```
/**
 * @file        Huffman.cpp
 * @synopsis    Definition Klassen BitFileOut und BitFileIn.
 * @author      Jan Tammen (FH Konstanz), <jan.tammen@fh-konstanz.de>
 * @author      Christoph Eck (FH Konstanz), <christoph.eck@fh-konstanz.de>
 * @date        2005-06-20
 */

#include "BitFileIO.h"

// {{{ BitFileOut::BitFileOut ()
/**
 * Konstruktor: Zieldatei oeffnen
 * @param fn    String mit Zieldatei-Namen
 */
BitFileOut::BitFileOut (const string& fn) :
    fp(fn.c_str(), ios::out | ios::binary),
    obc(0),
    och(0)
{
    if (fp.fail())
        throw FileNotWriteableException("Zieldatei nicht schreibbar!");

    /// Status-Info: Anzahl der zu ignorierenden Bits auf 0 setzen
    writeBits(string(8, '0'));
}
// }}}

// {{{ BitFileOut::~BitFileOut ()
/**
 * Destruktor.
 *
 * Sofern noch Daten im Puffer vorhanden sind, werden
 * diese noch auf die Datei geschrieben. Anschliessend
 * wird die Anzahl der zu ignorierenden Bits entsprechend
 * angepasst.
 */
BitFileOut::~BitFileOut ()
{
    if (obc != 0)
    {
        int ign = (8-obc);
        fp << och;
        obc = 0;
        och = 0;

        /// Anzahl der zu ignorierenden Bits an den Dateianfang schreiben
        fp.seekp(0);
```



```
        string strBits = BitFileOut::decToBin(ign);

        /// String von vorne mit Nullen fuellen
        strBits.insert(strBits.begin(), (8-strBits.length()), '0');
        writeBits(strBits);
    }

    fp.close();
}
// }}}

// {{{ BitFileOut::writeBits ()
/**
 * Bits in die Datei schreiben.
 * @param str   String mit der zu schreibenden Bitfolge
 */
void BitFileOut::writeBits (const string& str)
{
    for (int i = 0; i < (int) str.length(); i++)
    {
        int bit = str[i] - '0';
        och |= bit << (7-obc);

        if (++obc == 8)      /// Puffer voll, auf Datei schreiben
        {
            fp << och;
            if (fp.fail ())
                throw Exception("Konnte Daten nicht schreiben.");

            obc = 0;
            och = 0;
        }
    }
}
// }}}

// {{{ BitFileOut::decToBin ()
/**
 * Hilfsfunktion: Integer in Binaerdarstellung (String) wandeln
 * @param n Die zu wandelnde Zahl
 */
string BitFileOut::decToBin (unsigned int n)
{
    string b, b2;
    stringstream bs;

    for (; n > 0 ; n /= 2)
    {
```

```
        bs << (n % 2);
        bs >> b2;
        b  = b2 + b;
        bs.clear();
    }
    return b;
}
// }}}

// {{{ BitFileIn::BitFileIn ()
/**
 * Konstruktor: Quelldatei oeffnen.
 * @param fn    String mit Quelldatei-Namen
 */
BitFileIn::BitFileIn (const string& fn) :
    fp(fn.c_str(), ios::in | ios::binary),
    len(0), obc(8), och(0), numIgnore(0)
{
    if (fp.fail() || fp.eof() || !fp.is_open())
        throw FileNotReadableException("Quelldatei nicht lesbar!");

    fp.seekg(0, ios::beg);
    ifstream::pos_type begin_pos = fp.tellg();
    fp.seekg(0, ios::end);
    len = fp.tellg()-begin_pos;
    fp.clear ();
    fp.seekg (0, ios::beg);

    numIgnore = readBits(8);    /// Wieviele Bits am Ende ignorieren?
}
// }}}

// {{{ BitFileIn::getBit ()
/**
 * Ein Bit aus der Quelldatei (bzw. Puffer) holen.
 */
int BitFileIn::getBit ()
{
    /// Wir sind beim letzten Byte angekommen.
    /// Falls wir am Ende Bits ignorieren muessen (numIgnore != 0), wird
    /// hier an der entsprechenden Stelle abgebrochen
    if ((getLength() == fp.tellg()) && numIgnore != 0)
    {
        if ((8-numIgnore) == obc)
            throw int(numIgnore);
    }

    /// Neue Daten von Datei lesen
```



```
    if (obc == 8)
    {
        och = fp.get();
        obc = 0;
    }

    return (och >>(7-obc++)) & 1;
}
// }}}

// {{{ BitFileIn::readBits ()
/**
 * Bel. Anzahl Bits aus Datei lesen.
 * @param n Anzahl der Bits
 */
int BitFileIn::readBits (int n)
{
    int v = 0;
    for (int i = 0; i < n; i++)
        v = (v << 1) | getBit();
    return v;
}
// }}}
```