```
/**
 * @file        Huffman.cpp
 * @synopsis    Definition Klasse Huffman.
 * @author      Jan Tammen (FH Konstanz), <jan.tammen@fh-konstanz.de>
 * @author      Christoph Eck (FH Konstanz), <christoph.eck@fh-konstanz.de>
 * @date        2005-06-09
 */

#include "Huffman.h"

// {{{ Huffman::Huffman ()
Huffman::Huffman () :
    zeichenHaeufigkeiten(256, 0),
    codeTabelle(),
    wurzelKnoten(),
    cntBits(0),
    cntChars(0),
    cntUniqueChars(0) {}
// }}}

// {{{ Huffman::~Huffman ()
Huffman::~Huffman ()
{
    loescheBaum(wurzelKnoten);
}
// }}}

// {{{ Huffman::loescheBaum ()
/**
 * Alle Knoten des Baumes loeschen, Speicher freigeben.
 * @param p Zeiger auf den Startknoten
 */
void Huffman::loescheBaum (const Knoten* p)
{
    if (p != 0)
    {
        loescheBaum( p->getKindLinks() );
        loescheBaum( p->getKindRechts() );
        delete p;
    }
}
// }}}

// {{{ Huffman::codiereDatei ()
/**
 * Eine Datei Huffman-codieren.
 * @param src   String mit dem Quelldatei-Namen
 * @param dest  String mit dem Zieldatei-Namen
```



```
 */
void Huffman::codiereDatei (const string& src, const string& dest)
{
    ifstream srcFileStream(src.c_str(), ios::in);
    if (srcFileStream.fail())
        throw FileNotReadableException("Quelldatei nicht lesbar!");

    /// 1. Zeichen zaehlen, 2. Baum erstellen, 3. Codetabelle erstellen
    zaehleZeichen(srcFileStream);
    erstelleBaum();
    erstelleCodeTabelle();

    /// Fuer die codierte Datei bitweise Schreibeoperationen verwenden
    BitFileOut destFile(dest);
    schreibeHaeufigkeiten(&destFile);

    clog << "Beginne Codieren der Quelldatei: " << src << endl;
    /// Jedes Zeichen der Quelldatei erneut lesen, um es zu codieren
    for (char c; srcFileStream >> noskipws >> c;)
    {
        destFile.writeBits(codeTabelle[c]);     /// Bitcode schreiben
        cntChars++;                             /// Zeichen zaehlen
    }
    srcFileStream.close();

    /// Statistik ausgeben und dann zuruecksetzen
    druckeStatistik();
    cntBits = cntChars = cntUniqueChars = 0;

    clog << "Codieren vollstaendig abgeschlossen, Ergebnisdatei: "
         << dest << endl << endl;
}
// }}}

// {{{ Huffman::druckeStatistik ()
/**
 * Ausgabe der Codierungs-Statistik.
 */
void Huffman::druckeStatistik () const
{
    long int bitRed = (cntChars*sizeof(char)*8)-(cntBits+8);
    long int memChars = cntChars*sizeof(char)*8;

    /// 8 Bit fuer Anzahl der Zeichen + fuer jedes Zeichen 8 Bit +
    /// fuer jede Zeichenhaeufigkeit 32 Bit
    long int memSymFreqSize = 8+(cntUniqueChars*8)+(cntUniqueChars*32);

    cout << endl
```

```
         << "STATISTIK: Codierung" << endl
         << string(76, '=') << endl;

    cout << "  1: Theoretischer Reduktionsfaktor:"
         << endl
         << setw(45) << left << "     Anzahl Zeichen in der Quelldatei: "
         << setw(6)  << left << cntChars
         << " => Speicher: "
         << setw(6)  << left << memChars << " Bit" << endl
         << setw(45) << left << "     Fuer Codierung benoetigte Bitanzahl: "
         << setw(6)  << left << cntBits+8   /// 8 fuer IgnoreBits
         << " => Speicher: "
         << setw(6)  << left << cntBits+8 << " Bit" << endl
         << "     == Reduktion: " << bitRed << " Bit"
         << ", Faktor: " << double((bitRed*100)/memChars) << "%"
         << endl << endl;

    cout << "  2: Praktischer Reduktionsfaktor (inkl. Haeufigkeitsverteilung):"
         << endl
         << setw(45) << left << "     Anzahl Zeichen in der Quelldatei: "
         << setw(6)  << left << cntChars
         << " => Speicher: "
         << setw(6)  << left << memChars << " Bit" << endl
         << setw(45) << left << "     Fuer Codierung benoetigte Bitanzahl: "
         << setw(6)  << left << cntBits+8   /// 8 fuer IgnoreBits
         << " => Speicher: "
         << setw(6)  << left << cntBits+8 << " Bit" << endl
         << setw(45) << left << "     Verschiedene Zeichen: "
         << setw(6)  << left << cntUniqueChars
         << " => Speicher: "
         << setw(6)  << left << memSymFreqSize << " Bit" << endl
         << "     == Reduktion: " << bitRed-memSymFreqSize << " Bit"
         << ", Faktor: " << double(((bitRed-memSymFreqSize)*100)/memChars) << "%"
         << endl;

    cout << endl;
}
// }}}

// {{{ Huffman::decodiereDatei ()
/**
 * Datei Huffman-decodieren.
 * @param src   String mit dem Quelldatei-Namen
 * @param dest  String mit dem Zieldatei-Namen
 */
void Huffman::decodiereDatei (const string& src, const string& dest)
{
    clog << "Beginne Decodieren der Datei: " << src << endl;
```



```
    /// Aus der Quelldatei wird bitweise gelesen
    BitFileIn srcFile(src);

    /// Die decodierten Daten werden in eine Datei geschrieben
    ofstream destFileStream(dest.c_str(), ios::out);
    if (destFileStream.fail())
        throw FileNotWriteableException("Datei fuer decodierte Daten nicht schreibbar!");

    /// Haeufigkeiten einlesen und Baum aufbauen
    wurzelKnoten = 0;
    liesHaeufigkeiten(&srcFile);
    erstelleBaum();

    clog << "Beginne Durchsuchen des Baumes." << endl;

    int bit;
    const Knoten* p = wurzelKnoten;         /// Bei der Wurzel beginnen
    while (!srcFile.eof() && p != 0)
    {
        if (p->istAussenKnoten())           /// Blatt erreicht
        {
            destFileStream << p->getSymbol();
            p = wurzelKnoten;               /// Wieder zur Wurzel zurueck
        }

        try {
            bit = srcFile.getBit();         /// Naechstes Bit holen
        }
        catch (int& n) {                    /// Letzten n Bits ignorieren
            clog << "Ignoriere die letzten "
                 << n << " Bits!"
                 << endl;
            break;
        }

        switch (bit)                        /// 1 => rechts, 0 => links
        {
            case 1:
                p = p->getKindRechts(); break;
            case 0:
                p = p->getKindLinks();  break;
            default:
                throw Exception("Ungueltiges Zeichen bei der Decodierung!");
        }
    }

    destFileStream.close();
```

```
    clog << "Datei vollstaendig decodiert, Ergebnisdatei: " << dest
         << endl << endl;
}
// }}}

// {{{ Huffman::zaehleZeichen ()
/**
 * Alle Zeichen in der Quelldatei zaehlen.
 * @param src   Quelldatei-Strom
 */
void Huffman::zaehleZeichen (ifstream& src)
{
    /// Fuer jedes Zeichen dessen Haeufigkeit inkrementieren
    for (char c; src >> noskipws >> c;)
        zeichenHaeufigkeiten.at(c)++;

    src.clear();     /// Status-Bits loeschen und Position
    src.seekg(0);    /// zuruecksetzen fuer den zweiten Durchlauf
}
// }}}

// {{{ Huffman::erstelleBaum ()
/**
 * Aufbau des Huffman-Baums.
 */
void Huffman::erstelleBaum ()
{
    clog << "Beginne Erstellung d. Huffman-Baums." << endl;

    /// Vorrangwarteschlange erstellen
    VergleicheKnoten cmpFunc;
    priority_queue< Knoten*,
                    vector< Knoten* >,
                    VergleicheKnoten > pq(cmpFunc);

    /// Schritt 1: Alle Blattknoten erstellen und in
    /// Vorrangwarteschlange einfuegen
    vector< int >::const_iterator it;
    int i = 0;
    for (it  = zeichenHaeufigkeiten.begin();
         it != zeichenHaeufigkeiten.end();
         ++it)
    {
        if (*it != 0)                               /// Haeufigkeit != 0
        {
            clog << "Erstelle Blatt. "
                 << "Zeichen: " << char(i)
```



```
                 << ", Haeufigkeit: " << *it
                 << endl;

            /// Blatt erstellen und in Queue einfuegen
            Knoten* node = new AussenKnoten(i, *it);
            pq.push(node);

            cntUniqueChars++;                       /// Zeichen zaehlen
        }

        ++i;
    }

    /// Schritt 2: Jeweils zwei Knoten zusammenfassen,
    /// bis die Baum-Wurzel erreicht ist
    while (pq.size() > 1)
    {
        /// Linkes Kind holen und dann loeschen
        Knoten* kindLinks = pq.top(); pq.pop();

        /// Rechtes Kind holen und dann loeschen
        Knoten* kindRechts = pq.top(); pq.pop();

        clog << "Erstelle Vater. "
             << "Linkes Kind: " << kindLinks->getSymbol()
             << " / " << kindLinks->getHaeufigkeit()
             << ", Rechtes Kind: " << kindRechts->getSymbol()
             << " / " << kindRechts->getHaeufigkeit()
             << " -> Summe: "
             << kindLinks->getHaeufigkeit()+kindRechts->getHaeufigkeit()
             << endl;

        /// Elternknoten anlegen und in Queue einfuegen
        Knoten* elternKnoten = new InnenKnoten(kindLinks, kindRechts);
        pq.push(elternKnoten);
    }

    /// Wurzelknoten auf letzten verbleibenden Knoten setzen
    wurzelKnoten = pq.top();

    clog << "Wurzelknoten erreicht, Baum vollstaendig erstellt!"
         << endl << endl;
}
// }}}

// {{{ Huffman::erstelleCodeTabelle ()
/**
 * Code-Tabelle durch Traversieren des Baumes erstellen.
```

```
 */
void Huffman::erstelleCodeTabelle ()
{
    clog << "Beginne Traversieren des Baumes zur Ermittlung der Bitcodes."
         << endl;

    vector< int > ts;
    traversiereBaum(wurzelKnoten, ts);

    clog << "Code-Tabelle vollstaendig erstellt!"
         << endl << endl;
}
// }}}

// {{{ Huffman::schreibeHaeufigkeiten ()
/**
 * Haeufigkeitsverteilung in Datei schreiben.
 * @param destFile  Zieldatei
 */
void Huffman::schreibeHaeufigkeiten (BitFileOut* destFile)
{
    clog << "Beginne Schreiben der Haeufigkeitsverteilung." << endl;

    /// Status-Info: Anzahl der verschiedenen Zeichen schreiben
    string strNumChars = BitFileOut::decToBin(cntUniqueChars);
    strNumChars.insert(strNumChars.begin(), (8-strNumChars.length()), '0');
    destFile->writeBits(strNumChars);

    vector< int >::const_iterator it;
    int i = 0;
    for (it  = zeichenHaeufigkeiten.begin();
         it != zeichenHaeufigkeiten.end();
         ++it)
    {
        if (*it != 0)
        {
            /// Status-Info: Zeichenhaeufigkeits-Verteilung schreiben.
            /// Dazu wird jeweils 1 Byte fuer das Zeichen + 4 Byte fuer die
            /// dazugehoerige Haeufigkeit benoetigt

            /// Zeichen i, codiert in 1 Byte
            string strChar = BitFileOut::decToBin(i);
            strChar.insert(strChar.begin(), (8-strChar.length()), '0');
            destFile->writeBits(strChar);

            /// Haeufigkeit *it, codiert in 4 Byte
            string strFreq = BitFileOut::decToBin(*it);
            strFreq.insert(strFreq.begin(), (32-strFreq.length()), '0');
```



```
            destFile->writeBits(strFreq);
        }

        ++i;
    }

    clog << "Haeufigkeitsverteilung vollstaendig geschrieben!"
         << endl << endl;
}
// }}}

// {{{ Huffman::liesHaeufigkeiten ()
/**
 * Haeufigkeitsverteilung aus Datei lesen.
 * @param srcFile   Quelldatei
 */
void Huffman::liesHaeufigkeiten (BitFileIn* srcFile)
{
    clog << "Beginne Lesen der Haeufigkeitsverteilung." << endl;

    /// Status-Info lesen: Anzahl der verschiedenen Zeichen (1 Byte)
    int numUniqueChars = srcFile->readBits(8);

    /// Haeufigkeitsverteilung extrahieren
    char c;
    int freq;
    for (int i = 0; i < numUniqueChars; ++i)
    {
        /// Zeichen, codiert in 1 Byte
        c = srcFile->readBits(8);

        /// Haeufigkeit, codiert in 4 Byte
        freq = srcFile->readBits(32);

        /// In Tabelle einfuegen
        zeichenHaeufigkeiten.at(c) = freq;
    }

    clog << "Haeufigkeitsverteilung vollstaendig gelesen!" << endl;
}
// }}}

// {{{ Huffman::traversiereBaum ()
/**
 * Traversieren des Huffman-Baums.
 * @param p     Zeiger auf Startknoten
 * @param ts    Zwischenspeicher fuer Bitcode
 */
```

```
void Huffman::traversiereBaum (const Knoten* p, vector< int >& ts)
{
    if (p != 0)
    {
        if (p->istAussenKnoten())    /// Endknoten ist erreicht
        {
            vector< int >::const_iterator it;
            string stringCode;
            for (it = ts.begin(); it != ts.end(); ++it)
                stringCode.push_back((*it)+'0');   /// Code an String anfuegen

            /// Update der Statistik: Codelaenge * Haeufigkeit des akt. Zeichens
            cntBits += (stringCode.length()*p->getHaeufigkeit());

            ts.pop_back();      /// Letztes "Bit" entfernen

            pair< char, string > tmp(p->getSymbol(), stringCode);
            codeTabelle.insert(tmp);    /// Daten in Code-Tabelle einfuegen

            clog << "Blatt erreicht. Zeichen: " << p->getSymbol()
                 << ", Bitcode: " << stringCode << endl;
        }
        else                    /// Kein Endknoten, weiter traversieren
        {
            ts.push_back(0);    /// prefix: gehe links, 0 eintragen
            traversiereBaum( p->getKindLinks(), ts );

            ts.push_back(1);    /// infix; gehe rechts, 1 eintragen

            traversiereBaum( p->getKindRechts(), ts );
            ts.pop_back();      /// postfix; eine Ebene zurueckgehen
        }
    }
}
// }}}
```