iostream>
#include <fstream>
using std::ifstream;
using std::ofstream;
using std::cout;
using std::cerr;
using std::endl;


int main () {
   const int sz = 999;      // hinreichend groß
   char buf [sz];
   {                        //  Öffnen eines neuen Block
      ifstream in ("testleer.txt");
      if  (!in) {
            cerr << "Eingabedatei konnte nicht geöffnet"
                     " werden!"  << endl;
            exit (99);
      }
      ofstream out ("Strauss.txt");
      if  (!out) {
            cerr << "Ausgabedatei konnte nicht geöffnet"
                     " werden!"  << endl;
            exit (98);
      }
```

```
      //  Benutzung von get:
      while (true) {
         in.get  (buf,  sz);
         cout << "Gelesene Zeichen = " << in.gcount () << endl;
         if  (in.eof ())
            break;
         else if  (in.bad ()) {
            cerr << "Fehler bei Eingabe!" << endl;
            exit (98);
         } else if  (in.fail ()) {
            cerr << "Behebbarer Fehler?" << endl;
            in.clear ();
            in.get ();                    // Überlesen von (\n)
         }
         cout << buf << endl;
         out << buf << endl;
      }
   } // Destruktion von in und out

   //Neuöffnen von testleer.txt
   ifstream in  ("testleer.txt");
   //  Benutzung von getline
      cout << endl << endl << "Zweiter Versuch!!" << endl
           << endl;
   while (in.getline (buf,  sz)) {
      //  getline entfernt Zeichen 'delim'.
      cout << "Gelesene Zeichen = " << in.gcount () << endl;
      char* cp = buf;
      cout << cp << endl;
   }
}//main
```

Inhalt der Datei testleer.txt:

Eine Datei mit Leerzeilen.
-- <leer> --
Nun 2 Leerzeilen.
-- <leer > --
-- <leer> --
Letzte Zeile und 1 Leerzeile.
-- <leer> --

/* *Ausgabe

Gelesene Zeichen = 26
Eine Datei mit Leerzeilen.
Gelesene Zeichen = 0
Behebbarer Fehler?

Gelesene Zeichen = 0
Behebbarer Fehler?

Gelesene Zeichen = 17
Nun 2 Leerzeilen.
Gelesene Zeichen = 0
Behebbarer Fehler?

Gelesene Zeichen = 0
Behebbarer Fehler?

Gelesene Zeichen = 0
Behebbarer Fehler?

Gelesene Zeichen = 29
Letzte Zeile und 1 Leerzeile.
Gelesene Zeichen = 0
Behebbarer Fehler?

Gelesene Zeichen = 0

Zweiter Versuch!!

Gelesene Zeichen = 27
Eine Datei mit Leerzeilen.
Gelesene Zeichen = 1

Gelesene Zeichen = 18
Nun 2 Leerzeilen.
Gelesene Zeichen = 1

Gelesene Zeichen = 1

Gelesene Zeichen = 30
Letzte Zeile und 1 Leerzeile.
***/**

Inhalt der Datei Strauss.txt:

Eine Datei mit Leerzeilen.
-- <leer> --
-- <leer> --
Nun 2 Leerzeilen.
-- <leer> --
-- <leer> --
-- <leer> --
Letzte Zeile und 1 Leerzeile.
-- <leer> --
-- <leer> --

```
//
//  Protokollierung
//

#include <iostream>
#include <iomanip>
#include <fstream>
using  std::setw;
using  std::hex;
using  std::dec;
using  std::oct;



class protocol {
public:
	// Hier ohne Fehlerkontrolle
	protocol (char*  fn) {
		ofs.open (fn);
	}
	~protocol () {
		ofs.close ();
	}
	template <class T>
	protocol& operator << (T v) {
		std::cout << v;
		ofs << v;
	return *this;
	}
private:
	std::ofstream  ofs;
};//protocol
```

```
int main () {
        protocol pro ("protocol.txt");
        pro << "sedezimal" << "  dezimal" << "  oktal" << '\n';
        for (int i = 0;   i < 16;   ++i) {
                pro << hex << setw (5) << i
                        << dec << setw (10) << i
                        << oct << setw (8) << i << '\n';
        }
}//main
```

/ *Inhalt von protocol.txt:*

sedezimal	dezimal	oktal
0	0	0
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5
6	6	6
7	7	7
8	8	10
9	9	11
a	10	12
b	11	13
c	12	14
d	13	15
e	14	16
f	15	17

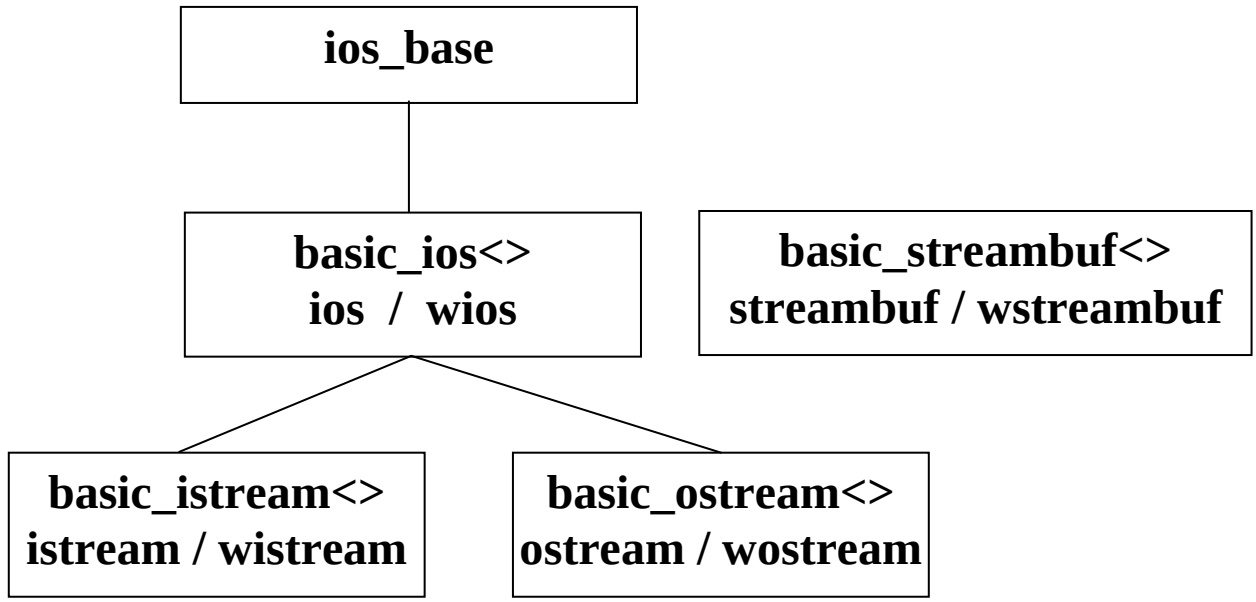
/

Header-Dateien und Klassenüberblick

<iosfwd>	**Vorwärtsdeklarationen für Template-Klassen**
<iostream>	**Grundobjekte**
<ios>	**Basisklassen**
<streambuf>	**Pufferklassen für Ein- und Ausgabe**
<istream>	**"input stream"**
<ostream>	**"output stream"**
<iomanip>	**Manipulatoren für Stream-Klassen**
<sstream>	**"string stream"**
<fstream>	**Datei-Ein- und Ausgabe**
<cstdio>	**Erbe von C**
<cinttypes>	**Erbe von C**

Bemerkung: **Die Header-Datei cstdio umfaßt die Makros, Typen und Funktionen der C-Header-Datei stdio.h.**

Klassenhierarchie der templatisierten Stream-Klassen, jeweils für die Zeichen-Typen char und wchar_t.



Vordefinierte Objekte:

```
istream     cin,
ostream     cout,
ostream     cerr,
ostream     clog,
wistream    wcin,
wostream    wcout,
wostream    wcerr,
wostream    wclog.
```

Angaben über Zustand eines Stroms:

Nutzung von Zustandsbit:

```
        goodbit
        eofbit      /*  Dateiende erkannt  */
        failbit     /*  Leichter Fehler  */
        badbit      /*  Schwerwiegender Fehler  */
```

Nutzung von Funktionen:

```
        good ()     /*  alles O. K.  */
        eof ()      /*  Dateiende  */
        fail ()     /*  failbit  oder  badbit  */
        bad ()      /*  nur badbit  */
```

Bemerkung: Mit dem eofbit wird auch das failbit gesetzt.

Operatoren für Ströme:

```
operator  void* ()      entspricht ! fail ()
operator ! ()           entspricht  fail ()
```

```
//  Beispiel zu eof ()
#include <iostream>
#include <cstdlib>
using namespace std;


double rp (istream& strm) {
   double value, sum;

   sum = 0;
   while (strm >> value) {
      sum += value;
   }

   if (!strm.eof ()) {
        throw ios::failure ("Eingabefehler in rp ()");
   }
return sum;
}//rp



int main ()  {

   double sum;

   try {
      sum = rp (cin);
   }
   catch (const ios::failure&  error) {
      cerr << "E/A exception: " << error.what () << endl;
      return EXIT_FAILURE;
   }
```

```
   catch (const exception&  error) {
      cerr << "standard exception: " << error.what () << endl;
      return EXIT_FAILURE;
   }
   catch (...) {
      cerr << "unknow exception" << endl;
      return EXIT_FAILURE;
   }

   cout << "Summe = " << sum << endl;
}//main


/*  Zwei Läufe:

Lauf 1:
1.2
2.3
3.4
^Z
Summe = 6.9

Lauf 2:
12.3
23.4
0x23.4
E/A exception: Eingabefehler in rp ()

*/
```

```
//  Ein neues Format für Gleitpunktzahlen in C99
//  Compiler:  gcc mindestens ab Version 4.2.0

#include <stdio.h>

int main () {
   double d1 = 0xa.fP10;
   double d2 = 0xb.abcdef1P101;
   double d3 = 0x.00ffP1003;
   double d4 = 0x1.123456789abcdep1001;

   printf  ("d1 = %g (Normalformat)\nd1 = %0.14A "
            "(P-Format)\n\n", d1, d1);
   printf  ("d2 = %g (Normalformat)\nd2 = %0.14a "
            "(p-Format)\n\n", d2, d2);
   printf  ("d3 = %g (Normalformat)\nd3 = %0.14A "
            "(P-Format)\n\n", d3,d3);
   printf  ("d4 = %g (Normalformat)\nd4 = %0.14A "
            "(P-Format)\n", d4,d4);
return 0;
} // main

/*  Ausgabe:
d1 = 11200 (Normalformat)
d1 = 0X1.5E000000000000P+13 (P-Format)

d2 = 2.95898e+031 (Normalformat)
d2 = 0x1.7579bde2000000p+104 (p-Format)

d3 = 3.33538e+299 (Normalformat)
d3 = 0X1.FE000000000000P+994 (P-Format)

d4 = 2.29541e+301 (Normalformat)
d4 = 0X1.123456789ABCE0P+1001 (P-Format)
*/
```

```
//  Beispiel zu hexfloat,  Neuerung in c++0x,
//  ausgeführt mit Visual Studio 2012.
#include <iostream>
#include <iomanip>
using std::cout;
using std::hexfloat;
using std::setprecision;

int main () {
   double d1 = 1042;
   double d2 = 2.95898e+31;
   double d3 = 3.33538e+299;
   std::ios_base::fmtflags  alt = cout.flags ();
   cout << "Normalformat  d1 = " << d1 << "\n"
      << "p-Format      d1 = " << hexfloat << d1 << "\n"
      << "Precision 14  d1 = " << setprecision(14) << d1
      << "\n\n";
   cout.flags (alt);
   cout << "Normalformat  d2 = " << d2 << "\n"
      << "p-Format  14  d2 = " << hexfloat << d2 << "\n\n";
   cout.flags (alt);
   cout << "Normalformat  d3 = " << d3 << "\n"
      << "p-Format  14  d3 = " << hexfloat << d3 << "\n\n";
}
/*Ausgabe:
Normalformat  d1 = 1042
p-Format          d1 = 0x1.048000p+10
Precision 14      d1 = 0x1.04800000000000p+10

Normalformat  d2 = 2.95898e+031
p-Format  14    d2 = 0x1.7579cce3aa9430p+104

Normalformat  d3 = 3.33538e+299
p-Format  14    d3 = 0x1.fdffd352b25520p+994    */
```

```
//  hexfloat und defaultfloat
#include <iostream>
#include <iomanip>
using std::cout;                using std::endl;
using std::hexfloat;            using std::defaultfloat;

int main () {
  double a  =  3.14159265358979323846264338327950;

  cout  << "hexfloat:\n" << hexfloat;
  for (int i  =  12; i < 16;  ++i) {
     cout.precision (i);
     cout << "precision = " << i << "   " << a << endl;
  }
  cout << endl;
  cout << "defaultfloat:\n" << defaultfloat;
  for (int i  =  12; i < 16;  ++i) {
     cout.precision(i);
     cout << "precision = " << i << "   " << a << endl;
   }
}
/*  Ausgabe:
hexfloat:
precision = 12  0x1.921fb54442d1p+1
precision = 13  0x1.921fb54442d18p+1
precision = 14  0x1.921fb54442d180p+1
precision = 15  0x1.921fb54442d1800p+1

defaultfloat:
precision = 12  3.14159265359
precision = 13  3.14159265359
precision = 14  3.1415926535898
precision = 15  3.14159265358979
*/
```

```
#include <iostream>
#include <sstream>
#include <iomanip>
using std::stringstream;
using std::cout;
using std::hexfloat;
using std::setprecision;

int main () { // Nutzung von hexfloat

  stringstream str;
  double d = 1.0/3.0;
  str << setprecision (13);
  str << d;
  double res = 0.0;
  str >> res;
  cout << setprecision (13);
  cout << hexfloat << d << "   " << res
       << (d == res ? "  " : "  nicht ") << " gleich " << '\n';
  str.clear ();
  str << setprecision (13);
  str << hexfloat << d;
  res = 0.0;
  str >> res;
  cout << setprecision (13);
  cout << hexfloat << d << "   " << res
       << (d == res ? "  " : " nicht ") << "  gleich " << '\n';
}//main

/*
0x1.5555555555555p-2  0x1.55555555552fdp-2 nicht gleich
0x1.5555555555555p-2  0x1.5555555555555p-2 gleich
*/
```

```
//  Ein- und Ausgabe von Ganzzahlgrößen
#include <iostream>
using std::cout;
using std::cin;
using std::endl;
using std::hex;
using std::oct;
using std::dec;
using std::showbase;

int main () {
   int  ia,  ib,  ic,  id;
   cout << "dec, hex und oct:\n"
        "zunaechst zwei Hex-Werte,\n"
        "dann zwei Oct-Werte."<< endl;
   cin >> hex >> ia >> ib;
   cin >> oct >> ic >> id;
   // Ausgabe
   cout << "Gelesen wurden:\n";
   cout << ia << "   " << ib << endl
       << showbase << hex
       << ic << "   " << id << endl;
}
/*
dec, hex und oct:
zunaechst zwei Hex-Werte,
dann zwei Oct-Werte.
0x24   2f
024    34
Gelesen wurden:
36     47
0x14   0x1c
*/
```

```
//  Einlesen von Ganzzahlen
#include <iostream>

using std::cin;
using std::cout;
using std::endl;
using std::ios;
using std::hex;
using std::oct;

int main () {
   int id  =  -1;
   int ih  =  -1;
   int io  =  -1;

   cin.setf (0,  ios::basefield);
   cout << "Bitte Zahlen in drei Schreibweisen: " << endl;

   cin >> id >> ih >> io;

   cout << "Gelesen wurden:" << endl;
   cout << "id = " << id << endl;
   cout << "ih = " << hex << ih << endl;
   cout << "io = " << oct << io <<endl;
}

/*
Bitte Zahlen in drei Schreibweisen:
  9876  0xfedcba  023477
Gelesen wurden:
id = 9876
ih = fedcba
io = 23477
*/
```

```
//  Ein- und Ausgabe Boolscher Größen
#include <iostream>
using std::cout;
using std::cin;
using std::endl;
using std::boolalpha;
using std::noboolalpha;


int main () {
   bool ba, bb, bc, bd;
   cout << "boolalpha und noboolalpha:\n"
        "zunaechst zwei Werte in Wortdarstellung,\n"
        "dann zwei Werte in Zahldarstellung."<< endl;
   cin >> boolalpha >> ba >> bb;
   cin >> noboolalpha >> bc >> bd;
   // Ausgabe
   cout << "Gelesen wurden:\n";
   cout << ba << "   " << bb
       << endl << boolalpha << bc << "   " << bd << endl;
}


/*
boolalpha und noboolalpha:
zunaechst zwei Werte in Wortdarstellung,
dann zwei Werte in Zahldarstellung.
false true
1  3
Gelesen wurden:
0   1
true  true
*/
```

Formatierung

Formatanzeigen

boolalpha
dec
defaultfloat
fixed
hex
hexfloat
internal
left
oct
right
scientific
showbase
showpoint
showpos
skipws
unitbuf
uppercase

basefield
dec oder hex oder oct

adjustfield
left oder right oder internal

floatfield
fixed oder scientific

Funktionen

```
char_type fill (),
char_type fill (char_type)
streamsize precision (),
streamsize precision (streamsize)
streamsize width (),
streamsize width (streamsize)
```

```
//  Ausgabe von Gleitpunktzahlen
#include <iostream>

using std::cout;
using std::endl;
using std::fixed;
using std::scientific;
using std::hexfloat;
using std::defaultfloat;
using std::ios_base;

int main() {
   cout      << "Zahl 0.01 in fixed:          " << fixed
             << 0.01 << endl
             << "Zahl 0.01 in scientific:     " << scientific
             <<  0.01  << endl
             << "Zahl 0.01 in hexfloat:       " << hexfloat
             <<  0.01  << endl
             << "Zahl 0.01 in defaultfloat:  " << defaultfloat
             << 0.01 << endl << endl;
```

```
// Nutzung von  setf  und  unsetf
   cout.setf (ios_base::fixed, ios_base::floatfield);
   cout << "Zahl 29.03 in fixed:       " << 29.03 << endl;
   cout.setf (ios_base::scientific,  ios_base::floatfield);
   cout << "Zahl 29.03 in scientific:   "  << 29.03  <<  endl;
   cout.setf (ios_base::fixed
            | ios_base::scientific, std::ios_base::floatfield);
   cout << "Zahl 29.03 in hexfloat:     "  << 29.03  <<  endl;
   cout.unsetf (ios_base::floatfield);
   cout << "Zahl 29.03 in defaultfloat:  "  <<  29.03  <<  endl;
}//main



/*  Ausgabe:

Zahl 0.01 in fixed:                0.010000
Zahl 0.01 in scientific:           1.000000e-002
Zahl 0.01 in hexfloat:             0x1.47ae14p-7
Zahl 0.01 in defaultfloat:         0.01

Zahl 29.03 in fixed:               29.030000
Zahl 29.03 in scientific:          2.903000e+001
Zahl 29.03 in hexfloat:            0x1.d07ae1p+4
Zahl 29.03 in defaultfloat:        29.03

*/
```

Die Formatanzeigen werden in einer Bitleiste vom Typ ios_base::fmtflags gespeichert.

Abfrage und Setzen aller Anzeigen erfolgt über das Funktionenpaar:

```
fmtflags  flags () const
fmtflags  flags (fmtflags)
```

Setzen und Rücksetzen einzelner Anzeigen erfolgt über:

```
fmtflags  setf (fmtflags)
fmtflags  setf (fmtflags,  fmtflags)
void    unsetf (fmtflags)
```

```
#include <iostream>

using std::cout;
using std::endl;
using std::dec;
using std::hex;
using std::ios;

int main () {
   ios::fmtflags ff = cout.flags ();
   int i = 123;
   cout << "Voreinstellung (hex): " << hex << ff  << endl;
   cout << "i = " << dec << i << endl;
   ios::fmtflags ffa  =  cout.setf (ios::hex,  ios::basefield);
   cout << "Nach ios::hex " << endl;
   cout << "i = " << i << endl;
   ios::fmtflags ffb = cout.setf (ios::showbase | ios::showpos);
   cout << "Nach zusaetzlich ios::showbase und "
                                   "ios::showpos "  << endl;
```

```
    cout << "i = " << i << endl;
    cout << "Ruecksetzen in zwei Schritten" << endl;
    cout.flags (ffb);
    cout << "i = " << i << endl;
    cout.flags (ffa);
    cout << "i = " << i << endl;
}//main


/*   Ausgabe:

Voreinstellung (hex):  201
i = 123
Nach ios::hex
i = 7b
Nach zusaetzlich ios::showbase und ios::showpos
i = 0x7b
Ruecksetzen in zwei Schritten
i = 7b
i = 123
*/
```

```
//
//  Beispiel zur formatierten Ausgabe
//
#include <fstream>
#include <string>
using  std::ofstream;
using  std::endl;
using  std::ios;

#define D(A)  T << #A << endl; A
ofstream T ("Ausgabe_formata.txt");

int main () {
   D (bool  b = 1;)
   D (T.setf (ios::boolalpha);)
   D (T << "b = " << b << endl << endl;)

   D (int i = 471;)
   D (T << "i = " << i << endl << endl;)

   D (T.setf (ios::showbase | ios::showpos | ios::uppercase);)
   D (T << "i = " << i << endl << endl;)

   D (T.setf (ios::hex, ios::basefield);)
   D (T << "i = " << i << endl << endl;)

   D (T.unsetf (ios::uppercase | ios::showbase);)
   D (T.setf (ios::dec,  ios::basefield);)
   D (T.setf (ios::left,  ios::adjustfield);)
   D (T.fill ('*');)
   D (T << "fill char: " << T.fill () << endl;)
   D (T.width (10); T << i << endl << endl;)
   D (T << i << endl << endl);

   D (T.setf (ios::right,  ios::adjustfield);)
   D (T.width (10);  T << i << endl << endl;);
```

```
   D (T.setf (ios::internal,  ios::adjustfield);)
   D (T.width (10);  T << i << endl << endl; )

   D (double d = 23114.41415908123456789012345 67;)

   D (T << "prec = " << T.precision () << endl;)
   D (T.setf (ios::scientific,  ios::floatfield);)
   D (T << "d = " << d << endl << endl;)

   D (T.setf (ios::fixed,  ios::floatfield);)
   D (T << "d = " << d << endl << endl;)

   D (T.unsetf (ios::showpos);)
   D (T.precision (20);)
   D (T << "prec = " << T.precision() << endl;)
   D (T << "d = " << d << endl << endl;)

   D (double e = 1234.0;)
   D (T << "e = " << e << endl << endl;)

   D (T.setf (ios::showpoint);)
   D (T << "e = " << e << endl << endl;)

   D (T.setf (ios::scientific,  ios::floatfield);)
   D (T << "d = "<< d << endl << endl;)

   D (T.setf (ios::fixed,  ios::floatfield);)
   D (T << "d = " << d << endl << endl;)

   D (std::string s = "Ein wenig Text!";)
   D (T.width (10);  T << s << endl << endl;)

   D (T.width (40); T << s << endl << endl;)

   D (T.setf (ios::left,  ios::adjustfield);)
   D (T.width (40);  T << s << endl << endl;)
}//main
```

Datei: Ausgabe_formata.txt

```
bool b = 1;
T.setf (ios::boolalpha);
T << "b = " << b << endl << endl;
b = true

int i = 471;
T << "i = " << i << endl << endl;
i = 471

T.setf (ios::showbase | ios::showpos | ios::uppercase);
T << "i = " << i << endl << endl;
i = +471

T.setf (ios::hex,  ios::basefield);
T << "i = " << i << endl << endl;
i = 0X1D7

T.unsetf (ios::uppercase | ios::showbase);
T.setf (ios::dec,  ios::basefield);
T.setf (ios::left,  ios::adjustfield);
T.fill ('*');
T << "fill char:  " << T.fill () << endl;
fill char:  *
T.width (10); T << i << endl << endl;
+471******

T << i << endl << endl
+471
```

T.setf (ios::right, ios::adjustfield);
T.width (10); T << i << endl << endl;
********+471**

T << i << endl << endl;
+471

T.setf (ios::internal, ios::adjustfield);
T.width (10); T << i << endl << endl;
+*****471**

T << i << endl << endl;
+471

double d = 23114.414159081234567890123456 7;
T << "prec = " << T.precision () << endl;
prec = +6
T.setf (ios::scientific, ios::floatfield);
T << "d = " << d << endl << endl;
d = +2.311441e+004

T.setf (ios::fixed, ios::floatfield);
T << "d = " << d << endl << endl;
d = +23114.414159

T.unsetf (ios::showpos);
T.precision (20);
T << "prec = " << T.precision() << endl;
prec = 20
T << "d = " << d << endl << endl;
d = 23114.414159081236000000 00

double e = 1234.0;
T << "e = " << e << endl << endl;
e = 1234.00000000000000000

T.setf (ios::showpoint);
T << "e = " << e << endl << endl;
e = 1234.00000000000000000

T.setf (ios::scientific, ios::floatfield);
T << "d = "<< d << endl << endl;
d = 2.3114414159081236000e+004

T.setf (ios::fixed, ios::floatfield);
T << "d = " << d << endl << endl;
d = 23114.414159081236000000

std::string s = "Ein wenig Text!";
T.width (10); T << s << endl << endl;
Ein wenig Text!

T.width (40); T << s << endl << endl;
****************************Ein wenig Text!**

T.setf (ios::left, ios::adjustfield);
T.width (40); T << s << endl << endl;
Ein wenig Text!**************************

Manipulatoren

boolalpha	**noboolalpha**
showbase	**noshowbase**
showpoint	**noshowpoint**
showpos	**noshowpos**
skipws	**noskipws**
uppercase	**nouppercase**
unitbuf	**nounitbuf**

dec, hex, oct

left, right, internal

fixed, scientific, hexfloat, defaultfloat

flush, ends, endl, ws

Einbindung von <iomanip> ist erforderlich für Manipulatoren mit Parametern

setiosflags (fmtflags)
resetiosflags (fmtflags)
setfill (char)
setprecision (int)
setw (int)
setbase (int)

Aus Standard:
template <class charT>
T11 **quoted (const charT* s, charT delim=charT('"'),**
charT escape=charT('\\'));

```
//  Beispiel zu quoted aus  N3654
//  VS  Compiler version: 19.00.22318(x86)

#include <iostream>
#include <sstream>
#include <iomanip>
#include <cassert>
using namespace std;

int main () {
   std::stringstream ss;
   std::string original = "foolish me";
   std::string round_trip;

   ss << quoted(original);
   ss >> quoted(round_trip);

   std::cout << original;     // outputs: foolish me
   std::cout << round_trip;   // outputs: foolish me

   assert(original == round_trip); // assert will not fire

   // outputs: "She said \"Hi!\""
   std::cout << quoted("She said \"Hi!\"");
}//main


/*  Ausgabe

foolish me
foolish me
"She said \"Hi!\""
/*
```

```
// Beispiel zu setiosflags und resetiosflags
#include <iostream>
#include <iomanip>
using std::cout;  using std::endl;  using std::ios;
using std::setiosflags;  using std::resetiosflags;

int main () {
   int i = 123;
   cout << "Voreinstellung: " << cout.flags () << endl;
   cout << "i = " << i << endl;
   //  cout << resetiosflags (ios::dec);
   cout << setiosflags (ios::hex);
   cout << "Nach ios::hex: " << endl;
   cout << "i = " << i << endl;
   cout << setiosflags (ios::showbase | ios::showpos);
   cout << "Nach zusätzlich ios::showbase und ios::showpos: "
        << endl;
   cout << "i = " << i << endl;
   cout << "Rücksetzen in zwei Schritten" << endl;
   cout << resetiosflags (ios::showbase | ios::showpos);
   cout << "i = " << i << endl;
   cout << resetiosflags (ios::hex);
   cout << "i = " << i << endl;
}//main
/* Ausgabe:
Voreinstellung:  513
i = 123
Nach ios::hex
i = 123                    WARUM?     ios::hex  wirkungslos?
Nach zusaetzlich ios::showbase und ios::showpos
i = +123
Ruecksetzen in zwei Schritten
i = 123
i = 123     */
```

```
//
//  formatierte Ausgabe,  Nutzung von Manipulatoren
//
#include <fstream>
#include <string>
#include <iomanip>
using  std::endl;


#define D(A) T << #A << endl; A

std::ofstream T ("Ausgabe_formatab.txt");

int main () {
   D (bool  b = true;)
   D (T << "b = " << b << endl << endl;)
   D (T << std::boolalpha;)
   D (T << "b = " << b << endl << endl;)

   D (int i = 47;)
   D (T << "i = " << i << endl << endl;)

   D (T << std::showbase << std::showpos << std::uppercase;)
   D (T << "i = " << i << endl << endl;)

   D (T << std::hex;)
   D (T << "i = " << i << endl << endl;)

   D (T << std::oct;)
   D (T << "i = " << i << endl << endl;)

   D (T << std::dec << std::left << std::setfill ('*');)
   D (T << "fill char: " << T.fill () << endl;)
   D (T << std::setw (10) << i << endl << endl;)
```

```
    D (T << std::right;)
    D (T << std::setw (10) << i << endl << endl;)

    D (T << std::internal;)
    D (T << std::setw (10) << i << endl << endl;)

    D (double d = 23114.41415908123456789012345 67;)
    D (T << "prec = " << T.precision () << endl;)
    D (T << std::scientific;)
    D (T << "d = " << d << endl << endl;)

    D (T << std::fixed << "d = " << d << endl << endl;)

    D (T << std::noshowpos << std::setprecision (20);)
    D (T << "prec = " << T.precision () << endl;)
    D (T << "d = " << d << endl << endl;)

    D (double e = 1234.0;)
    D (T << "e = " << e << endl << endl;)

    D (T << std::noshowpoint << "e = " << e << endl << endl;)

    D (T << std::scientific << "d = "<< d << endl << endl;)
    D (T << std::fixed << "d =" << d << endl << endl;)

    D (std::string s = "Ein wenig Text!";)
    D (T << std::setw (10) << s << endl << endl;)
    D (T << std::setw (40) << s << endl << endl;)
    D (T << std::left;)
    D (T << std::setw (40) << s << endl << endl;)
}//main
```

Datei: Ausgabe_formatab.txt

```
bool b = true;
T << "b = " << b << endl << endl;
b = 1

T << std::boolalpha;
T << "b = " << b << endl << endl;
b = true

int i = 47;
T << "i = " << i << endl << endl;
i = 47

T << std::showbase << std::showpos << std::uppercase;
T << "i = " << i << endl << endl;
i = +47

T << std::hex;
T << "i = " << i << endl << endl;
i = 0X2F

T << std::oct;
T << "i = " << i << endl << endl;
i = 057

T << std::dec << std::left << std::setfill ('*');
T << "fill char: " << T.fill () << endl;
fill char: *
T << std::setw (10) << i << endl << endl;
+47*******
```

```
T << std::right;
T << std::setw (10) << i << endl << endl;
*******+47

T << std::internal;
T << std::setw (10) << i << endl << endl;
+*******47

double d = 23114.414159081234567890123456 7;
T << "prec = " << T.precision () << endl;
prec = +6
T << std::scientific;
T << "d = " << d << endl << endl;
d = +2.311441E+004

T << std::fixed << "d = " << d << endl << endl;
d = +23114.414159

T << std::noshowpos << std::setprecision (20);
T << "prec = " << T.precision () << endl;
prec = 20
T << "d = " << d << endl << endl;
d = 23114.414159081236000000 00

double e = 1234.0;
T << "e = " << e << endl << endl;
e = 1234.00000000000000000 0

T << std::noshowpoint << "e = " << e << endl << endl;
e = 1234.00000000000000000 0
```

T << std::scientific << "d = "<< d << endl << endl;
d = 2.3114414159081236000E+004

T << std::fixed << "d =" << d << endl << endl;
d =23114.414159081236000000

std::string s = "Ein wenig Text!";
T << std::setw (10) << s << endl << endl;
Ein wenig Text!

T << std::setw (40) << s << endl << endl;
****************************Ein wenig Text!**

T << std::left;
T << std::setw (40) << s << endl << endl;
Ein wenig Text!*************************

Schreiben eigener Manipulatoren

Im Standard findet man etwa folgende Definition für den Ausgabeoperator <<

```
template <class charT,  class traits>
basic_ostream <charT,  traits>&  operator <<
      (basic_ostream <charT,  traits>& (*pf)
                  (basic_ostream<charT, traits>&))
//   calls  *pf (*this)
//   returns   *this
```

Eine Spezialisierung für das Template endl sieht etwa so aus.

```
template <class charT,  class traits>
std::basic_ostream <charT,  traits>&
std::endl  (std::basic_ostream <charT,  traits>&  strm) {
      strm.put (strm.widen ('\n'));
      strm.flush ();
      return strm;
}
```

Für die Klasse ostream erhält man:

```
std::ostream&  std::endl  (std::ostream&   strm) {
      strm.put  ('\n');
      strm.flush ();
      return strm;
}
```

Spezialisierung und Verallgemeinerung:

```
ostream& ostream::operator<< (ostream& (*pf) (ostream&) {
      return (*pf) (*this);
}
```

Das beschriebene Rezept kann man sofort für die Definition eines eigenen Zeilenendemanipulators nutzen.

```
//   nl.cpp
#include <iostream>
using std::cout;

std::ostream& nl (std::ostream& os = std::cout) {
   return os << '\n';
}//nl

int main() {
   cout << "Zeilenbruch";  nl (cout);
   cout << "nach";   nl (cout);
   cout << "jedem";  nl ();
   cout << "Wort.";  nl ();

   cout << "Zeilenbruch" << nl << "nach" << nl
        << "jedem" << nl << "Wort." << nl;
}//main

/* Ausgabe:
Zeilenbruch
nach
jedem
Wort.
Zeilenbruch
nach
jedem
Wort.
*/
```

```
//  Eine Variante von  nl.cpp
#include <iostream>
#include <sstream>
using namespace std;

int i = 100;
string s1 = "hallo ";    string s2 = "   ";
stringstream inout;

ostream& nl (ostream& os) {  //  nl ist eine einparametrige
   inout << s1 << ++i << s2;    //  Funktion
   return  os << '\n';}

int main() {
       cout << "Zeilenbruch";  nl (cout);
       cout << "nach";   nl (cout);
       cout << "jedem" << nl;
       cout << "Wort." <<  nl;
       cout << "Zeilenbruch" << nl << "nach" << nl
            << "jedem" << nl << "Wort." << nl;
       cout << inout.str () << endl;
}//main
/*
Zeilenbruch
nach
jedem
Wort.
Zeilenbruch
nach
jedem
Wort.
hallo 101  hallo 102  hallo 103  hallo 104  hallo 105  hallo 106
hallo 107  hallo 108
*/
```

```
//  Eigenes defaultfloat
#include <iostream>

using std::cout;
using std::ios_base;
using std::scientific;

inline
ios_base& floatnormal (ios_base& io) {
  io.setf ((ios_base::fmtflags) 0, ios_base::floatfield);
  return io;
}

int main () {
  ios_base::fmtflags  flags  =  cout.flags ();

  //  Näherung für pi
  double npi = 22.0/7.0;

  cout << "npi = " << scientific
       << npi * 1000 << '\n';

  cout << "npi = " << floatnormal
       << npi * 1000 << '\n';

  cout.flags (flags);
}


/*  Ausgabe:
npi = 3.142857e+003
npi = 3142.86
*/
```

```
//Beispiel eines parametrisierten Manipulators
#include <iostream>
#include <iomanip>
using std::cout;
using std::endl;
using std::ostream;
using std::setfill;

class outm {
public:
   outm (ostream& (*pf)(ostream&, int), int i): fn (pf), ia (i) {}
   friend ostream& operator << (ostream& os, outm m)
      {return m.fn (os,  m.ia);}
private:
   int ia;
   ostream& (*fn)(ostream&,  int);
};

ostream& padn (ostream& os,  int n) {
   while (os && n--)
      os << os.fill ();
   return os;
}

outm pad (int n) {
   return  outm (&padn,  n);
}

int main () {
   cout << setfill('-') << "11" << pad (20) << "22" << endl;
}//main

/* Ausgabe
11----------------22      */
```

```
//  Weiterer Manipulator

#include <iostream>
using namespace std;

class doppel {
private:
   int wert;
public:
   doppel (int zudoppeln) : wert (zudoppeln) {}
   ostream& operator()(ostream &) const;
};

inline
ostream& operator<<(ostream& out,  doppel zahl) {
   return zahl (out);
}

ostream&  doppel::operator()(ostream& out) const {
   out << 2*wert;
   return out;
}


int main () {
   cout << doppel (24) << endl;
}


/*  Ausgabe:
48
*/
```

```
//  Beispiel von Stroustrup
#include <iostream>
#include <sstream>

using std::cout;
using std::ostream;
using std::ostringstream;
using std::ios_base;

class Form;

struct Bound_Form {
   const Form& f;
   double val;
};

class Form {
   friend ostream& operator<< (ostream&, Bound_Form&);
public:
   explicit Form (int p = 6, ios_base::fmtflags f = 0)
                     : prc{p}, fmt{f} {}
   Bound_Form Form::operator() (double d) const {
      return Bound_Form {*this, d};
   }
   Form& scientific() {fmt=ios_base::scientific; return *this;}
   Form& fixed()      {fmt=ios_base::fixed;  return *this;}
   Form& general()    {fmt = 0; return *this;}
   Form& precision (int p)  {prc = p; return *this;}
private:
   int prc;
   int fmt;

};
```

```
ostream& operator << (ostream& os,  Bound_Form& bf) {
    ostringstream s;
    s.precision (bf.f.prc);
    s.setf (bf.f.fmt, ios_base::floatfield);
    s << bf.val;
    return os << s.str();
}


void f (double d) {
    Form gen4 {4};
    Form sci8;
    sci8.scientific().precision (8);
    cout << d << "  " << gen4 (d) << "  "  << sci8 (d) << "  " <<
d << '\n';
    Form sci (10, ios_base::scientific);
    cout << d << "  " << gen4 (d) << "  " << sci (d) << "  "
            << d << '\n';
}


int main () {
    f (1234.56789);
}


/*
1234.57  1235  1.23456789e+003  1234.57
1234.57  1235  1.2345678900e+003  1234.57
*/
```

Stringstream

```
//  Beispiel zu stringstream
#include <iostream>
#include <sstream>
#include <string>
using namespace std;

int main () {
	stringstream inout;
	inout << "Dies steht in der ersten Zeile." << endl;
	inout << "Und dies in der zweiten Zeile." << endl;
	inout << "Endlich:  die letzte Zeile." << endl;
	string s;
	getline (inout, s);
	cout << "1:  " << s << endl;
	getline (inout, s);
	cout << "2:  " << s << endl;
	getline (inout, s);
	cout << "3:  " << s << endl;
	cout << endl << "Gesamtausgabe von inout:"
		<< endl << inout.str ();
}//main

/*  Ausgabe:
1:  Dies steht in der ersten Zeile.
2:  Und dies in der zweiten Zeile.
3:  Endlich:  die letzte Zeile.

Gesamtausgabe von inout:
Dies steht in der ersten Zeile.
Und dies in der zweiten Zeile.
Endlich:  die letzte Zeile.
*/
```

```
//  Beispiel zu stringstream
//  Zahlanpassung mittels stringstream

#include <iostream>
#include <iomanip>
#include <sstream>


int main () {
      std::istringstream s0 ("47 1.414 This is a test!");
      int         i;
      double    d;
      char       buf [100];

      s0 >> i >> d >> buf;       // Begrenzung durch Weißraum

      std::cout << "i = " << i << ", d = " << d
                  << ", buf = *" << buf << '*' << std::endl;

      // Ausgabe des Restes
      std::cout << s0.rdbuf () << std::endl << std::endl;

      double d1 = 3.12345678912545678912345678 9;
      std::stringstream s1 (std::ios::in | std::ios::out);
      s1 << std::setprecision (14) << d1 << std::endl;
      s1.seekg (0, std::ios::beg);
      double d2 = 0;
      s1 >> d2;

      std::stringstream s2 (std::ios::in | std::ios::out);
      s2 << std::setprecision (12) << d2 << std::endl;
      s2.seekg (0, std::ios::beg);
      double d3 = 0;
      s2 >> d3;
```

```
        std::cout << "d1 = " << std::setprecision (16)
                  << d1 << '\n';
        std::cout << "d2 = " << std::setprecision (16)
                  << d2 << '\n';
        std::cout << "d3 = " << std::setprecision (16)
                  << d3 << '\n';

}//main



/*  Ausgabe:

i = 47, d = 1.414, buf = *This*
 is a test!

d1 = 3.123456789125457
d2 = 3.1234567891255
d3 = 3.12345678913

*/
```

Datenpuffer

```
//  Kopieren einer Datei  unter Nutzung des Datenpuffers

#include <fstream>
#include <iostream>
using std::cout;
using std::ifstream;

int main () {
      ifstream in  ("nl.cpp");
      cout << in.rdbuf ();
}//main


/*  Ausgabe:
//   nl.cpp:  Beispiel eines einfachen Manipulators
#include <iostream>
using std::ostream;
using std::cout;

ostream& nl (ostream& os = cout) {return   os  << '\n';}//nl

int main () {
      cout << "Zeilenbruch";  nl (cout);
      cout << "nach";   nl (cout);
      cout << "jedem";  nl ();
      cout << "Wort.";  nl ();

      cout << "Zeilenbruch" << nl << "nach" << nl
            << "jedem" << nl << "Wort." << nl;
}//main
*/
```

```
//  Positionieren in Streams

#include <iostream>
#include <fstream>
using  std::ios;
using  std::streampos;
using  std::cout;
using  std::endl;
using  std::ifstream;

int main () {

      ifstream in ("rdbuf0.cpp");

      in.seekg  (0,  ios::end);          // Dateiende
      streampos sp = in.tellg ();       // Dateigröße
      cout << "Dateigröße = " << sp << endl;

      in.seekg  (-100,  ios::end);
      streampos  sp2  =  in.tellg ();

      in.seekg  (0,  ios::beg);              // Dateianfang
      cout << in.rdbuf ();                   // Ausgabe der Datei
      in.seekg  (sp2);                       // Positionieren auf sp2

      // Letzte 100 Zeichen ausgeben
      cout << endl << endl << "Letzte 100 Zeichen:" << endl;
      cout << endl << in.rdbuf () << endl;

}//main
```

```
/*  Ausgabe:

Dateigröße = 342
//  rdbuf0.cpp
//  Direktes Schreiben in Puffer

#include <fstream>
#include <iostream>
using std::ifstream;
using std::cout;

int main () {
        ifstream in ("rdbuf0.cpp");
        while (in.get (*cout.rdbuf ())) {
                while (in.peek () == '\n') {
                  in.ignore (); // Zeilenende vernichten
                  cout << '\n';
                }
        }
}//main


Letzte 100 Zeichen:

{
                  in.ignore (); // Zeilenende vernichten
                  cout << '\n';
                }
        }
}//main
*/
```

```
//  Beispiel zur gemeinsamen Puffernutzung

#include <iostream>
#include <fstream>

using std::ostream;
using std::istream;
using std::filebuf;
using std::ios;
using std::cout;
using std::endl;


int main()  {
   filebuf  buffer;
   ostream output (&buffer);
   istream input (&buffer);
   buffer.open ("beispiel.txt",  ios::in | ios::out | ios::trunc);

   for (int i = 1;   i <= 4;   ++i) {
      output << i << ". Zeile" << endl;

      // Ausgabe der Gesamtdatei
      input.seekg (0);          // zurück an den Anfang
      char c;
      while (input.get (c)) {
         cout.put (c);
      }
      cout << endl;
      input.clear ();          // löschen  eofbit und  failbit
   }//for
}//main
```

/* Ausgabe:

1. Zeile

1. Zeile
2. Zeile

1. Zeile
2. Zeile
3. Zeile

1. Zeile
2. Zeile
3. Zeile
4. Zeile

***/**

Statusfunktionen

```
iostate rdstate ()
bool  good ()
bool  eof ()
bool  fail ()
bool  bad ()
void  clear ()
void  clear (iostate)
void  setstate (iostate)
```

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

int main() {
   int i;
   ios::iostate status;
   while (true) {        // Schleifenabbruch mit break
       cout << "Abbruch oder Zahl:  ";
       cin >> i;
       status = cin.rdstate ();
       cout << "status   = " << status << endl;
       cout << "good () = " << cin.good () << endl;
       cout << "eof ()    = " << cin.eof () << endl;
       cout << "fail ()    = " << cin.fail () << endl;
       cout << "bad ()   = " << cin.bad () << endl;
       if (cin.eof ())
          break;              // Abbruch
       if (status) {
           cin.clear ();      // Fehlerbits zurücksetzen
           cin.get ();        // ggf. fehlerhaftes Zeichen entfernen
       } else
```

```
            cout << "Zahl = " << i << endl;
    }
}//main

/*  Ausgabe:

Abbruch oder Zahl:  23
status   = 0
good () = 1
eof ()    = 0
fail ()    = 0
bad ()   = 0
Zahl = 23
Abbruch oder Zahl:  x45   // nur lesen von x
status   = 2
good () = 0
eof ()    = 0
fail ()    = 1
bad ()   = 0
Abbruch oder Zahl:  status = 0  //lesen von 45
good () = 1
eof ()    = 0
fail ()   = 0
bad ()  = 0
Zahl = 45
Abbruch oder Zahl:  ^Z
status   = 3
good () = 0
eof ()    = 1
fail ()   = 1
bad ()  = 0

*/
```

Öffnen von Dateien

Öffnungsanzeigen

app,
ate,
binary,
in,
out,
trunc

Positionieren

Bezuggröße
- **beg,**
- **cur,**
- **end**

Funtionen:

```
pos_type   tellg ()
pos_type   tellp ()
istream&    seekg  (pos_type)
istream&    seekg (off_type,  seekdir)
ostream&   seekp (pos_type)
ostream&   seekp (off_type, seekdir)
```

```
//  Lesen und Schreiben mit read und write
#include <iostream>
#include <fstream>

using std::fstream;
using std::ios;
using std::cout;



int main() {
   {fstream fstr ("daten.txt", ios::out);
   }  //  Datei daten.txt wird geschlossen

   fstream fstr ("daten.txt", ios::in | ios::out);

   //  Füllen von daten.txt
   fstr << "abcdefghij0123456789";

   // Zurück zum Anfang
   fstr.seekg (0, ios::beg);

   const int buffer_size = 100;
   char buffer [buffer_size];

   fstr.read (buffer, 5);

   // 3 Positionen zurück zum Schreiben
   fstr.seekp (-3, ios::cur);

   char buf1 [] = "ABC";
   fstr.write (buf1, 3);

   ios::pos_type pos = fstr.tellg();
```

```
    char buf2 [] = "DEF";
    fstr.write (buf2, 3);


    // Neue Schreibposition
    fstr.seekp (pos + fstream::off_type(2));

    char buf3[] = "++";
    fstr.write (buf3, 2);
    fstr.seekg (0, ios::end);
    pos = fstr.tellg ();
    fstr.seekg (0, ios::beg);
    fstr.read (buffer, pos);
    buffer [pos] = '\0';
    cout <<"Dateiinhalt = " << buffer << '\n';
}//main

/*
Dateiinhalt = abABCDE++j0123456789

*/
```

```
//  Lesen und Schreiben auf *einer* Datei
#include <fstream>
#include <iostream>
using std::fstream;      using std::ios;
using std::cout;         using std::endl;

int main() {
   // Datei anlegen
   fstream fs ("dateia", ios::out | ios::trunc);
   fs.close ();                    // leere Datei existiert jetzt
   char a;
   // Datei zum Lesen und Schreiben öffnen
   fs.open ("dateia", ios::in | ios::out);
   // schreiben
   for (int i = 48;  i < 123;  ++i)
      fs << (char) (i);
   fs << endl;
   // lesen
   fs.seekg (0);                   // Am Anfang beginnen
   while (!fs.eof ()) {
     fs >> a;
     if (!fs.eof ())
        cout << a;                 // Kontrollausgabe
   }
   cout << endl;
   fs.clear ();
   fs.seekp (8);                   // Position 8 suchen
   fs << "neuer Text ";            // Ab Position 8 überschreiben
   fs.seekg (0);                   // Rücksetzen
   char buf [100];
   fs.getline (buf, 100);          // Zeile lesen
   cout << buf << endl;            // Kontrollausgabe
}//main
```

/* **Ausgabe:**

```
0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[
\]^_`abcdefghijklmnopqrstuvwxyz

01234567neuer Text CDEFGHIJKLMNOPQRSTUVWXYZ[
\]^_`abcdefghijklmnopqrstuvwxyz
```

*/

```
//  Unformatiertes Lesen und Schreiben

#include <fstream>
using std::ifstream;
using std::ofstream;
using std::ios_base;

int main () {
      char buf [19];              // hinreichend groß

      ifstream in ("satire.txt",  ios_base::binary | ios_base::in );
      ofstream out ("kop_sat.txt",  ios_base::binary |
                                                ios_base::out);

      while (in.read (buf,  sizeof buf))
           out.write (buf,  sizeof buf);
      //  Ausgabe des Restes nach Setzen von eofbit
      if  (in.gcount ()  >  0)
           out.write (buf,  in.gcount ());
}
```

Funktionen

istream& istream::read (char* buffer, streamsize anz)
Bis zu anz Zeichen werden in den Puffer gelesen, bei Erreichen des Dateiendes wird das failbit gesetzt.

streamsize istream:: gcount () const;
Liefert die Anzahl der Zeichen, die beim letzten Lesebefehl gelesen wurde.

ostream& ostream::write (char const* buffer, streamsize anz)
Gibt anz Zeichen aus.

Internationalisierung

Die Ein- und Ausgaben für ein Programm möchten den lokalen Gepflogenheiten Rechnung tragen. Hierzu dient in C++ der "locale"-Mechanismus. Insbesondere für die Schreibweise von Zahlen, Geldbeträgen, Datums- und Zeitangaben möchte man an die lokalen Gegebenheiten anpassen.

```
#include <iostream>
#include <locale>

int main () {
      float  a  =  float (0.1);
      float  b  =  float (0.1);
      float  c  =  0;
      std::cout.imbue (std::locale ("german"));
      std::cout  <<  "  a     =  " <<  a  << std::endl;
      std::cout  <<  "  b     =  " <<  b  << std::endl;
      std::cout  <<  "  a*b  =  " <<  a*b  << std::endl;
      c  +=  a*b;   c  -=  a*b;
      std::cout  <<  std::endl
                 <<  "  c   =  "  <<  c  <<  std::endl;
}//main

/*  Ausgabe
  a     =  0,1            Man beachte das Dezimalkomma!
  b     =  0,1
  a*b  =  0,01

  c      =  4,09782e-010                Warum ist c ≠ 0?
*/
```

```
//  Verschiedene Zahldarstellungen
//  C-spezifisch und deutsch-spezifisch

#include <iostream>
#include <iomanip>
#include <locale>
using std::cin;
using std::cout;
using std::endl;
using std::locale;
using std::setprecision;

int main() {
   // Einlesen von Zahlen mittels C locale
   cin.imbue (locale::classic ());
   // Ausgabe von Zahlen gemäß deutscher Tradition
   cout << setprecision (12);
   cout.imbue (locale ("german"));
   // Lesen und Ausgeben von Gleitpunktzahlen
   double value;
   while (cout << "Bitte Zahl: ",  cin >> value) {
      cout <<    "Gelesen:    " << value << endl;
   }
}//main

/*  Ein Beispiellauf:
Bitte Zahl: 9876543
Gelesen:     9.876.543
Bitte Zahl: 0.012345
Gelesen:     0,012345
Bitte Zahl: 1234567.7654
Gelesen:     1.234.567,7654
Bitte Zahl: ^Z
*/
```

```
//  Der Bezeichner für das genutzte  "locale"  wird durch
//  das Betriebssystem festgelegt.  Hier einige Beispiele für //
Windows.


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

int main()  {
        locale native ("");
        locale usa ("American_USA.1252");
        locale holland ("Dutch");
        locale global;
        locale canada ("English_Canada");
        locale GB ("english_United Kingdom");
        locale deutsch ("deu_deu");
        locale kroatisch ("Croatian");
        locale russisch ("Russian_Russia");
        locale polnisch ("polish_Poland.1250");
        locale baskisch ("Basque");
        locale estnisch ("Estonian");

        //  Ausgabe der vollständigen Bezeichnung
        cout << "native      : " << native.name() << endl;
        cout << "classic     : " << locale::classic().name()
                << endl;
        cout << "global      : " << global.name() << endl;
        cout << "holland     : " << holland.name() << endl;
        cout << "usa         : " << usa.name() << endl;
        cout << "canada      : " << canada.name() << endl;
        cout << "britisch    : " << GB.name () << endl;
        cout << "deutsch     : " << deutsch.name () << endl;
        cout << "kroatisch : " << kroatisch.name () << endl;
```

```
    cout << "polnisch  : " << polnisch.name ()  << endl;
    cout << "russisch   : " << russisch.name () << endl;
    cout << "baskisch  : " << baskisch.name ()  << endl;
    cout << "estnisch   : " << estnisch.name () << endl;

}//main



/*  Ausgabe

native          : German_Germany.1252
classic         : C
global          : C
holland         : Dutch_Netherlands.1252
usa             : English_United States.1252
canada          : English_Canada.1252
britisch        : English_United Kingdom.1252
deutsch         : German_Germany.1252
kroatisch       : Croatian_Croatia.1250
polnisch        : Polish_Poland.1250
russisch        : Russian_Russia.1251
baskisch        : Basque_Spain.1252
estnisch        : Estonian_Estonia.1257

*/
```

```
//  Unbekannte Bezeichner für locales führen zum
//  Abbruch während der Laufzeit.

#include <iostream>
#include <fstream>
#include <locale>
#include <string>
#include <stdexcept>

using namespace std;
int main( ) {
  try {
    locale loc ("");
    cout << "locale name = " << loc.name () << endl
         << endl;

    locale locf  ("french");
    locale locus ("english-american");
    locale locb  ("portuguese-brazilian");

    cout.imbue (locf);
    cout << "Geschützter Leerraum in Zahldarstellung\n";
    cout << "2343.14 (franzoesisch) = " << 2343.14
         << endl;
    cout << "name of locf = " << locf.name () << endl
         << endl;

    cout.imbue (locus);
    cout << "2343.14 (englisch) = " << 2343.14 << endl;
    cout << "name of locus = " << locus.name () << endl
         << endl;
```

```
    cout.imbue (locb); //  Brazilian formatting
    cout << "2343.14 (brasilianisch) = " << 2343.14
         << endl;
    cout << "name of locb = " << locb.name () << endl
         << endl;
  }
   catch (runtime_error& e) {
    // Ein unbekannter locale Name führt zu einer
    // exception.
    cerr << "Error: " << e.what () << endl;
  }
}//main


/*
locale name = German_Germany.1252

Geschützter Leerraum in Zahldarstellung
2343.14 (franzoesisch) = 2 343,14
name of locf = French_France.1252

2343.14 (englisch) = 2,343.14
name of locus = English_United States.1252

2343.14 (brasilianisch) = 2.343,14
name of locb = Portuguese_Brazil.1252
*/


/*
unter g++  (MinGW)
locale name = C
Error: locale::facet::_S_create_c_locale name not valid
*/
```

```
#include <locale>
#include <iostream>
#include <string>

using std::cout;
using std::endl;
using std::boolalpha;

class germanBool : public std::numpunct_byname<char> {
public:
   germanBool (const std::string& name)
     : std::numpunct_byname<char>(name) {}
protected:
   virtual std::string do_truename () const {
      return "wahr";
   }
   virtual std::string do_falsename () const {
      return "falsch";
   }
};

int main() {
   bool bb = 0;
   std::locale loc (std::locale(""), new germanBool (""));
   std::cout.imbue (loc);
   std::cout << boolalpha << true << std::endl;
   cout << boolalpha << bb << endl;
}

/*
wahr
falsch
*/
```

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

class Separator_facet: public numpunct<char> {
public:
   explicit Separator_facet( size_t refs = 0):
                        numpunct<char>(refs){}
protected:
   virtual string do_grouping() const {return "\2";}
};

int main() {
   cout << "urspruengliche Ausgabe\n";
   const int million = 1000000;
   const double dn  = 1234.56789;
   cout << "Million = " << million << '\n'
        << fixed << setprecision (5) << dn << endl;
   cout << "Nach Aenderung auf Gruppengroesse 2!\n";
   locale separator_locale (cout.getloc(),  new
                                  Separator_facet);
   cout.imbue (separator_locale);
   cout << "Million = " << million << '\n'
        << fixed << setprecision (5) << dn << endl;
}
/*
urspruengliche Ausgabe
Million = 1000000
1234.56789
Nach Aenderung auf Gruppengroesse 2!
Million = 1,00,00,00
12,34.56789
*/
```

```
//  Währungssymbole

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

int main () {
   locale  locus ("english-american");
   locale  locgb ("english_United Kingdom");
   locale  locgm ("german_Germany");

   wcout.imbue (locus);
   wcout << 98765 << endl;
    wstring dollars = use_facet<moneypunct<wchar_t>>
                                        (locus).curr_symbol ();
    wstring pounds = use_facet<moneypunct<wchar_t>>
                                        (locgb).curr_symbol ();
    wstring euros = use_facet<moneypunct<wchar_t>>
                                        (locgm).curr_symbol ();
    wcout << L"Währungssymbol in US:  " << dollars
          << endl;
    wcout << L"Währungssymbol in GB:  " << pounds
          << endl;
    wcout << L"Währungssymbol in DE:  " << euros
          << endl;
}//main

/*  Ausgabe unter Lucida Console unter CP  1252

98,765
Währungssymbol in US:  $
Währungssymbol in GB:  £
Währungssymbol in DE:  €
*/
```

```
// Elemente zur Beschreibung von Geldbeträgen

#include <iostream>
#include <locale>
#include <string>

using namespace std;

string printPattern (moneypunct<char>::pattern& pat) {
    string s (pat.field);  // pat.field is a char[4]
    string r;

    for (int i = 0; i < 4; ++i) {
      switch (s[i]) {
      case moneypunct<char>::sign:
           r += "sign ";
           break;
      case moneypunct<char>::none:
           r += "none ";
           break;
     case moneypunct<char>::space:
          r += "space ";
          break;
     case moneypunct<char>::value:
          r += "value ";
          break;
     case moneypunct<char>::symbol:
          r += "symbol ";
          break;
     }
    }
  return(r);
}
```

```
int main () {

  //locale loc ("danish");
  //locale loc ("english");
    locale loc ("french");

  const moneypunct<char>& punct =
     use_facet<moneypunct<char>>(loc);

  cout << "Decimal point:       " << punct.decimal_point ()
       << '\n'
     << "Thousands separator: " << punct.thousands_sep
        () << '\n'
     << "Currency symbol:      " << punct.curr_symbol ()
     << '\n'
     << "Positive sign:             " << punct.positive_sign ()
     << '\n'
     << "Negative sign:             " << punct.negative_sign ()
     << '\n'
     << "Fractional digits:        " << punct.frac_digits ()
     << '\n'
     << "Positive format:          "
     << printPattern (punct.pos_format ()) << '\n'
     << "Negative format:          "
     << printPattern (punct.neg_format ()) << '\n';

  // Grouping is represented by a string of chars,
  // but the meaning of each char is its integer value,
  // not the char it represents.
  string s = punct.grouping ();
  for (string::iterator p = begin (s); p != end (s); ++p)
    cout << "Groups of: " << (int)*p << '\n';
}
```

/* Dänisch:

Decimal point: ,
Thousands separator: .
Currency symbol: kr.
Positive sign:
Negative sign: -
Fractional digits: 2
Positive format: symbol sign space value
Negative format: symbol sign space value
Groups of: 3
***/**

/* Amerikanisch

Decimal point: .
Thousands separator: ,
Currency symbol: $
Positive sign:
Negative sign: -
Fractional digits: 2
Positive format: sign symbol value none
Negative format: sign symbol value none
Groups of: 3
***/**

/* Französisch:

Decimal point: ,
Thousands separator:
Currency symbol: €
Positive sign:
Negative sign: -
Fractional digits: 2
Positive format: sign value space symbol
Negative format: sign value space symbol
Groups of: 3
***/**

```
//  Sortierfolge

#include <iostream>
#include <locale>
#include <string>
#include <vector>
#include <algorithm>

using namespace std;

bool lessthan (const string& s1, const string& s2) {

    const collate<char>& col =
      use_facet<collate<char>>(locale ()); //  global locale

    const char* pb1 = s1.data ();
    const char* pb2 = s2.data ();

return (col.compare (pb1,  pb1 + s1.size (),
                     pb2,  pb2 + s2.size ()) < 0);
}//lessthan

int main( ) {

    // Drei kurze Zeichenketten
    string s1 = "diät";
    string s2 = "dich";
    string s3 = "dies";

    vector<string> v;
    v.push_back (s1);
    v.push_back (s2);
    v.push_back (s3);
```

```
    //  Sortiere nach Standard
    cout << "Standardsortierung.\n";
    sort (begin (v), end (v));
    for (auto p : v)
        cout << p << endl;
    cout << endl;

    // Nutze deutsche Sortierfolge
    cout << "deutsche Sortierfolge.\n";
    locale::global (locale ("german"));
    sort (begin (v), end (v), lessthan);
    for (auto i : v)
       cout << i << endl;
}//main



/*

Standardsortierung.
dich
dies
diät

deutsche Sortierfolge.
diät
dich
dies

*/
```