МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
УО «Витебский государственный технологический университет»
Факультет повышения квалификации и переподготовки кадров

Кафедра информатики

Курсовая работа

«Проектирование и реализация приложения для компрессии и декомпрессии данных»

Студент гр. ИС-10 _____ А.В. Молчанов
подпись, дата

Руководитель
кандидат физ-мат наук, доцент _____ С.П.Кунцевич
подпись, дата

Работа защищена «__»_____20__г. с оценкой « _____».

Члены комиссии _____ _____
подпись Фамилия И.О

_____ _____
подпись Фамилия И.О

г. Витебск, 2011

Содержание

Обозначения и сокращения

BWT — Burrows-Wheeler transform, преобразование Барроуза-Уилера

RLE — Run-length encoding, кодирование длин серий

MTF — Move-To-Front, перемещение стопки книг — преобразование для кодирования данных

CRC — Cyclic redundancy check, циклический избыточный код

Введение

Компрессия (сжатие) данных — алгоритмическое преобразование данных, производимое с целью уменьшения их объёма. Применяется для более рационального использования устройств хранения и передачи данных. Обратная процедура называется восстановлением данных (декомпрессией).

Часто перед сжатием данные архивируют — объединяют несколько входных файлов в один выходной. При этом существует возможность их распаковки, сохранив неизменными их содержимое и названия (и, возможно, права доступа).

К настоящему времени создано множество приложений, использующих компрессию/декомпрессию данных:

— компрессоры

  bzip2

  WinRAR

  и др.

— форматы сжатия аудио/видео/изображений

  MP3

  MPEG-4

  JPEG (коды Хаффмана, арифметическое кодирование)

  и др.

— протоколы передачи данных

  TCP/IP (RFC 1144)

  PPP (Stac LZS, Predictor, Deflate, BSD, MPPC)

  и др.

— файловые системы

  SquashFS (gzip, LZMA)

  NTFS (LZ77)

и др.

Число программ, использующих компрессию постоянно растёт. Методы компрессии так же постоянно модифицируются и пополняются новыми.

Так недавно (25.03.2011) компания Google открыла код Snappy, библиотеки для сжатия данных [1].

Цель данной работы — проектирование и реализация приложения для компрессии/декомпрессии и предварительной архивации данных.

Задачи:

а) осуществить проектирование и реализацию приложения с соблюдением принципов объектно-ориентированного программирования

б) провести серию тестов созданного приложения

в) сделать вывод об эффективности работы приложения

г) наметить пути совершенствования разработанного приложения

1 Постановка задачи

Для реализации приложения необходимо решить следующие задачи:

а) реализовать алгоритмы (де-)кодирования BWT, MTF, RLE, алгоритм Хаффмана

б) создать класс, служащий для чтения, последующей модификации ((де-)компрессии) и записи данных

в) осуществить возможность идентификации считанных сжатых данных (алгоритм кодирования, исходный размер, CRC-сумма и т.д.)

г) организовать контроль целостности архива, корректности распаковки

д) возможность получения содержимого архива без предварительной распаковки

е) предусмотреть возможность извлечения информации из повреждённых архивов

ж) организовать поддержку различных локалей в названиях файлов

з) осуществить поддержку различных ОС (GNU/Linux, Windows)

и) создать графический интерфейс, позволяющий осуществлять компрессию/декомпрессию выбранных файлов, и отображать результаты выполнения операций

к) провести тестирование эффективности работы разработанного приложения и аналогичных программных продуктов

л) на основании сравнения результатов тестирования сделать вывод о практическом значении разработки

м) наметить пути совершенствования приложения

2 Теоретическая часть

2.1 Алгоритмы кодирования информации

В качестве используемых алгоритмов кодирования были выбраны: BWT, MTF, RLE и алгоритм Хаффмана, как широко используемые в различной комбинации во многих существующих компрессорах [2]. В частности в компрессоре bzip2 [3].

Необходимость использования связки алгоритмов обусловлена затруднениями при сжатии разнородной информации, с малой избыточностью данных.

2.1.1 BWT

BWT — это обратимый алгоритм перестановки символов во входном потоке, таким образом, что повторяющиеся подстроки образуют на выходе идущие подряд последовательности одинаковых символов.

Если после этого выполнить шаг по замене каждого символа расстоянием до его предыдущей встречи (т. н. алгоритм move to front, MTF) — то полученный набор чисел будет иметь крайне удачное статистическое распределение для применения энтропийного сжатия типа Хаффмана или же арифметического.

2.1.1.1 Иллюстрация работы

Вкратце, процедура преобразования происходит так:

а) выделяется блок из входного потока

б) формируется матрица всех перестановок, полученных в результате циклического сдвига блока

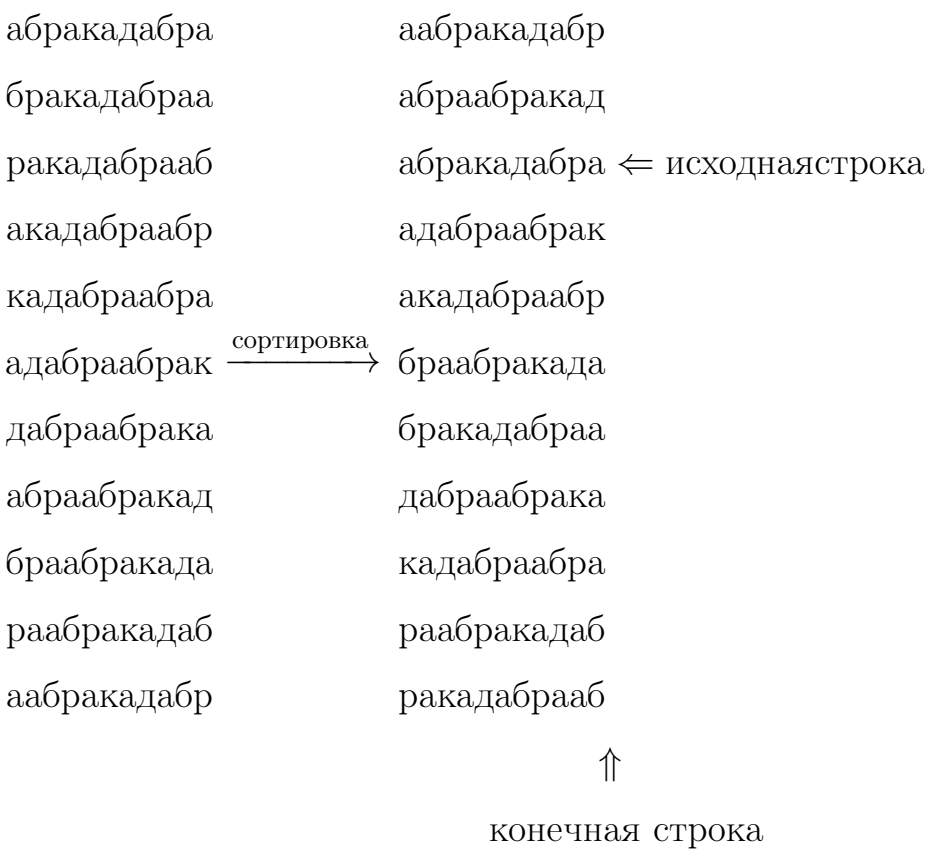
в) все перестановки сортируются в соответствии с лексикографическим порядком символов каждой перестановки

г) на выход подаётся последний столбец матрицы и номер строки, соответствующей оригинальному блоку

Пусть есть входной текст:

```
абракадабра
```

Выполним циклические сдвиги для получения упомянутых перестановок. Затем отсортируем полученные строки, предварительно пометив исходную.



Таким образом, в результате преобразования, взяв последний столбец, мы получили строку «рдакраааабб» и номер исходной строки в отсортированной матрице (первичный индекс) — 2 (считая с 0). Можно заметить, поскольку использовался сдвиг циклический, символы последнего столбца предшествуют продолжениям, по которым матрица сортировалась.

Сравним исходную и конечную строки текста:

```
абракадабра
рдакраааабб
```

Видно, что полученная строка содержит большее число повторяющихся символов. Это будет способствовать увеличению степени сжатия.

Однако отличительная особенность BWT не в том, что оно создаёт более легко кодируемые выходные данные — многие тривиальные операции позволяют это сделать, а в том, что оно *обратимо*, позволяя восстановить исходный документ из данных последнего столбца.

2.1.2 MTF

MTF — преобразование для кодирования данных, разработанное для улучшения производительности энтропийного кодирования. При хорошей реализации, оно достаточно быстро для включения как дополнительный шаг в алгоритмах сжатия данных.

Основной идеей преобразования является замена каждого входного символа его номером в специальном стеке недавно использованных символов. Последовательности идентичных символов, к примеру, будут заменены оригинальным алгоритмом (начиная со второго символа) на последовательность нулей. Если же символ долго не появлялся во входной последовательности, он будет заменен большим числом. Преобразование заменяет последовательность входных символов на последовательность целых чисел, если во входных данных было много локальных корреляций, то среди этих чисел будут преобладать небольшие, лучше сжимаемые энтропийным кодированием, чем исходные данные.

2.1.2.1 Иллюстрация работы

Пусть на выходе BWT-преобразования есть текст, длиной 20 символов:

```
bbbbbccccddddddaaaaa
```

Попробуем сжать эту последовательность при помощи, например, метода Хаффмана.

Вероятности всех четырех символов в данном примере равны 1/4. Легко посчитать, что в результате кодирования мы получим последовательность длиной 20*2 = 40 бит.

Теперь проделаем тоже самое со строкой, подвергнутой MTF-преобразованию (предположим, начальный список выглядит как 'a', 'b', 'c', 'd').

```
bbbbbccccddddddaaaaa   исходная строка
10000200003000030000   строка после MTF
```

Теперь проделаем тоже самое со строкой, подвергнутой MTF-преобразованию (предположим, начальный список выглядит как 'a', 'b', 'c', 'd').

Символ	Частота	Вероятность	Код Хаффмана
0	16	4/5	0
1	2	1/10	10
2	1	1/20	110
3	1	1/20	111

В результате сжатия получаем последовательность длиной 16*1 + 2*2 + 3*2 = 26 бит.

В итоге выигрыш составил 14 бит. Степень сжатия 65%.

2.1.3 RLE

Это один из наиболее старых методов сжатия. Суть этого метода заключается в замене идущих подряд одинаковых символов числом, характеризующим их количество.

RLE широко применяется для сжатия деловую графику — изображения с большими областями повторяющегося цвета. Этот алгоритм с различными модификациями используются в форматах графики PCX, TIFF, TGA.

Однако главное назначение кодирования длин повторов в связке с BWT — увеличить скорость сжатия и разжатия.

RLE можно применить дважды — до преобразования и после. До преобразования данный метод может пригодиться, если мы имеем дело с потоком, содержащим много повторов одинаковых символов. Сортировка строк матрицы перестановок — наиболее длительная из процедур, необходимых для сжатия при помощи BWT. В случае высокоизбыточных данных время выполнения этой процедуры может существенно (в разы) возрастать. Сейчас разработаны методы сортировки, устойчивые к такого рода избыточности данных, но ранее метод кодирования длин повторов широко использовался на этом этапе ценой небольшого ухудшения сжатия. RLE следует применять, если указанных повторов уж слишком много.

2.1.3.1 Иллюстрация работы

Пусть есть входная информация (шестнадцатеричная форма) из 20 байт:

```
05 05 05 05 05 05 01 01 03 03 03 03 03 03 05 03 FF FF FF FF
```

Заменим последовательность входных байт на последовательность <количество повторов, значение>:

```
06 05 02 01 06 03 01 05 01 03 04 FF
```

В результате сжатия длина последовательности 12 байт. Степень сжатия 60%.

2.1.4 Алгоритм Хаффмана

Код Хаффмана — адаптивный жадный алгоритм оптимального префиксного кодирования алфавита с минимальной избыточностью.

В основе кода Хаффмана лежит идея: вместо того чтобы кодировать все символы одинаковым числом бит (как это сделано, например, в ASCII кодировке, где на каждый символ отводится ровно по 8 бит), будем кодировать символы, которые встречаются чаще, меньшим числом бит, чем те, которые встречаются реже. Более того, потребуем, чтобы код был оптимален или, другими словами, минимально-избыточен.

Первым такой алгоритм опубликовал Дэвид Хаффман (David Huffman) в 1952 году. Алгоритм Хаффмана двухпроходный:

а) построение частотного словаря и генерация кодов

б) кодирование

2.1.4.1 Иллюстрация работы

Пусть есть входной текст:

```
abracadabra
```

При кодировке его двоичным кодом постоянной длины, вида:

a	b	r	c	d
000	001	010	011	100

,

мы получаем сообщение длиной 33 бит:

```
000001010000011000100000001010000
```

Построим таблицу частот символов:

a	b	r	c	d
5	2	2	1	1

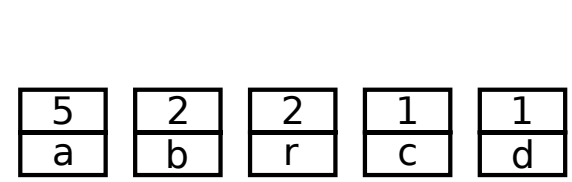
Создадим узлы из символов, входящих в текст с весом равным частоте их появления (рисунок 2.1(а)).

Выбираем два символа с наименьшим весом — **c** и **d**. Объединим их в новый узел, вес которого равен сумме весов входящих в него символов (рисунок 2.1(б)). Левую ветвь считаем за 0, правую 1.

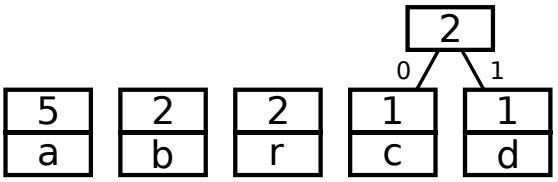
На следующем шаге объединяем узлы **b** и **r** (рисунок 2.1(в)).

Далее объединяем узлы **b**/**r** и **c**/**d** (рисунок 2.1(г)).

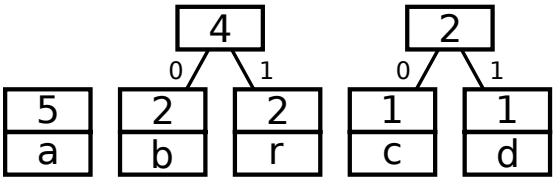
На последнем шаге объединяем узлы **a** и **(b/r)/(c/d)** (рисунок 2.1(д)).



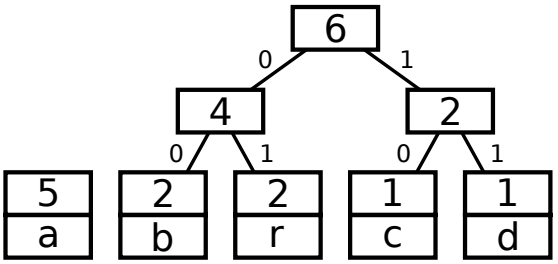
(а) Создание узлов



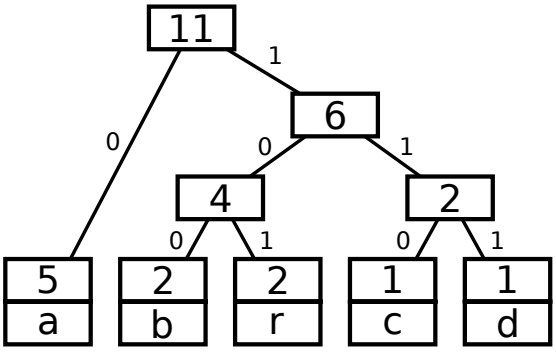
(б) Первый шаг



(в) Второй шаг



(г) Третий шаг



(д) Последний шаг

Рисунок 2.1 — Построение дерева кодирования Хаффмана

Чтобы определить код для каждого из символов, входящих в исходный текст, мы должны пройти путь от листа дерева, соответствующего данному символу до корня дерева, накапливая биты при перемещении по

ветвям дерева. Полученная последовательность битов является кодом данного символа, записанного в обратном порядке.

Для данной таблицы символов коды Хаффмана будут выглядеть следующим образом:

a	b	r	c	d
0	100	101	110	111

Используя полученные коды, закодируем исходный текст:

```
01001010110011101001010
```

В результате сжатия длина последовательности 23 бит. Степень сжатия 70%.

Стоит отметить, что за 50 лет со дня опубликования, код Хаффмана ничуть не потерял своей актуальности и значимости. Сжатие данных по Хаффману применяется при сжатии фото- и видеоизображений (JPEG, стандарты сжатия MPEG), в архиваторах (BZIP2, PKZIP, LZH и др.), в протоколах передачи данных MNP5 и MNP7.

2.2 Используемые библиотеки

В приложении используются различные функции и классы, как входящие в состав стандартной библиотеки, так и реализованные в сторонних библиотеках.

В исходные тексты shcodec и libdivsufsort был внесён ряд изменений, такие как: замена макроподстановок на шаблоны функций, переход на работу с динамической памятью в стиле C++, использованы алгоритмы из STL.

2.2.1 Стандартная библиотека

2.2.1.1 Функции

memset — заполняет участок памяти указанным символом

```
#include <string.h>
void *memset(void *s, int c, size_t n);
```

Функция memset() заполняет первые **cnt** байтов буфера **dest** символом **c**.

memcpy — копирование участка памяти

```
#include <string.h>
void *memcpy(void *dest, const void *src, size_t n);
```

Функция memcpy() копирует **n** байт из участка памяти **src** в участок памяти **dest**. Адреса участков памяти не должны перекрываться друг другом.

strncpy — копирование строки

```
#include <string.h>
char *strncpy(char *dest, const char *src, size_t n);
```

Копирует первые **n** литер строки **src** в строку **dest**.

Если **src** длиннее **dest**, результат исполнения не будет завершаться '\0'.

Если в **src** меньше **n** литер, строка **dest** будет дополнена до **n** литерами '\0'.

Функция возвращает **dest**.

2.2.1.2 Алгоритмы

swap – заменяет один объект другим

```
#include <algorithm>
void swap( Assignable& a, Assignable& b );
```

Функция swap() меняет местами значения переменных a и b. swap() ожидает, что аргументы будут иметь присваиваемый тип, т.е. они должны иметь конструктор копирования и работать с оператором =. Эта функция производит одну копию и два присваивания.

search – поиск элементов одного диапазона в другом

```
#include <algorithm>
forward_iterator search( forward_iterator start1, forward_iterator
    end1, forward_iterator2 start2, forward_iterator2 end2 );
```

Алгоритм search() ищет элементы диапазона [**start2**,**end2**) в диапазоне [**start1**,**end1**).

Если search() находит совпадающий поддиапазон элементов, он возвращает итератор на начало этого диапазона. Если совпадений не найдено, возвращается итератор на **end1**.

lower_bound — разновидность бинарного поиска

```
#include <algorithm>
forward_iterator lower_bound( forward_iterator first,
   forward_iterator last, const T& val );
```

Алгоритм lower_bound() — разновидность бинарного поиска. Эта функция ищет первое место в упорядоченном диапазоне, определённом итераторами **first** и **last**, куда можно вставить значение **val**, сохраняя упорядоченность. Эквивалентная функция возвращает итератор на первый элемент, не являющийся меньшим, чем **val**, либо возвращает итератор **end**, если все элементы диапазона меньше чем **val**.

Функция работает только с упорядоченными диапазонами.

Возвращаемое значение функции — итератор, указывающий на место, куда можно безопасно вставить значение **val**. Для упорядочивания используется оператор <.

for_each — применяет функцию ко всем объектам

```
#include <algorithm>
UnaryFunction for_each( input_iterator start, input_iterator end,
   UnaryFunction f );
```

Алгоритм for_each() применяет функцию **f** к каждому элементу между **start** и **end**. Возвращаемое значение for_each - функция **f**.

2.2.1.3 Контейнеры

Все контейнеры C++ могут быть сравнены и присвоены с помощью стандартных операторов: ==, !=, <=, >=, <, > и =.

Общие методы для всех используемых в приложении контейнеров приведены в таблице 2.1

Таблица 2.1 — Общие методы контейнеров STL

Метод	Описание
Ёмкость	
bool empty()**const**	Проверяет, пуст ли контейнер
size_type size()**const**	Возвращает фактическое количество элементов
Функции получения итераторов	
iterator begin()	Возвращает итератор произвольного доступа для первого элемента
iterator end()	Возвращает итератор произвольного доступа для позиции за последним элементом
Вставка и удаление элементов	
void clear()	Удаляет все элементы

Вектор (std::vector)

Вектор содержит смежные элементы, хранимые в массиве.

Используемые методы приведены в таблице 2.2

Таблица 2.2 — Методы контейнера **vector::list**

Метод	Описание
Конструкторы	
vector()	Создаёт пустой вектор
vector(size_type num, **const** T& val = T())	Создаёт вектор с **num** объектами. Если **val** объявлена, то каждый из этих объектов будет инициализирован ее значением; в противном случае объекты получат значение конструктора по умолчанию
vector(input_iterator start, input_iterator end)	Создаёт вектор, состоящий из элементов между **start** и **end**.
Ёмкость	
void reserve(size_type size)	Увеличивает ёмкость вектора, если текущая ёмкость меньше заданной
void resize(size_type count, T value = T())	Приводит вектор к размеру **count**, если заданный размер больше текущего, новые элементы инициализируются значением **value**, либо значением по умолчанию
Доступ к элементам	
T& [size_type pos]	Возвращает элемент с индексом **pos** (без интервальной проверки)
T& back()	Возвращает последний элемент (без проверки существования)
T* data()	Возвращает указатель на диапазон данных [data(), data() + size())
T& front()	Возвращает первый элемент (без проверки существования)
Присваивание	
void assign(iterator first, iterator last)	Присваивает элементы интервала [**first**; **last**)

Продолжение на след. стр.

Продолжение таблицы 2.2

Вставка и удаление элементов	
void pop_back()	Удаляет последний элемент
void push_back(**const** T & val)	Присоединяет копию элемента **val** в конец вектора

Список (std::list)

Список — вид контейнера STL, отличающийся высокой скоростью вставки и удаления элементов в любой позиции. А также быстрым последовательным доступом к элементам.

Используемые методы приведены в таблице 2.3

Таблица 2.3 — Методы контейнера **std::list**

Метод	Описание
Конструкторы	
list()	Создаёт пустой список
Доступ к элементам	
T& back()	Возвращает последний элемент (без проверки существования)
T& front()	Возвращает первый элемент (без проверки существования)
Вставка и удаление элементов	
void pop_back()	Удаляет последний элемент
void push_back(**const** T & val)	Присоединяет копию элемента **val** в конец списка

Отображения (std::map)

Отображение — сортированный ассоциативный контейнер который содержит уникальные пары ключ/значение.

Используемые методы приведены в таблице 2.4

Таблица 2.4 — Методы контейнера **std::map**

Метод	Описание
Конструкторы	
map()	Создаёт пустое отображение
Вставка и удаление элементов	
size_type erase(**const** key_type& key)	Удаляет все элементы со значением **key** и возвращает количество удалённых элементов
pair<iterator,**bool**> insert(**const** T& pair)	Вставляет копию **pair** и возвращает позицию нового элемента **iterator**, также возвращается признак успешного выполнения операции **bool**
Операции над элементами	
iterator find(**const** key_type& key)	Возвращает позицию первого элемента с ключом **key**(или **end()**)

2.2.2 Boost

Boost [4] — собрание библиотек, расширяющих C++. Свободно распространяются по лицензии Boost Software License вместе с исходным кодом.

2.2.2.1 Классы

Динамический битовый массив (boost::dynamic_bitset)

boost::dynamic_bitset — класс, представляющий собой набор битов.

Используемые методы приведены в таблице 2.5

Таблица 2.5 — Методы класса **boost::dynamic_bitset**

Метод	Описание
Конструкторы	
dynamic_bitset()	Создаёт пустой массив
Ёмкость	
void resize(size_type num_bits, **bool** value = **false**)	Приводит массив к размеру **num_bits**, если заданный размер больше текущего, новые элементы инициализируются значением **value**, либо значением **false**
Доступ к элементам	
reference [size_type pos]	Возвращает ссылку на бит с индексом **pos** (без интервальной проверки)
Операции над элементами	
bool any()**const**	Возвращает **true**, если есть хотя бы один ненулевой бит
bool none()**const**	Возвращает **true**, если все биты нулевые

2.2.3 Qt

Qt [5] — кросс-платформенный инструментарий разработки ПО на языке программирования C++. Есть также «привязки» ко многим другим языкам программирования: Python, Ruby, Java, PHP, и другие.

2.2.3.1 Классы

QByteArray

Класс **QByteArray** предоставляет массив байт.

Используемые методы приведены в таблице 2.6

Таблица 2.6 — Методы класса **QByteArray**

Метод	Описание
Функции	
const char * QByteArray:: constData ()**const**	Возвращает указатель на данные хранящиеся массиве

QCheckBox

Класс **QCheckBox** представляет собой флажок с текстовой подписью.

Используемые методы приведены в таблице 2.7

QComboBox

Класс **QComboBox** представляет собой поле ввода с выпадающим списком

Используемые методы приведены в таблице 2.8

QDialog

Класс **QDialog** является базовым для диалоговых окон.

Используемые методы приведены в таблице 2.9

QDir

Класс **QDir** предоставляет доступ к структуре каталогов и их содержимому.

Используемые методы приведены в таблице 2.10

QFile

Класс **QFile** предоставляет интерфейс для чтения и записи файлов.

Используемые методы приведены в таблице 2.11

QFileDialog

Класс **QFileDialog** предоставляет диалог, позволяющий пользователю выбрать файлы или каталоги.

Используемые методы приведены в таблице 2.12

QLabel

Класс **QLabel** служит для отображения текста или изображений.

Используемые методы приведены в таблице 2.13

QMainWindow

Класс **QMainWindow** предоставляет главное окно приложения.

Используемые методы приведены в таблице 2.14

QProgressDialog

QProgressDialog предоставляет диалог, обеспечивающий обратную связь в ходе выполнения медленной операции.

Используемые методы приведены в таблице 2.15

QPushButton

Класс **QPushButton** представляет собой кнопку.

Используемые методы приведены в таблице 2.16

QSpinBox

Класс **QSpinBox** представляет собой поле ввода со счётчиком.

Используемые методы приведены в таблице 2.17

QString

Класс **QString** представляет строку в кодировке Unicode.

Используемые методы приведены в таблице 2.18

QStringList

Класс **QStringList** представляет собой список строк.

Используемые методы приведены в таблице 2.19

QTableWidget

Класс **QTableWidget** представляет собой таблицу с моделью данных по-умолчанию.

Используемые методы приведены в таблице 2.20

QThread

Класс **QThread** обеспечивает платформо-независимую реализацию потоков.

Используемые методы приведены в таблице 2.21

Таблица 2.7 — Методы класса **QCheckBox**

Метод	Описание
Конструкторы	
QCheckBox (**const** QString & text, QWidget ∗ parent = 0)	Создаёт флажок с подписью **text** и указанным родительским виджетом **parent**
Функции	
bool isChecked ()**const**	Возвращает **true** если флаг поднят, иначе **false**
Сигналы	
void stateChanged (**int** state)	Испускается при изменении состояния флажка. Новое состояние передаётся в **state**

Таблица 2.8 — Методы класса **QComboBox**

Метод	Описание
Конструкторы	
QComboBox (QWidget * parent = 0)	Создаёт объект с указанным родительским виджетом **parent**
Функции	
void setCurrentIndex (**int** index)	Возвращает индекс выбранного элемента
void insertItems (**int** index, **const** QStringList & list)	Вставляет элементы из списка **list** в позицию **index**
void setCurrentIndex (**int** index)	Устанавливает индекс выбранного элемента
Сигналы	
void currentIndexChanged (**int** index)	Испускается при изменении индекса выбранного элемента. Новое значение передаётся в **index**

Таблица 2.9 — Методы класса **QDialog**

Метод	Описание
Конструкторы	
QDialog (QWidget ∗ parent = 0, Qt::WindowFlags flags = 0)	Создаёт окно диалога с указанным родительским виджетом **parent** и параметрами окна **flags**
Функции	
void adjustSize ()	Подгоняет размер окна под его содержимое
int height ()**const**	Возвращает высоту окна без рамки
void setFixedSize (**const** QSize & s)	Устанавливает фиксированный размер окна
void setLayout (QLayout ∗ layout)	Устанавливает менеджер компоновки
void setWindowTitle (**const** QString &)	Задаёт заголовок окна диалога
int width ()**const**	Возвращает ширину окна без рамки
Слоты	
void show ()	Отображает диалог

Таблица 2.10 — Методы класса **Qdir**

Метод	Описание
Функции	
QString relativeFilePath (**const** QString & fileName) **const**	Возвращает путь к fileName относительно каталога

Таблица 2.11 — Методы класса **QFile**

Метод	Описание
Конструкторы	
QFile (**const** QString & name)	Создаёт новый объект, соответствующий файлу с именем **name**
Функции	
void QFile::close ()	Закрывает файл и устанавливает его режим доступа в **NotOpen**. Также сбрасывается последняя ошибка доступа
bool QFile::open (OpenMode mode)	Открывает файл в режиме доступа **mode**. Возвращает **true** в случае успеха; при неудаче возвращается **false**
qint64 QFile::pos ()**const**	Возвращает текущую позицию файлового курсора
qint64 QIODevice::read (**char** ∗ data, qint64 maxSize)	Читает не более **maxSize** байт из файла в буфер **data**. Возвращает число прочитанных байт
bool QFile::remove ()	Удаляет файл. Возвращает **true** в случае успеха; при неудаче возвращается **false**.
bool QFile::seek (qint64 off)	Устанавливает текущюю позицию чтения в **off**. Возвращает **false** при возникновении ошибки, иначе **true**
qint64 QFile::size ()**const**	Возвращает размер файла

Таблица 2.12 — Методы класса **QFileDialog**

Метод	Описание
Конструкторы	
QFileDialog (QWidget ∗ parent = 0, **const** QString & caption = QString(), **const** QString & directory = QString(), **const** QString & filter = QString())	Создаёт диалог с указанным родительским виджетом **parent** и заголовком **caption**, который отображает содержимое указанного каталога **directory**. Перед отображением содержимое отфильтровано используя список фильтров в **filter**
Функции	
QDir directory ()**const**	Возвращает каталог, отображённый в диалоге
void setFileMode (FileMode mode)	Устанавливает режим отображения элементов
void setWindowTitle (**const** QString &)	Задаёт заголовок окна диалога
QStringList selectedFiles()**const**	Возвращает список абсолютных путей к выбранным элементам
int exec ()	Отображает диалог в качестве модального. Возвращает **DialogCode**

Таблица 2.13 — Методы класса **QLabel**

Метод	Описание
Конструкторы	
QLabel (**const** QString & text , QWidget ∗ parent = 0, Qt:: WindowFlags f = 0)	Создаёт объект, отображающий текст **text** с указанным родительским виджетом **parent** и параметрами окна **f**
Функции	
void setText (**const** QString &)	Устанавливает отображаемый текст

Таблица 2.14 — Методы класса **QMainWindow**

Метод	Описание
Конструкторы	
QMainWindow (QWidget * parent = 0, Qt::WindowFlags flags = 0)	Создаёт окно с указанным родительским виджетом **parent** и параметрами окна диалога **flags**
Функции	
void addDockWidget (Qt ::DockWidgetArea area, QDockWidget * dockwidget)	Добавляет заданный прикрепляемый виджет (**dockwidget**) в определённую область **area**
QToolBar * addToolBar (**const** QString & title)	Создаёт объект **QToolBar**, устанавливая его заголовок в **title**, и вставляет его в верхнюю область панелей инструментов
QMenuBar * menuBar ()**const**	Возвращает панель меню для главного окна. Создаёт и возвращает пустую панель меню, если панель не существует
void setCentralWidget (QWidget * widget)	Устанавливает заданный виджет (**widget**), в качестве центрального виджета
void setupUi (QWidget * widget)	Устанавливает пользовательский интерфейс указанным виджетом
void setWindowTitle (**const** QString &)	Задаёт заголовок окна
Слоты	
void show ()	Отображает окно

Таблица 2.15 — Методы класса **QProgressDialog**

Метод	Описание
Конструкторы	
QProgressDialog (**const** QString & labelText, **const** QString & cancelButtonText, **int** minimum, **int** maximum , QWidget * parent = 0, Qt:: WindowFlags f = 0)	Создаёт диалог с указанным родительским виджетом **parent**; текстом, описывающим текущую операцию **labelText**; подписью кнопки отмены **cancelButtonText**; **minimum** и **maximum** — количество шагов, отображаемой операции; параметрами окна диалога **f**
Функции	
void setAutoClose (**bool** close)	Устанавливает автоматическое скрытие окна далога при получении сигнала слотом reset()
void setWindowModality (Qt::WindowModality windowModality)	Устанавливает модальность окна диалога
Слоты	
void reject ()	Скрывает окно диалога и устанавливает код результата в **Rejected**
void reset ()	Сбрасывает диалог
void show ()	Отображает диалог
Сигналы	
void canceled ()	Испускается, если была нажата кнопка отмены

Таблица 2.16 — Методы класса **QPushButton**

Метод	Описание
Конструкторы	
QPushButton (**const** QString & text, QWidget ∗ parent = 0, Qt::WindowFlags f = 0)	Создаёт кнопку, с надписью **text** с указанным родительским виджетом **parent** и параметрами окна **f**
Сигналы	
void clicked ()	Испускается при нажатии на кнопку

Таблица 2.17 — Методы класса **QSpinBox**

Метод	Описание
Конструкторы	
QSpinBox (QWidget ∗ parent = 0)	Создаёт объект с указанным родительским виджетом **parent**
Функции	
void setRange (**int** minimum, **int** maximum)	Устанавливает диапазон допустимых значений [**minimum** ; **maximum**]
void setSingleStep (**int** val)	Устанавливает инкремент/декремент значения счётчика при нажатии на стрелки
void setSuffix (**const** QString & suffix)	Устанавливает суффикс при отображении значения счётчика
int value ()**const**	Возвращает значение счётчика
Сигналы	
void valueChanged (**int** i)	Испускается при изменении значения счётчика. Новое значение передаётся в **i**

Таблица 2.18 — Методы класса **QString**

Метод	Описание
Конструкторы	
QString ()	Создаёт пустую строку
Функции	
QString QString::fromUtf8 (**const char** ∗ str, **int** size = −1)	Возвращает **QString**, инициализированный первыми **size** байтами строки **str**. Если **size** равен -1 (по умолчанию), он приравнивается к qstrlen(str)
QString QString::number (qulonglong n, **int** base = 10)	Возвращает строку, эквивалентную числу **n** в системе исчисления с основанием **base**. По умолчанию **base** равен 10
QString QString::number (**double** n, **char** format = ' g', **int** precision = 6)	Возвращает строку, эквивалентную числу **n**, форматированную в соответствии с указанным форматом **format** и точностью **precision**
QByteArray toUtf8 ()**const**	Возвращает UTF-8 представление строки как **QByteArray**

Таблица 2.19 — Методы класса **QStringList**

Метод	Описание
Функции	
int QList::count ()**const**	Возвращает число строк в списке
QString & QList::**operator** [] (**int** i)	Возвращает ссылку на строку с индексом **i**

Таблица 2.20 — Методы класса **QTableWidget**

Метод	Описание
Конструкторы	
QTableWidget (QObject * parent = 0)	Создаёт новую таблицу с указанным родительским виджетом
Функции	
void clear ()	Удаляет содержимое всех элементов
QTableWidgetItem * item (**int** row, **int** column) **const**	Возвращает элемент для строки **row** и столбца **column**. Если элемент не задан, возвращает 0
void setColumnCount (**int** columns)	Устанавливает число столбцов таблицы
void setHorizontalHeaderLabels (**const** QStringList & labels)	Устанавливает горизонтальные заголовки таблицы
void setItem (**int** row, **int** column, QTableWidgetItem * item)	Устанавливает элемент **item** в позицию (**row**, **column**)
void setRowCount (**int** rows)	Устанавливает число строк таблицы
Слоты	
void start (Priority priority = InheritPriority)	Запускает исполнение потока с заданным приоритетом
Сигналы	
void finished ()	Испускается при завершении выполнения потока
void started ()	Испускается в начале выполнения потока

Таблица 2.21 — Методы класса **QThread**

Метод	Описание
Конструкторы	
QThread (QObject * parent = 0)	Создаёт новый поток с указанным родительским
Слоты	
void start (Priority priority = InheritPriority)	Запускает исполнение потока с заданным приоритетом
Сигналы	
void finished ()	Испускается при завершении выполнения потока
void started ()	Испускается в начале выполнения потока

2.2.4 libdivsufsort

libdivsufsort [6] предоставляет библиотеку C, позволяющую строить суффиксные массивы и выполнять BWT.

2.2.4.1 Функции

divbwt — Прямое BWT преобразование

```
#include "divsuf/divsufsort.h"
int32_t divbwt(const uint8_t *T, uint8_t *U, int32_t *A, int32_t n
    );
```

Функция divbwt() выполняет преобразование входной строки **T** длиной **n**, используя буфер **A**. Полученная строка возвращается в **U**.

Функция возвращает первичный индекс.

inverse_bw_transform — обратное BWT преобразование

```
#include "divsuf/divsufsort.h"
inverse_bw_transform(const uint8_t *T, uint8_t *U, int32_t *A,
    int32_t n, int32_t idx);
```

Функция inverse_bw_transform() выполняет обратное преобразование входной строки **T** длиной **n** с первичным индексом **idx**, используя буфер **A**. Полученная строка возвращается в **U**.

Функция возвращает **0** в случае успешного преобразования, иначе **-1** или **-2**.

2.2.5 shcodec

shcodec [7] — каноничный статический Хаффман кодек.

2.2.5.1 Функции

sh_EncodeBlock — кодирование Хаффмана

```
#include "shclib.h"
int sh_EncodeBlock( uchar *iBlock, uchar *oBlock, unsigned int
    bSize );
```

Функция sh_EncodeBlock() выполняет преобразование входного блока **iBlock** длиной **bSize**, Полученный блок возвращается в **oBlock**. Функция возвращает размер полученного блока или **0** в случае ошибки.

sh_DecodeBlock — декодирование Хаффмана

```
#include "shclib.h"
int sh_DecodeBlock( uchar *iBlock, uchar *oBlock, int bSize );
```

Функция sh_DecodeBlock() выполняет обратное преобразование входного блока **iBlock** длиной **bSize**, Полученный блок возвращается в **oBlock**. Функция возвращает размер полученного блока или **0** в случае ошибки.

3 Проектирование приложения

Приложение состоит из множества модулей. Ниже приведён их список, включающий в себя предназначение содержащихся в них классов.

а) представление данных (раздел 4.1, страница 48)

DataBlock — блок данных

DataBlockHeader — заголовок блока

ReaderDataBlockHeader — чтение заголовка блока

Crc — проверка целостности (CRC32)

FilesTable — информация об обрабатываемых файлах

FileBlocksInfo — карта блоков файла

б) (де-)кодирование данных (раздел 4.2, страница 62)

CodecAbstract — абстрактный класс, родительский для всех кодеков

Codec — класс, наследующий кодеки всех методов

CodecBWT — кодек BWT

CodecRLE — кодек RLE

CodecMTF — кодек MTF

CodecHUFF — кодек Хаффмана

в) конечный программный интерфейс архиватора/компрессора (раздел 4.3, страница 68)

Compressor — архивация/распаковка файлов, проверка целостности и т.п.

г) потоки (раздел 4.4, страница 75)

CompressorThread — операции с компрессором (архивация/распаковка файлов)

д) настройка параметров (де-)компрессии (раздел 4.5, страница 80)

CompressSetingsPanel — настройка компрессии

DecompressSetingsPanel — настройка декомпрессии

е) отображение результатов (раздел 4.6, страница 84)

FileList — список файлов и результаты их обработки (целостность, сжаты, ожидают сжатия и т.д.)

StatInfoW — статистическая информация о результатах выполнения (средняя скорость, степень сжатия и т.д.)

ж) графический пользовательский интерфейс (раздел 4.7, страница 89)

Face — главное окно приложения

з) прочее (раздел 4.8, страница 93)

DataUnitsToQString — представление объёма, скорости передачи данных в виде строки QString с соответствующим суффиксом (MB, KB/s,..)

и) главный модуль (раздел 4.9, страница 94)

Main — создание объекта главного класса — **Face**

4 Реализация приложения

4.1 Представление данных

4.1.1 DataBlock

Входные данные представляются в виде блоков. В последующим их можно модифицировать ((де-)кодировать) и записывать в файл(-ы). Также выполняется контроль целостности.

Структура блока приведена на рисунке 4.1.

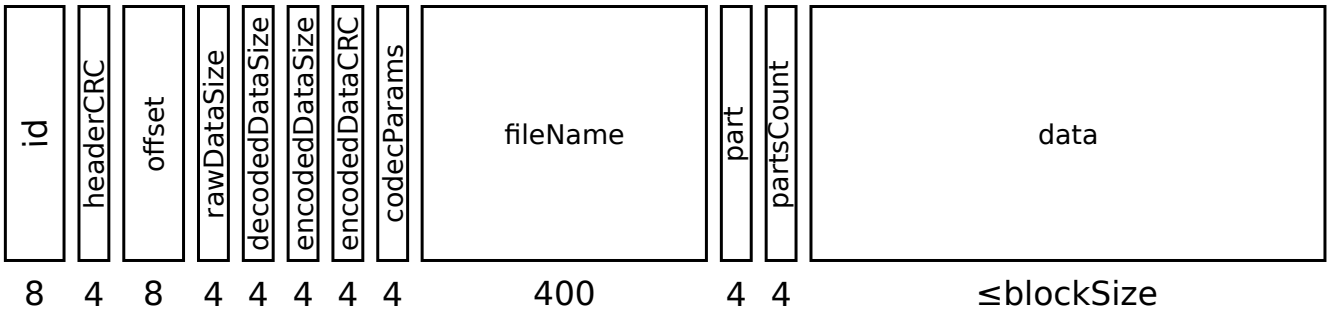


Рисунок 4.1 — Структура блока (число — объём части в байтах)

Пояснение назначения частей блока:

id — идентификатор архива. Включает подпись приложения и метод кодирования (рисунок 4.2).

headerCRC — CRC-32 сумма заголовка

offset — смещение начала блока от начала файла

rawDataSize — размер сырых данных

decodedDataSize — объём декодированных данных

encodedDataSize — объём закодированных данных

encodedDataCRC — CRC32 сумма закодированных данных

codecParams — параметры кодировщика

fileName — имя файла

part — № блока

partsCount — число блоков, составляющих исходный файл

Рисунок 4.2 — Идентификаторы архива

DataBlock — класс, служащий для выполнения вышеперечисленных операций над данными.

Атрибуты класса приведены в таблице 4.1

Таблица 4.1 — Атрибуты класса **DataBlock**

Атрибут	Описание
dataT *data	Данные
uint32_t dataSize	Объём данных
uint32_t nBytesToRead	Максимальный объём исходных данных
DataBlockHeader *header	Заголовок блока
dataT *outBlock	Представление блока в виде вектора байт
bool recoveryMode	Флаг режима чтения сжатых блоков: **true** — включён поиск заголовка блока в архиве, **false** — последовательное чтение блоков

Методы приведены в таблице 4.2

Таблица 4.2 — Методы класса **DataBlock**

Метод	Описание
Конструкторы	
DataBlock( )	Конструктор по-умолчанию
DataBlock(uint32_t block_size)	Создаёт объект, максимальный объём исходных данных в котором **block_size**

Продолжение на след. стр.

Продолжение таблицы 4.2

DataBlock(**unsigned char** ∗ inData)	Создаёт объект из представления в виде массива байт **inData**
Функции	
DataBlockHeader ∗ readRAW(QFile &in)	Чтение сырых данных из файла **in**. Возвращает указатель на заголовок блока
int read(QFile &in)	Чтение сжатых данных из файла **in**. В случае успешного чтения возвращает **0**, если файл повреждён возвращает **FILE_BROKEN**, елси повреждён заголовок **HEADER_CORRUPTED**, если размер файла меньше размера заголовка **FILE_TOO_SMALL**, если был достигнут конец файла **FILE_END**
int writeRAW( )	Запись сырых данных в соответствующие выходные файлы (с именами аналогичными до сжатия). В случае успешной записи возвращает **0**, иначе **OUTPUT_ERROR**
void write(QFile &out)	Запись сжатых данных в файл **out**
DataBlockHeader ∗ getHeader( )	Возвращает заголовок блока
unsigned char ∗ getData( )	Возвращает данные
void setData(**unsigned char**∗ inData, uint32_t inDataSize)	Устанавливает данные в **inData** объёмом **inDataSize**
void setData(dataT∗ inData)	Устанавливает данные в **inData**
dataT ∗getBlock( )	Возвращает представление блока в виде вектора байт
void setBlock(**unsigned char**∗ inBlock)	Устанавливает блок из представления в виде массива байт
uint32_t calcCRC( )	Вычисляет CRC сумму

Продолжение на след. стр.

Продолжение таблицы 4.2

int checkCRC( )	Проверка целостности блока. Возвращает **0** если проверка выполнена успешно, иначе **-1**
void recordCRC( )	Записывает CRC сумму в заголовок

4.1.2 DataBlockHeader

Класс **DataBlockHeader** служит для работы с заголовком блока.

В состав класса входит подкласс **HeaderDataType**.

Атрибуты **DataBlockHeader::HeaderDataType** приведены в таблице 4.3

Атрибуты класса **DataBlockHeader** приведены в таблице 4.4

Методы приведены в таблице 4.5

Таблица 4.5 — Методы класса **DataBlockHeader**

Метод	Описание
Конструкторы	
DataBlockHeader (**unsigned char** *in_header_data)	Создаёт заголовок из данных **in_header_data**
Функции	
void setCodecParams(uint32_t codec_params)	Записывает параметры кодека
uint32_t getCodecParams( )	Возвращает параметры кодека
void setRAWDataSize(uint32_t RAWDataSize)	Устанавливает объём сырых данных
uint32_t getRAWDataSize( ) **const**	Возвращает объём сырых данных
void setEncodedDataSize(uint32_t encodedDataSize)	Устанавливает объём закодированных данных
uint32_t getEncodedDataSize( ) **const**	Возвращает объём закодированных данных

Продолжение на след. стр.

Продолжение таблицы 4.5

void setDecodedDataSize(uint32_t decodedDataSize)	Устанавливает объём декодированных данных
uint32_t getDecodedDataSize( ) **const**	Возвращает объём декодированных данных
void setId(uint64_t id)	Устанавливает идентификатор кодека
uint64_t getId( )**const**	Возвращает идентификатор кодека
void setData(**unsigned char** ∗ inHeaderData)	Устанавливает заголовок из представления в виде массива
unsigned char ∗ getData( )	Возвращает представление заголовка в в виде массива
void setDataCRC(uint32_t crc)	Устанавливает CRC сумму данных
uint32_t getDataCRC( )	Возвращает CRC сумму данных
uint32_t calcCRC( )	Возвращает CRC сумму заголовка
void recordCRC( )	Записывает CRC сумму
int checkCRC( )	Проверка целостности. Возвращает **0** если проверка выполнена успешно, иначе **-1**
void setFileName (**const char** ∗ fileName)	Устанавливает имя файла
char ∗ getFileName ( )	Возвращает имя файла
void setPart(**const** uint32_t part)	Устанавливает № блока
uint32_t getPart( )**const**	Возвращает № блока
void setPartsCount(**const** uint32_t part)	Устанавливает число блоков
uint32_t getPartsCount( )**const**	Возвращает число блоков
void setHeaderCRC(uint32_t headerCRC)	Устанавливает CRC сумму заголовка

Продолжение на след. стр.

Продолжение таблицы 4.5

uint32_t getHeaderCRC( ) **const**	Возвращает CRC сумму заголовка
void setOffset(uint64_t offset)	Устанавливает смещение начала блока относительно начала входного файла
uint64_t getOffset( )	Возвращает смещение начала блока относительно начала входного файла
void clean( )	Очистка заголовка
void initRAW(uint64_t offset, **const** uint32_t rawDataSize)	Устанавливает смещение начала блока в **offset**, размер сырых данных **rawDataSize**

4.1.3 ReaderDataBlockHeader

Класс **ReaderDataBlockHeader** предназначен для чтения из входного файла заголовка блока.

Методы класса приведены в таблице 4.6

4.1.4 FilesTable

Класс **FilesTable** служит для хранения и получения информации об обрабатываемых файлах.

Атрибуты класса **FilesTable** приведены в таблице 4.7

Методы класса приведены в таблице 4.8

4.1.5 FileBlocksInfo

Класс **FileBlocksInfo** содержит карту блоков файла.

Атрибуты класса **FileBlocksInfo** приведены в таблице 4.9

Методы класса приведены в таблице 4.10

Таблица 4.3 — Атрибуты класса **DataBlockHeader::HeaderDataType**

Атрибут	Описание
uint64_t id	Идентификатор блока
uint32_t headerCRC	CRC сумма заголовка
uint64_t offset	Смещение начала блока относительно начала входного файла
uint32_t rawDataSize	Размер сырых данных
uint32_t decodedDataSize	Размер данных до кодирования
uint32_t encodedDataSize	Размер данных после кодирования
uint32_t encodedDataCRC	CRC сумма закодированных данных
uint32_t codecParams	Параметры кодировщика
char fileName[MAX_FILENAME_LENGTH]	Имя исходного файла
uint32_t part	Порядковый № блока
uint32_t partsCount	Число блоков, на которые был разбит исходный файл

4.1.6 crc

Модуль crc содержит функцию **crc32()**. Она выполняет вычисление CRC-32 суммы.

uint_least32_t crc32(const unsigned char * buf, size_t len)

```
#include "crc.h"
uint_least32_t crc32(const unsigned char * buf, size_t len);
```

Возвращает CRC32 сумму для массива **buf**, размером **len**.

Таблица 4.4 — Атрибуты класса **DataBlockHeader**

Атрибут	Описание
dataT *data	Данные
uint32_t dataSize	Объём данных
uint32_t nBytesToRead	Максимальный объём исходных данных
DataBlockHeader *header	Заголовок блока
dataT *outBlock	Представление блока в виде вектора байт
bool recoveryMode	Флаг режима чтения сжатых блоков: **true** — включён поиск заголовка блока в архиве, **false** — последовательное чтение блоков
HeaderDataType headerData	Структура с данными заголовка

Таблица 4.6 — Методы класса **ReaderDataBlockHeader**

Метод	Описание
Конструкторы	
ReaderDataBlockHeader( )	Создаёт объект
Функции	
int read(DataBlockHeader * outHeader, QFile &in, **bool** searchHeader = **false**)	Читает из файла **in** заголовок блока в **outHeader**, если флаг **searchHeader** установлен — выполняет поиск заголовка, иначе производит последовательное чтение. В случае неповреждённого заголовка возвращает **0**, если файл повреждён возвращает **FILE_BROKEN**, если размер файла меньше размера заголовка возвращает **FILE_TOO_SMALL**, если был достигнут конец файла **FILE_END**
bool find (DataBlockHeader * outHeader, QFile &fin)	Ищет заголовок в файле **fin**. В случае нахождения неповреждённого заголовка возвращает **false**, иначе **true**

Таблица 4.7 — Атрибуты класса **FilesTable**

Атрибут	Описание
map< string, FileBlocksInfo > fileBlocksTable	Сведения о файлах, хранящиеся в виде <имя файла, сведения>
map< string, vector <uint32_t> > brokenFiletable	Сведения о повреждённых файлах, хранящиеся в виде <имя файла, карта блоков>
vector <string> brokenFilesNames	Список имён повреждённых файлов
vector<FileInfo> archiveContent	Содержание архива

Таблица 4.8 — Методы класса **FilesTable**

Метод	Описание
Конструкторы	
FilesTable( )	Создаёт объект
Функции	
int add(DataBlockHeader * inHeader, **unsigned int** id = 0)	Добавляет блок с заголовком **inHeader** и присваивает соответствующему файлу идентификатор **id**
void remove(DataBlockHeader * inHeader)	Удаляет сведения о файле, которому соответствует заголовок **inHeader**
map< string, vector <uint32_t> > * getNonCompleteFilesBlocksInfo( )	Возвращает сведения о повреждённых файлах
vector<string> * getNonCompleteFilesNames( )	Возвращает список имён повреждённых файлов
vector<FileInfo> * getArchiveContent( )	Возвращает содержание архива
void clean( )	Очистка
unsigned int getId(DataBlockHeader * inHeader)**const**	Возвращает идентификатор файла, которому соответствует заголовок **inHeader**
unsigned int getId(string fileName)**const**	Возвращает идентификатор файла, с именем **fileName**

Таблица 4.9 — Атрибуты класса **FileBlocksInfo**

Атрибут	Описание
uint32_t blocksCount	Число блоков, на которые был разбит исходный файл
boost::dynamic_bitset<> blocks	Карта полученных блоков
vector<uint32_t> nonRecievedblocks	Список недополученных блоков
unsigned int id	Идентификатор файла

Таблица 4.10 — Методы класса **FileBlocksInfo**

Метод	Описание
Конструкторы	
FileBlocksInfo(**const** uint32_t _totalBlocks, **unsigned int** _id = 0)	Создаёт карту блоков с числом блоков **_totalBlocks** и идентификатор файла **_id**
Функции	
uint32_t getTotalBlocks( )	Возвращает число блоков
int setRecievedBlock(uint32_t blockN)	Устанавливает отметку о получении блока №**blockN**. В случае успеха возвращает **0**, если № блока > общего числа блоков **BLOCK_OUT_OF_RANGE**, если блок с данным номером был получен ранее **BLOCK_ALREADY_RECIEVED**, если были получены все блоки **ALL_BLOCKS_RECIEVED**, если был получен первый и последний блок файла **FIRST_AND_LAST_RECIEVED_BLOCK**
vector<uint32_t> * getNonRecievedBlocksInfo( )	Возвращает список недополученных блоков
bool complete( )	Возвращает **true** если были получены все блоки, иначе **false**
unsigned int getId( )**const**	Возвращает идентификатор файла

4.2 (Де-)кодирование данных

Для (де-)компрессии данных используется связка из классов, реализующих (де-)кодирование методами BWT, RLW, MTF и Хаффмана.

Родительским для всех классов кодеков является виртуальный класс **Codec_abstract**.

Дочерними для него являются классы кодеки, позволяющие как кодировать, так и декодировать данным методом. Например класс **CodecBWT**.

Наконец дочерним классом для всех классов кодеков, позволяющим (де-)кодировать любым из вышеперечисленных методом, является класс **Codec**.

Граф наследования для класса **Codec** приведена на рисунке 4.3

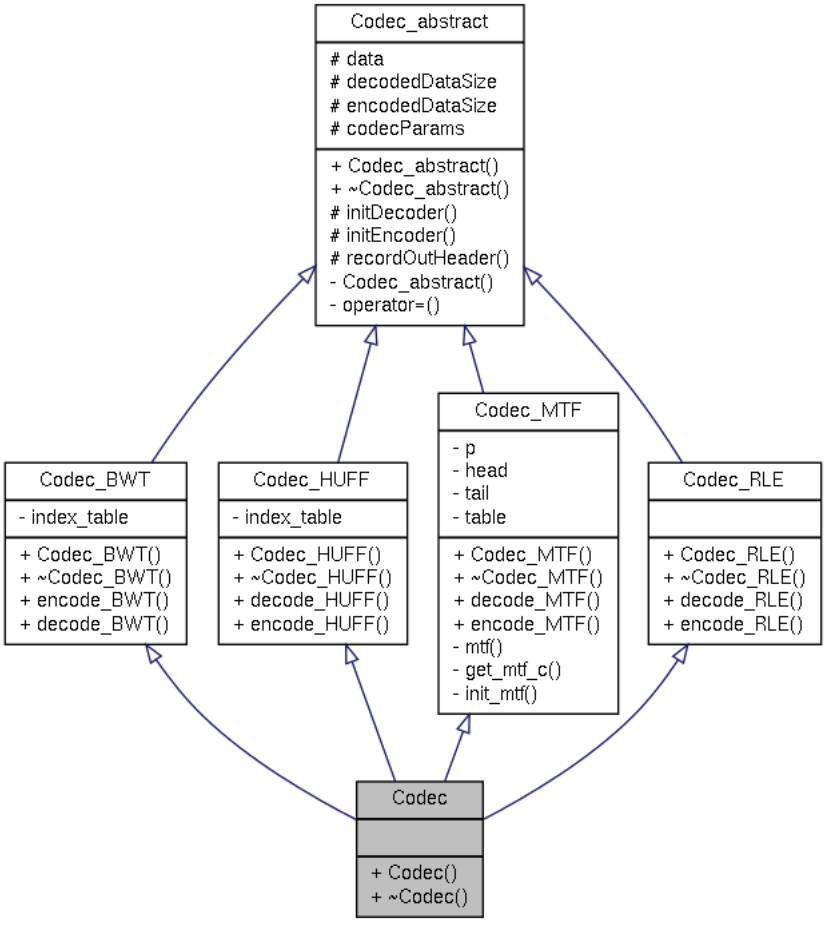


Рисунок 4.3 — Граф наследования класса Codec

4.2.1 Codec_abstract

Класс Codec_abstract содержит атрибуты и методы, использующиеся в каждом классе (де-)кодировщиков.

Атрибуты класса Codec_abstract приведены в таблице 4.11

Таблица 4.11 — Атрибуты класса **Codec_abstract**

Атрибут	Описание
∗data	Данные
unsigned int decodedDataSize	Размер данных до кодировани
unsigned int encodedDataSize	Размер данных после кодирования
uint32_t codecParams	Параметры кодека

Методы класса приведены в таблице 4.12

Таблица 4.12 — Методы класса **Codec_abstract**

Метод	Описание
Конструкторы	
Codec_abstract( )	Создаёт объект
Функции	
void initDecoder(DataBlock ∗ in_block)	Инициализирует декодирование блока **in_block**
void initEncoder(DataBlock ∗ in_block)	Инициализирует кодирование блока **in_block**
void recordOutHeader(DataBlockHeader ∗outHeader, **const** uint64_t id)	Записывает выходные сведения в заголовок **outHeader**, идентификатор кодека **id**

4.2.2 Codec_MTF

Класс **Codec_MTF** представляет собой MTF кодек.

В состав класса входит подкласс **mtf_list_t**.

Атрибуты **Codec_MTF::mtf_list_t** приведены в таблице 4.13

Таблица 4.13 — Атрибуты класса **Codec_MTF::mtf_list_t**

Атрибут	Описание
int c	Код символа
struct mtf_list *prev	Указатель на предыдущий элемент списка
struct mtf_list *next	Указатель на следующий элемент списка

Атрибуты класса **Codec_MTF** приведены в таблице 4.14

Таблица 4.14 — Атрибуты класса **Codec_MTF**

Атрибут	Описание
mtf_list_t *p, *head, *tail, *table	Части списка MTF

Методы класса приведены в таблице 4.15

4.2.3 Codec_RLE

Класс **Codec_RLE** представляет собой RLE кодек.

Атрибутов класс не имеет.

Методы класса приведены в таблице 4.16

4.2.4 Codec_BWT

Класс **Codec_BWT** представляет собой BWT кодек.

Атрибутов класс не имеет.

Методы класса приведены в таблице 4.17

Таблица 4.15 — Методы класса **Codec_MTF**

Метод	Описание
Конструкторы	
Codec_MTF( )	Создаёт объект
Функции	
void decode_MTF(DataBlock* inData)	Выполняет кодирование
void encode_MTF(DataBlock* inData)	Выполняет декодирование
int mtf(**int** c)	Возвращает MTF код символа **c**
int get_mtf_c(**int** i)	Возвращает символ, MTF код которого **c**
void init_mtf(**int** tsize)	Инициализирует список MTF с размером таблицы **tsize**

Таблица 4.16 — Методы класса **Codec_RLE**

Метод	Описание
Конструкторы	
Codec_RLE( )	Создаёт объект
Функции	
void encode_RLE(DataBlock* inData)	Выполняет кодирование
void decode_RLE(DataBlock* inData)	Выполняет декодирование

4.2.5 Codec_HUFF

Класс **Codec_HUFF** представляет собой Хаффман кодек.
Атрибутов класс не имеет.
Методы класса приведены в таблице 4.18

Таблица 4.17 — Методы класса **Codec_BWT**

Метод	Описание
Конструкторы	
Codec_BWT( )	Создаёт объект
Функции	
void encode_BWT(DataBlock* inData)	Выполняет кодирование
void decode_BWT(DataBlock* inData)	Выполняет декодирование

Таблица 4.18 — Методы класса **Codec_HUFF**

Метод	Описание
Конструкторы	
Codec_HUFF( )	Создаёт объект
Функции	
void encode_HUFF(DataBlock * inData)	Выполняет кодирование
void decode_HUFF(DataBlock * inData)	Выполняет декодирование

4.3 Конечный программный интерфейс архиватора/компрессора

4.3.1 CompressorStatus

Класс **CompressorStatus** служит для вывода диагностических сведений:

— имя и идентификатор файла, код ошибки

— процент завершения текущей операции

— скорость обработки данных

Для этих целей служат виртуальные функции:

— **virtual void** CompressorStatus::showProgress(**float** progress, **const** QString &fileName, **float** speed)

— **virtual void** CompressorStatus::showInfo(ErrorCode errorCode, **const** QString &fileName, **unsigned int** id)

В случае необходимости в них передаются вышеперечисленные сведения. В дальнейшем эти функции будут переопределены в классе CompressorThreadCompressorStatus.

Так же **CompressorStatus** позволяет управлять/следить за статусом выполнения операции (запуск/запущена, прерывание/остановлена)

Для этих целей служат функции:

— RunStatus getRunStatus( )

— **void** setRunStatus(RunStatus status)

В приведеных функциях используются перечисления (листинг 4.1).

Смысловая нагрузка этих значений:

а) RunStatus

RUN — операция выполняется

Листинг 4.1 — Перечисления в **CompressorStatus**

```
enum RunStatus
{
    RUN, STOP
};
enum ErrorCode
{
    SUCCESS, INPUT_FILE_OPEN_ERROR, OUTPUT_FILE_WRITE_ERROR,
        INPUT_FILE_CORRUPTED, INPUT_FILE_UNCORRUPTED,
        DECOMPRESS_FAIL, PROCEED, CANCELLED
};
```

STOP — операция остановлена

б) ErrorCode

SUCCESS — выполнено успешно

INPUT_FILE_OPEN_ERROR — ошибка чтения входного файла

OUTPUT_FILE_WRITE_ERROR — ошибка в записи выходного файла

INPUT_FILE_CORRUPTED — входной файл повреждён

INPUT_FILE_UNCORRUPTED — входной файл не повреждён

DECOMPRESS_FAIL — ошибка декомпрессии (нарушено функционирование декодера)

PROCEED — файл обрабатывается

CANCELLED — файл пропущен

4.3.2 Encoder

Функциональный класс Encoder служит для выполнения требуемого метода кодирования над блоком.

Он необходим для выполнения заданной последовательности кодировщиков над блоком, которое будет в дальнейшем использовано в классе Compressor.

Атрибуты класса **Encoder** приведены в таблице 4.19

Таблица 4.19 — Атрибуты класса **Encoder**

Атрибут	Описание
DataBlock *block	Обрабатываемый блок
Codec *codec	Кодек

Методы класса приведены в таблице 4.20

Таблица 4.20 — Методы класса **Encoder**

Метод	Описание
Конструкторы	
Encoder(DataBlock * block, Codec *codec)	Создаёт кодировщик для блока **block** и кодека **codec**
Функции	
void operator( )(Compressor:: CoderTypes coderType)	Выполняет кодирование блока методом **coderType**

4.3.3 Compressor

Класс **Compressor** является конечным классом архиватора/компрессора.

Он реализует всю функциональность, связанную с архивацией/компрессией/декомпрессией/распаковкой входных файлов.

В состав класса входит подкласс **Stat**, служащий для передачи статистической информации о результатах выполнения.

Атрибуты класса **Compressor::Stat** приведены в таблице 4.21

Таблица 4.21 — Атрибуты класса **Compressor::Stat**

Атрибут	Описание
off_t decodedSize	Объём декодированных данных
off_t encodedSize	Объём закодированных данных
float speed	Средняя скорость обработки данных (байт/с)
vector <string> brokenFilesNames	Имена повреждённых файлов

Так же в классе **Compressor** объявлен тип перечисления **CoderTypes** (листинг 4.2)

Смысловая нагрузка этих значений:

NONE — кодирование не применяется

RLE — RLE кодирование

BWT — BWT кодирование

MTF — MTF кодирование

HUFFMAN — Хаффмана кодирование

Атрибуты класса **Compressor** приведены в таблице 4.22

Методы класса приведены в таблице 4.23

Листинг 4.2 — Перечисление **Compressor::CoderTypes**

```
enum CoderTypes
{
    NONE, RLE, BWT, MTF, HUFFMAN
} ;
```

Таблица 4.22 — Атрибуты класса Compressor

Атрибут	Описание
Codec codec	Кодек
DataBlock *block	Обрабатываемый блок
CompressorStatus *status	Статус выполнения
off_t currReadBytesCount	Количество прочитанных байт в настоящее время
off_t decodedDataSize	Объём декодированных данных
off_t encodedDataSize	Объём закодированных данных
uint32_t totalProcessingTime	Суммарное время выполнения
unsigned int source_data_size	Объём сырых данных
FilesTable blocksTable	Сведения об обрабатываемых файлах
CompressorStatus:: ErrorCode error	Код ошибки
vector<FilesTable::FileInfo > contents	Содержимое архива
list< CoderTypes > defaultCompressSequence	Последовательность методов кодирования по-умолчанию

Таблица 4.23 — Методы класса **Compressor**

Метод	Описание
Конструкторы	
Compressor(CompressorStatus *status)	Создаёт компрессор, для контроля выполнения операций будет использоваться **status**
Функции	
CompressorStatus::ErrorCode compress(**const** QStringList &iFileNames, **unsigned int** iFilesCount, **const** QString &oFileName, **unsigned int** blocksize, **const** list< CoderTypes > * compressSequence = NULL)	Сжимает входные файлы с именами **iFileNames**, в количестве **iFilesCount**и записывает архив в файл с именем **oFileName**. Размер блока **blocksize**, последовательность кодировщиков **compressSequence**. Возвращает код ошибки
CompressorStatus::ErrorCode decompress(**const** QString & iFileName, **bool** keepBroken = **false**))	Распаковывает файл **iFileName**, если флаг **keepBroken** поднят — удаляет извлечённые повреждённые файлы. Возвращает код ошибки
void getStat(Stat *stat)	Возвращает статистическую информацию в **stat**
CompressorStatus::ErrorCode listArchiveContents(**const** QString &iFileName, vector< FilesTable::FileInfo> * **const** contents = NULL)	Возвращает содержимое архива **iFileName** в **contents** (так же по мере чтения содержимого архива сведения о файлах будут передаваться через **status**). Возвращает код ошибки
void showEncodingProgress(**const** QString &currFileName, **float** speed)	Отображает сведения о степени завершёности операции архивации

Продолжение на след. стр.

Продолжение таблицы 4.23

void showDecodingProgress(**const char** * currFileName, **float** speed)	Отображает сведения о степени завершёности операции распаковки (чтения содержимого архива)
void showInfo(CompressorStatus::ErrorCode errorCode, **const** QString & currFileName = "", **unsigned int** id = 0)	Отображает код ошибки **errorCode** выполнения операции над файлом **currFileName** с идентификатором **id**
void showInfo(CompressorStatus::ErrorCode errorCode, **const char** * currFileName, **unsigned int** id = 0)	Перегруженная функция
void compress(**const** list< CoderTypes > * compressSequence = NULL)	Выполняет сжатие блока последовательностью кодировщиков **compressSequence**
bool decompress( )	Выполняет распаковку блока (тип кодировщиков распознаётся автоматически по заголовку)
bool createEmptyFile(**const char** * fileName)	Создаёт пустой файл **fileName**. При успешном завершении возвращает **true**, иначе **false**
void removeBrokenFiles( )	Удаляет повреждённые файлы
float speed(**unsigned int** nBytes, clock_t elapsedTime)	Возвращает скорость обработки данных (байт/с), если за **elapsedTime** тактов процессора было обработано **nBytes** байт

4.4 Потоки

Для увеличения отзывчивости графического интерфейса приложения, операции архивации, распаковки и получения содержимого архива запускаются в отдельном потоке.

Этот поток реализован в классе **CompressorThread**.

4.4.1 CTCompressorStatus

Класс **CTCompressorStatus** является дочерним для класса **CompressorStatus**. В нём переопределены функции **showProgress()** и **showInfo()**.

Так же была добавлена функция **void** setCompressorThread(CompressorThread *comprThread) устанавливающая объект comprThread, в котором будут вызваны соответсвующие функции (листинг 4.3).

4.4.2 CompressorThread

Класс **CompressorThread** предоставляет поток операции архивации, распаковки и получения содержимого архива.

В классе объявлен тип перечисления **Mode**, определяющие возможные режимы работы (листинг 4.4).

Смысловая нагрузка этих значений:

NONE — кодирование не применяется

COMPRESSING — режим компрессора

DECOMPRESSING — режим декомпрессора

LIST — режим чтения содержимого архива

Атрибуты класса **CompressorThread** приведены в таблице 4.24

Листинг 4.3 — Реализация методов **showProgress()** и **showInfo()** в **CTCompressorStatus**

```
void
CTCompressorStatus::showProgress( float progress, const QString &
    fileName, float speed )
{
    comprThread->showProgress( progress, fileName, speed );
}

void
CTCompressorStatus::showInfo( ErrorCode errorCode, const QString &
    fileName, unsigned int id )
{
    comprThread->showInfo( errorCode, fileName, id );
}
```

Листинг 4.4 — Перечисление **CompressorThread::Mode**

```
enum Mode
{
    NONE, COMPRESSING, DECOMPRESSING, LIST
};
```

Таблица 4.24 — Атрибуты класса **CompressorThread**

Атрибут	Описание
QStringList iFileNames	Имена входных файлов
QString iFileName	Имя входного файла
QString destFileName	Имя выходного файла
unsigned int blocksize	Размер блока
QDir compressBaseDir	Базовый каталог со сжимаемыми файлами
bool keepBroken	Флаг, показывающий оставлять ли повреждённые файлы
QList< Compressor::CoderTypes> compressSequence	Последовательность кодеков
CTCompressorStatus *status	Объект для вывода диагностических сведений
Mode runMode	Режим работы

Методы класса приведены в таблице 4.25

Таблица 4.25 — Методы класса **CompressorThread**

Метод	Описание
Конструкторы	
CompressorThread( )	Создаёт поток
Функции	
void run( )	Запускает выполнение потока
void showProgress(**float** progress, **const** QString &fileName, **float** speed)	Отображает сведения о степени завершённости операции **progress**, выполняемой над файлом **fileName** с текущей скоростью **speed**

Продолжение на след. стр.

Продолжение таблицы 4.25

void showInfo(CompressorStatus:: ErrorCode errorCode, **const** QString &fileName, **unsigned int** id)	Отображает код ошибки **errorCode** выполнения операции над файлом **fileName** с идентификатором **id**
void initCompress(**const** QStringList &iFileNames, **const** QString & destFileName, **unsigned int** blocksize , **const** QDir &compressBaseDir, **const** QList< Compressor::CoderTypes> & compressSequence)	Инициализирует сжатие файлов **iFileNames** с базовым каталогом **compressBaseDir** и записывает архив в файл с именем **destFileName**. Размер блока **blocksize**, последовательность кодировщиков **compressSequence**
void initDecompress(**const** QString & iFileName, **bool** keepBroken = **false**)	Распаковывает файл **iFileName**, если флаг **keepBroken** поднят — удаляет извлечённые повреждённые файлы
void initList(**const** QString &iFileName)	Инициализирует вывод содержимого архива **iFileName**
void compress( )	Запускает сжатие
void decompress( )	Запускает распаковку
void list( )	Запускает вывод содержимого архива
Слоты	
void stop( )	Останавливает выполнение текущей операции
Сигналы	

Продолжение на след. стр.

Продолжение таблицы 4.25

void progressChanged(**int** progress, QString fileName, **float** speed)	Испускается при выполнении следующего шага текущей операции. Возвращает процент завершённости операции в **progress**, имя обрабатываемого файла в **fileName**, текущую скорость обработки данных в **speed**
void info (CTCompressorStatus:: ErrorCode error, QString fileName, **unsigned int** id)	Испускается при поступлении информации из **status**. Возвращает код ошибки в **error**, имя обрабатываемого файла в **fileName**, идентификатор в **id**
void statInfo(Compressor::Stat stat)	Испускается при завершении текущей операции. Выводит статистическую информацию в **stat**

4.5 Настройка параметров (де-)компрессии

Для настройки параметров (де-)компрессии созданы классы **DecompressSettingsPanel** и **CompressSettingsPanel** соответсвенно.

Оба класса представляют собой виджеты с элементами корректировки параметров выполнения сжатия/распаковки.

В дальнейшем они используются как прикрепляемые виджеты (dockWidgets) в главном окне приложения.

4.5.1 CompressSettingsPanel

Класс CompressSettingsPanel предоставляет возможность пользователю настроить параметры компрессии файлов:

— последовательность применяемых методов кодирования

— размер блока

— сбросить параметры к исходным

Атрибуты класса **CompressSettingsPanel** приведены в таблице 4.26

Методы класса приведены в таблице 4.27

4.5.2 DecompressSettingsPanel

Класс DecompressSettingsPanel предоставляет возможность пользователю настроить параметр декомпрессии файлов: оставлять ли распакованные повреждённые файлы.

Атрибуты класса **DecompressSettingsPanel** приведены в таблице 4.28

Методы класса приведены в таблице 4.29

Таблица 4.26 — Атрибуты класса **CompressSettingsPanel**

Атрибут	Описание
QSpinBox∗ blockSizeSpinBox	Поле ввода со счётчиком для указания размера блока
hline QComboBox ∗ encoder1ComboBox, ∗ encoder2ComboBox, ∗ encoder3ComboBox, ∗ encoder4ComboBox , ∗ encoder5ComboBox, ∗ encoder6ComboBox, ∗ encoder7ComboBox, ∗ encoder8ComboBox	Поля ввода с выпадающим списком для выбора кодировщика на каждом этапе из 8 возможных

Таблица 4.27 — Методы класса **CompressSettingsPanel**

Метод	Описание
Конструкторы	
CompressSettingsPanel( )	Создаёт виджет
Функции	
void setupWidgetsConnections( )	Устаналивает связи сигнал-слот с виджетами
Слоты	
void set(**unsigned int** blockSize, QList< Compressor:: CoderTypes> compressSequence)	Устанавливает настройки: размер блока **blockSize**, последовательность методов кодирования **compressSequence**
void get( )	Принимает сигналы от дочерних виджетов при изменении их значений (изменении настроек)
void disableEncoders( )	Отключает кодировщики на всех этапах
Сигналы	
void settingsChanged(**unsigned int** blockSize, QList < Compressor::CoderTypes> compressSequence)	Испускается при изменении настроек. Возвращает размер блока в **blockSize**, последовательность кодировщиков в **compressSequence**
void resetToDefaults(**void**)	Испускается при сбросе параметров к исходным

Таблица 4.28 — Атрибуты класса **DecompressSettingsPanel**

Атрибут	Описание
QCheckBox * keepBrokenFilesCheckbox	Флажок для указания оставлять ли распакованные повреждённые файлы

Таблица 4.29 — Методы класса **DecompressSettingsPanel**

Метод	Описание
Конструкторы	
DecompressSettingsPanel( )	Создаёт виджет
Функции	
void setupWidgetsConnections( )	Устанавливает связи сигнал-слот с виджетами
Слоты	
void set(**bool** keepBroken)	Устанавливает настройки: если keepBroken=**true** распакованные повреждённые файлы удаляются
void get( )	Принимает сигналы от дочерних виджетов при изменении их значений (изменении настроек)
Сигналы	
void settingsChanged(**bool** keepBroken)	Испускается при изменении настроек
void resetToDefaults(**void**)	Испускается при сбросе параметров к исходным

4.6 Отображение результатов

Для отображения результатов выполнения операций служат классы **FileList** и **StatInfoW**.

4.6.1 FileList

Класс **FileList** предоставляет виджет таблицу, отображающую список обрабатываемых файлов и результат их обработки.

Например список файлов для упаковки и результат (обрабатывается, ожидает, возникла ошибка, пропущен).

Атрибуты отсутствуют.

Методы класса приведены в таблице 4.30

Таблица 4.30 — Методы класса **FileList**

Метод	Описание
Конструкторы	
FileList( )	Создаёт виджет
Функции	
void setFileList(**const** QStringList & fileList, QDir & basedir)	Устаналивает список файлов **fileList** с базовым каталогом **basedir**
Слоты	
void init( )	Инициализирует таблицу
void setProceedFileStatus(**const** QString &fileName, **unsigned int** id)	Устанавливает для файла с идентификатором **id** и именем **fileName** статуса **обрабатывается**
void setFailFileStatus(**const** QString &fileName, **unsigned int** id)	Устанавливает для файла с идентификатором **id** и именем **fileName** статуса **ошибка**
void setSuccessFileStatus(**const** QString &fileName, **unsigned int** id)	Устанавливает для файла с идентификатором **id** и именем **fileName** статуса **успешно обработан**

Продолжение на след. стр.

Продолжение таблицы 4.30

void setCancelledFilesStatus (**const** QString &fileName, **unsigned int** id)	Устанавливает для файла с идентификатором **id** и именем **fileName** статуса **отменён**
void setCorruptedFileStatus (**const** QString &fileName, **unsigned int** id)	Устанавливает для файла с идентификатором **id** и именем **fileName** статуса **повреждён**
void setUnCorruptedFileStatus (**const** QString &fileName, **unsigned int** id)	Устанавливает для файла с идентификатором **id** и именем **fileName** статуса **не повреждён**
void showInfo(CTCompressorStatus::ErrorCode error, QString fileName, **unsigned int** id)	Отображает код ошибки **error**для файла с идентификатором **id** и именем **fileName**
void initTable(**int** rowCount, **int** columnCount)	Инициализирует таблицу и устанавливает количество столбцов **columnCount**, количество строк **rowCount**
QTableWidgetItem∗ getItemById(**int** id)	Возвращает указатель на элемент, соответствующий файлу с идентификатором **id**

4.6.2 StatInfoW

Класс StatInfoW предоставляет диалоговое окно со статистической информации о результатах выполнения

— последовательность кодировщиков

— размер блока

— объём входных файлов

— объём после выполнения операции

— коэффициент сжатия

— средняя скорость обработки данных

В класс **StatInfoW** входит подкласс **StatInfo**, контейнер со статистической информацией.

Атрибуты класса **StatInfoW::StatInfo** приведены в таблице **??**

Таблица 4.31 — Атрибуты класса StatInfoW::StatInfo

Атрибут	Описание
QList< Compressor:: CoderTypes> compressSequence	Последовательности компрессоров
unsigned int blockSize	Размер блока
off_t inputSize	Объём входных данных
off_t outputSize	Объём выходных данных
float speed	Скорость обработки данных

Атрибуты класса **StatInfoW** приведены в таблице 4.32

Методы класса приведены в таблице 4.33

Таблица 4.32 — Атрибуты класса StatInfoW

Атрибут	Описание
QLabel *blockSizeLabel	Надпись с текстом <размер блока>
QLabel *inputDataSizeLabel	Надпись с текстом <объём входных данных>
QLabel *outputDataSizeLabe	Надпись с текстом <объём выходных данных>
QLabel *ratioLabel	Надпись с текстом <степень сжатия, %>
QLabel *speedLabe	Надпись с текстом <скорость обработки данных>
QLabel *encodingSequenceLabel	Надпись с текстом <последовательность компрессоров>
QLabel * encodingSequenceColumnLabel	Подпись к последовательности компрессоров

Таблица 4.33 — Методы класса **StatInfoW**

Метод	Описание
Конструкторы	
StatInfoW(QWidget *parent = NULL)	Создаёт виджет с родительским **parent**
Слоты	
void showInfo(StatInfo info)	Выводит информацию из **info**

4.7 Графический пользовательский интерфейс

Графический интерфейс приложения представлен классом главного окна **Face** в котором используются вышеперечисленные классы.

4.7.1 Face

Класс **Face** является дочерним к стандартному классу главного окна Qt **QMainWindow**.

Атрибуты класса приведены в таблице 4.34

Таблица 4.34 — Атрибуты класса **Face**

Атрибут	Описание
CompressSettingsPanel * compressSettingsPanel	виджеты с элементами корректировки параметров выполнения сжатия
DecompressSettingsPanel * decompressSettingsPanel	виджеты с элементами корректировки параметров выполнения распаковки
StatInfoW *statInfoW	Окно со статистической информацией о выполненной операции
CompressorThread * compressThread	Поток сжатия
CompressorThread * decompressThread	Поток распаковки архива
CompressorThread * listArchiveThread	Поток получения содержимого архива
Ui::face widget	Оформление окна
Compressor *compressor	Компрессор
QToolBar *actionToolBar	Панель инструментов
FileList *filelist	Таблица, отображающая список обрабатываемых файлов и результат их обработки
QStringList sourceFileNames	Имена исходных файлов
QStringList encodedfileNames	Имена сжатых файлов

Продолжение на след. стр.

Продолжение таблицы 4.34

QString destFileName	Имя файла архива
QString destDirName	Имя каталога назначения
QDir compressBaseDir	Каталог, базовый для исходных файлов
QDir decompressBaseDir	Каталог, базовый для сжатых файлов
QDockWidget * compressSettingsDock	Прикрепляемый виджет для панели настройки параметров сжатия
QDockWidget * decompressSettingsDock	Прикрепляемый виджет для панели настройки параметров распаковки
QAction * selectFilesToCompressAction	Вызов диалога выбора файлов для сжатия
QAction * selectFileToDecompressAction	Вызов диалога выбора файла для распаковки
QAction *compressAction	Вызов сжатия
QAction *decompressAction	Вызов распаковки
QAction *exitAction	Вызов завершения работы приложения
QAction *aboutAction	Вызов справки
QAction *aboutQtAction	Вызов справки Qt
QMenu *fileMenu	Пункт меню <File>
QMenu *helpMenu	Пункт меню <Help>
QProgressDialog * compressingProgressDialog	Окно с информацией о прогрессе сжатия
QProgressDialog * decompressingProgressDialog	Окно с информацией о прогрессе распаковки
QProgressDialog * listArchiveContentsProgressDialog	Окно с информацией о прогрессе получения содержимого архива

Продолжение на след. стр.

Продолжение таблицы 4.34

bool brokenFileWarningShown	Флаг, если поднят, то предупреждение о повреждённости архива не будет показано
QList< Compressor:: CoderTypes> compressSequence	Последовательность кодировщиков
unsigned int blockSize	Размер блока
bool keepBrokenFiles	Флаг, если поднят, то распакованные повреждённые файлы не будут удалены

Методы класса приведены в таблице 4.35

Таблица 4.35 — Методы класса **Face**

Метод	Описание
Конструкторы	
Face( )	Создаёт виджет
Функции	
void activateCompressMode( )	Активирует режим сжатия
void activateDecompressMode( )	Активирует режим распаковки
void createActions( )	Создаёт действия
void createToolBars( )	Создаёт панели инструментов
void createMenus( )	Создаёт меню
void setupWidgetsConnections( )	Устанавливает связи сигнал-слот с виджетами
Слоты	
void about( )	Выводит сведения о приложении
void selectFilesToCompress( )	Выбор файлов для сжатия
void selectFileToDecompress( )	Выбор файла для распаковки
void compress( )	Сжатие
void decompress( )	Распаковка

Продолжение на след. стр.

Продолжение таблицы 4.35

void listArchiveContents( )	Получение содержимого архива
void displayCompressStatus(**int** progress, QString fileName, **float** speed)	Отображает сведения о степени завершённости сжатия **progress**, файла **fileName** с текущей скоростью **speed**
void displayDecompressStatus(**int** progress, QString fileName, **float** speed)	Отображает сведения о степени завершённости распаковки **progress**, файла **fileName** с текущей скоростью **speed**
void displayListStatus(**int** progress, QString fileName, **float** speed)	Отображает сведения о степени завершённости **progress** получения содержимого архива **fileName** с текущей скоростью **speed**
void showInfo(CTCompressorStatus::ErrorCode error, QString fileName)	Отображает код ошибки **errorCode** выполнения операции над файлом **fileName**
void showCompressStatInfo(Compressor::Stat stat)	Выводит статистическую информацию о сжатии **stat**
void showDecompressStatInfo(Compressor::Stat stat)	Выводит статистическую информацию о распаковке **stat**
void setCompressSettings(**unsigned int** blockSize, QList < Compressor::CoderTypes> compressSequence)	Устанавливает настройки сжатия: размер блока **blocksize**, последовательность кодировщиков **compressSequence**
void setDecompressSettings(**bool** keepBrokenFiles)	Устанавливает настройки распаковки: если флаг **keepBrokenFiles** поднят, то распакованные повреждённые файлы не будут удалены
void initSettings( )	Инициализирует настройки по-умолчанию

4.8 Прочее

4.8.1 DataUnitsToQString

Класс **DataUnitsToQString** служит для представления объёма, скорости передачи данных в виде строки QString с соответсвующим суффиксом (MB, KB/s,..).

Атрибуты отсутствуют.

Методы класса приведены в таблице 4.36

Таблица 4.36 — Методы класса **DataUnitsToQString**

Метод	Описание
Конструкторы	
DataUnitsToQString( )	Создаёт объект **parent**
Слоты	
static QString convertDataSize(off_t dataSize, **int** prec)	Возвращает строку, эквивалентную объёму **dataSize** с точностью **prec** и соответствующим суффиксом
static QString convertDataSpeed(**int** dataSpeed, **int** prec)	Возвращает строку, эквивалентную скорости передачи **dataSpeed** с точностью **prec** и соответствующим суффиксом

4.9 Главный модуль

В главном модуле происходит создание объекта класса главного окна приложения и отображения его на экране (листинг Б.41).

5 Интерфейс приложения

Внешний вид главного окна приложения после запуска приведён на рисунке 5.1

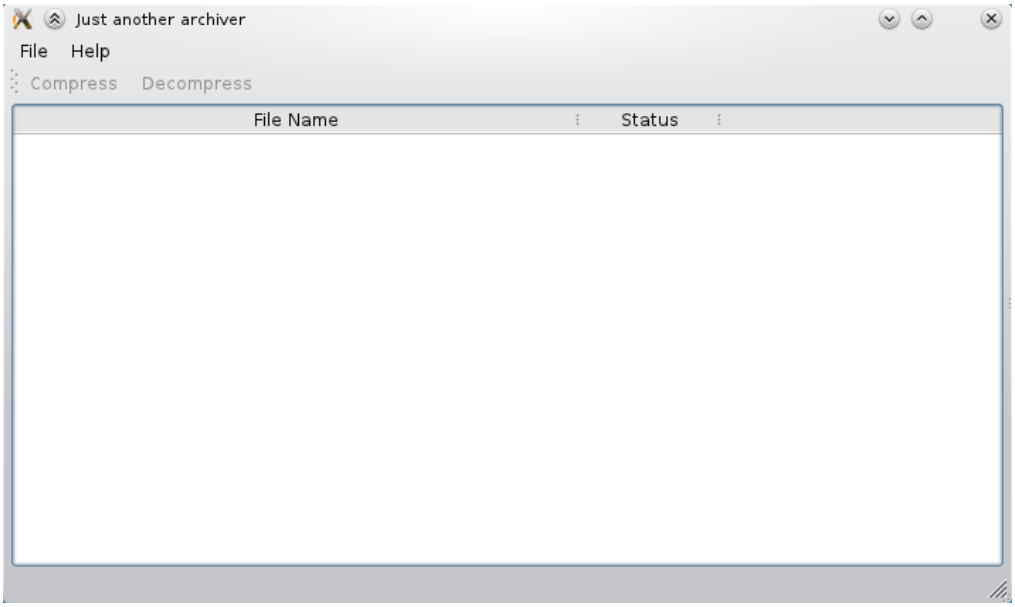


Рисунок 5.1 — Внешний вид главного окна приложения после запуска

Видно, что пункты панели инструментов сжать («Compress») и распаковать («Decompress») не доступны. Это объясняется тем, что не было выбрано файлов для упаковки или распаковки.

5.1 Упаковка файлов

Для начала необходимо выбрать файлы, которые следует упаковать.

Для этого выбираем пункт меню File->Select files to compress (рисунок 5.2)

Далее появляется окно диалога выбора файлов для архивации (рисунок 5.3)

После подтверждения выбора в главном окне появляется список выбранных файлов (рисунок 5.4). Так же становится доступным пункт панели инструментов Compress и слева появляется панель настроек сжатия.

5.1.1 Настройка параметров сжатия

Параметры настройки сжатия:

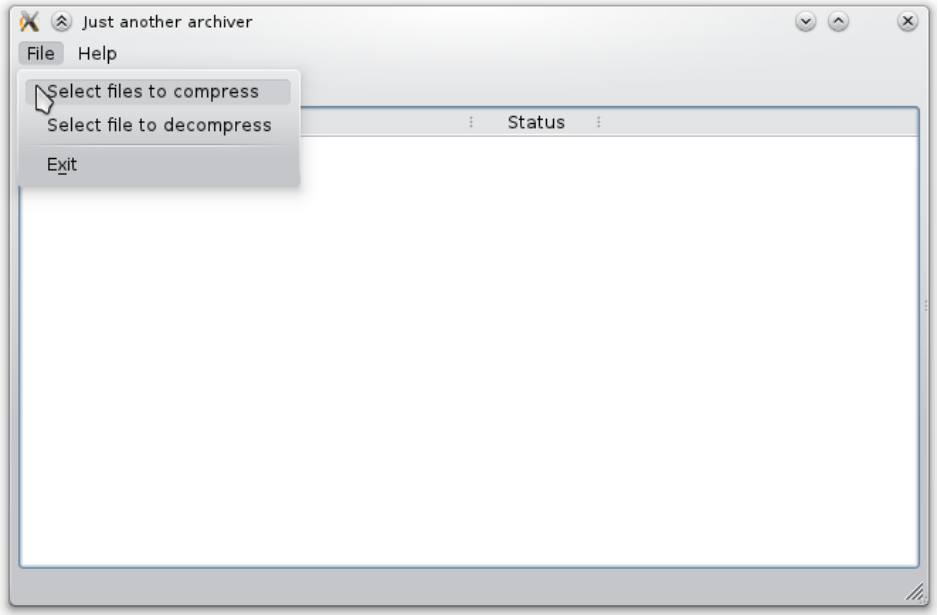


Рисунок 5.2 — Пункт меню выбора файлов для упаковки

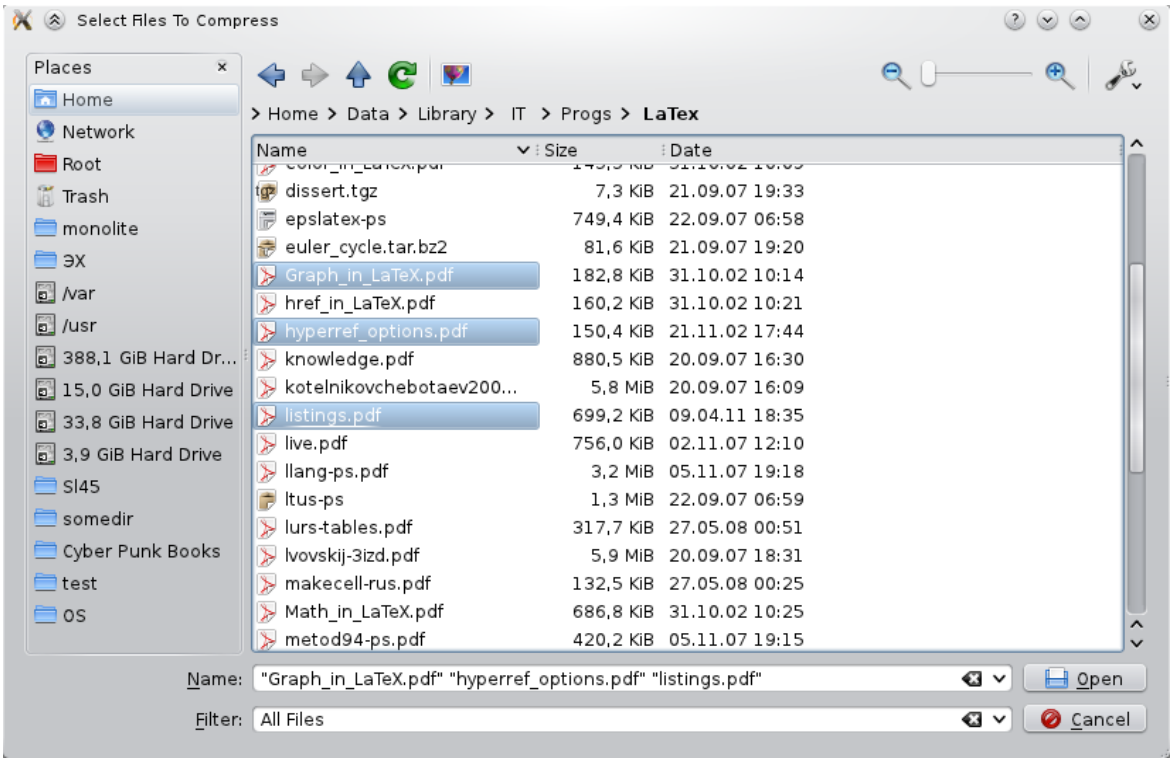


Рисунок 5.3 — Окно диалога выбора файлов для архивации

Block Size — размер блока в килобайтах. При увеличении повышается степень сжатия, однако снижается скорость.

1(2,3,..,8) Encoder — метод кодирования на 1(2,3,..,8) этапе

Disable All Encoders — отключить все кодировщики

Reset To Defaults — сброс настроек к исходным

Далее запускаем процесс сжатия.

5.1.2 Запуск

Для запуска сжатия служит пункт панели инструментов Compress (рисунок 5.5).

После нажатия на этой пункт появляется окно диалога выбора файла назначения (рисунок 5.6).

В случае возникновения ошибки доступа к сжимаемому файлу будет выведено предупреждение (рисунок 5.7)

По завершении сжатия будет выведено окно со статистической информацией (рисунок 5.8), такой как:

— последовательность кодировщиков

— размер блока

— объём входных файлов

— объём архива

— коэффициент сжатия

— средняя скорость обработки данных

В списке сжимаемых файлов будет сделана пометка о результатах сжатия (рисунок 5.9)

Видно, что сжатие 1-го и 3-го файлов прошло успешно, а сжатие 2-го завершилось неудачей.

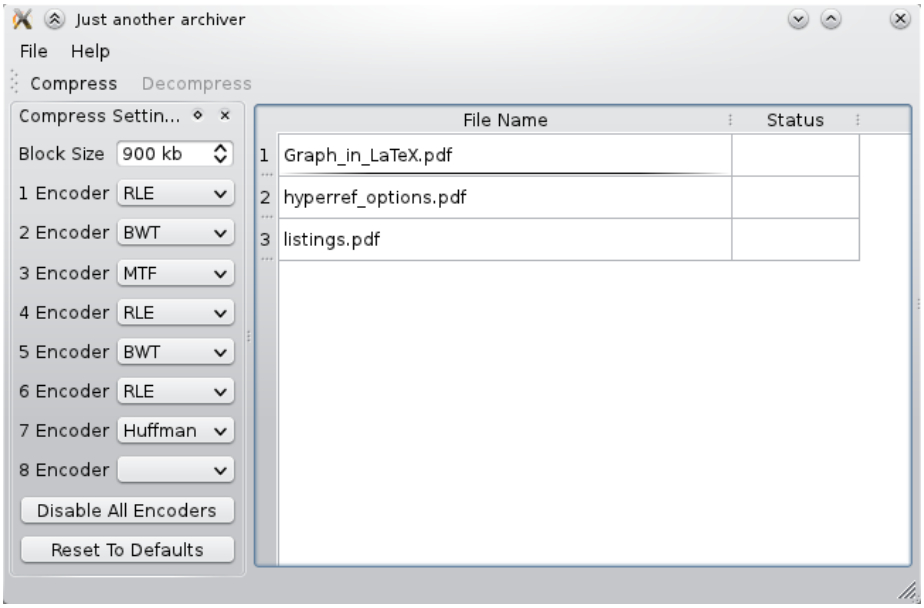


Рисунок 5.4 — Окно со списком файлов для архивации

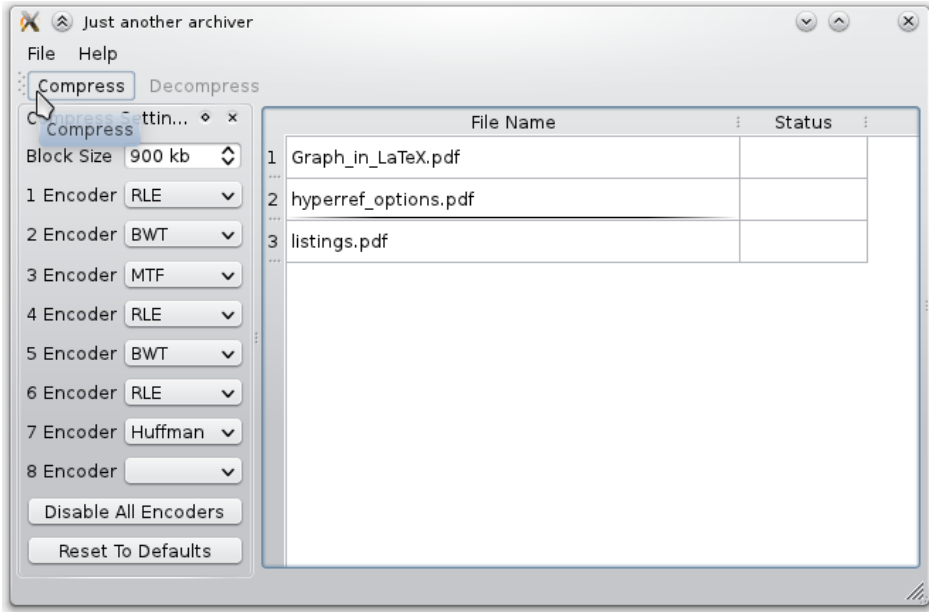


Рисунок 5.5 — Пункт панели инструментов Compress

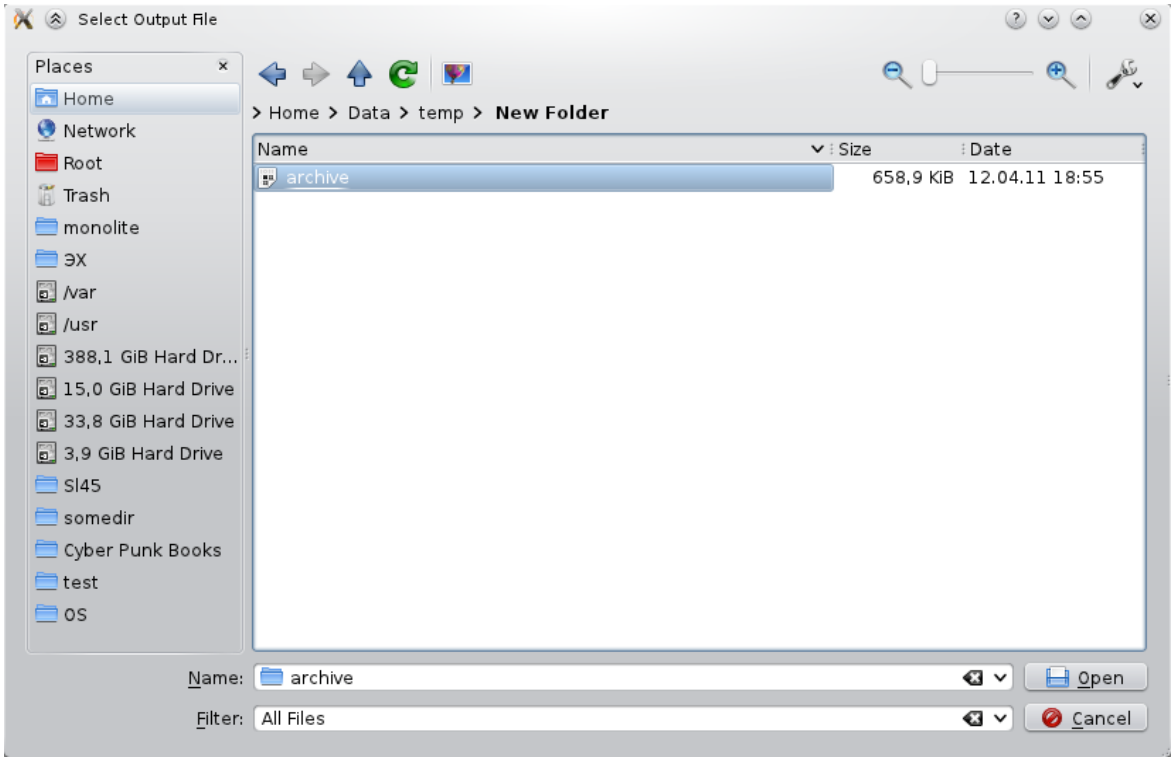


Рисунок 5.6 — Окно диалога выбора файла назначения

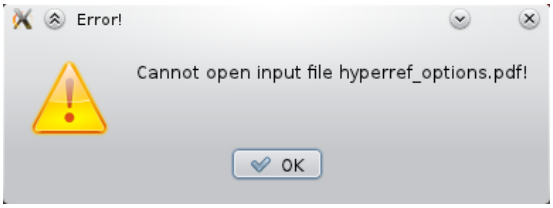


Рисунок 5.7 — Окно предупреждения об ошибке доступа к сжимаемому файлу

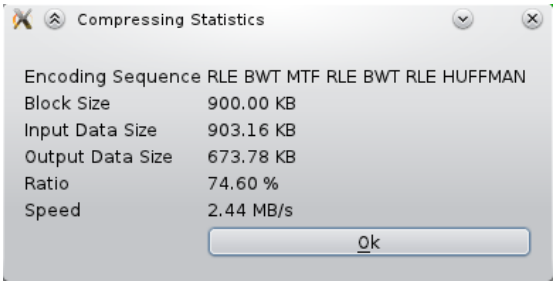


Рисунок 5.8 — Окно с информацией о результатах сжатия

5.2 Распаковка

Для начала необходимо выбрать файл-архив, который следует распаковать.

Для этого выбираем пункт меню File->Select file to decompress (рисунок 5.10)

Далее появляется окно диалога выбора файлов для распаковки (рисунок 5.11)

После подтверждения выбора в главном окне появляется содержимое архива (рисунок 5.13).

Если возникает ошибка при чтении архива выводится окно ошибки (рисунок 5.12)

Для каждого файла, входящего в состав архива выводится состояние ((не-)повреждён) (рисунок 5.13).

Так же становится доступным пункт панели инструментов Decompress и слева появляется панель настроек распаковки.

5.2.1 Настройка параметров распаковки

В качестве параметра распаковки присутствует флажок для указания оставлять ли распакованные повреждённые файлы.

Далее запускаем процесс распаковки.

5.2.2 Запуск

Для запуска распаковки служит пункт панели инструментов Decompress (рисунок 5.14).

По нажатии на этой пункт появляется окно диалога выбора каталога назначения (рисунок 5.15).

В случае возникновения ошибки записи распакованных файлов будет выведено окно ошибки (рисунок 5.16)

По завершении сжатия будет выведено окно со статистической информацией (рисунок 5.17), такой как:

— объём архива

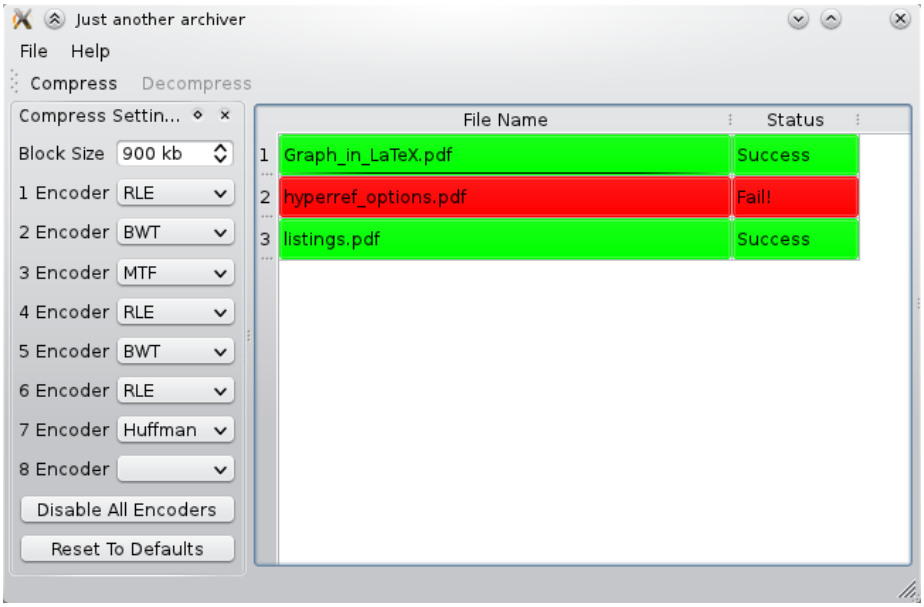


Рисунок 5.9 — Список файлов с результатами сжатия

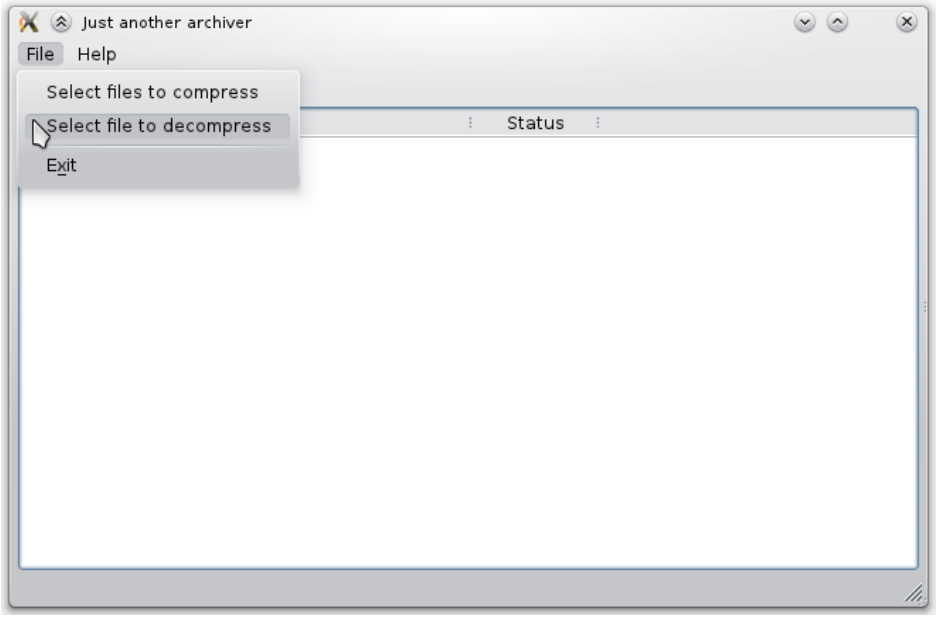


Рисунок 5.10 — Пункт меню выбора файла для распаковки

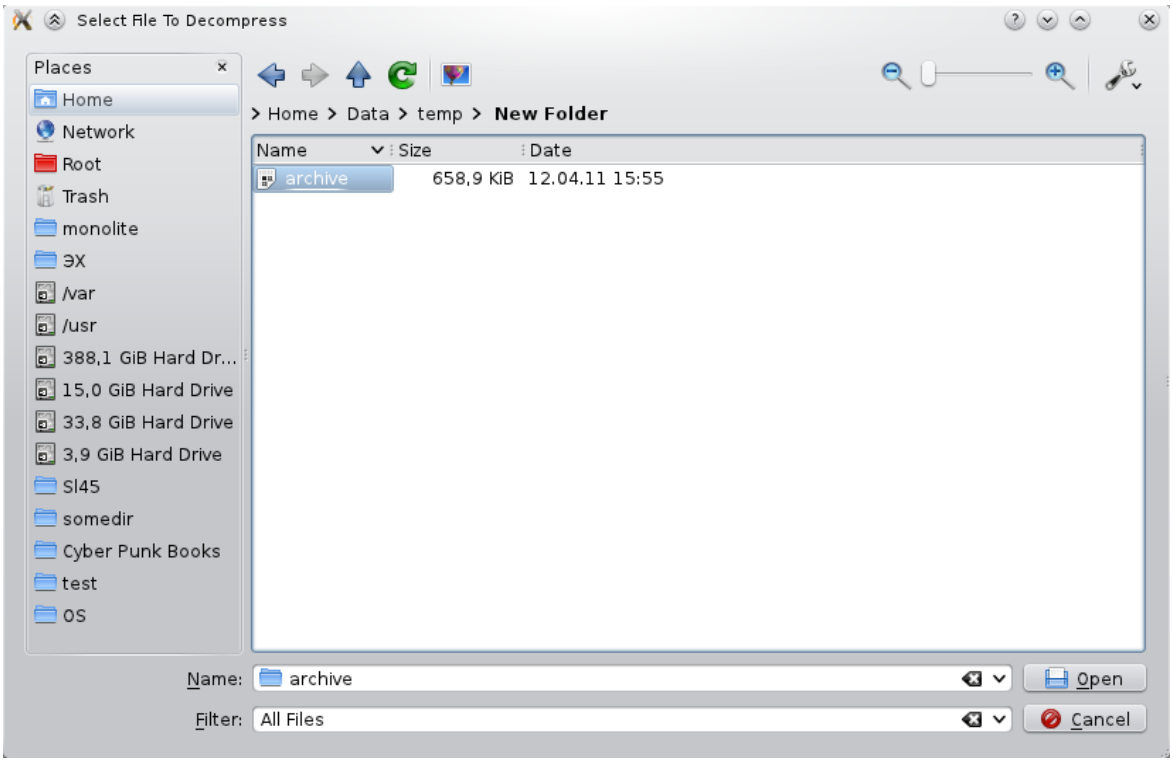


Рисунок 5.11 — Окно диалога выбора файла для распаковки

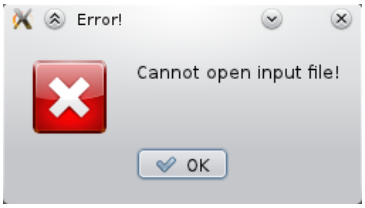


Рисунок 5.12 — Окно ошибки чтения файла архива

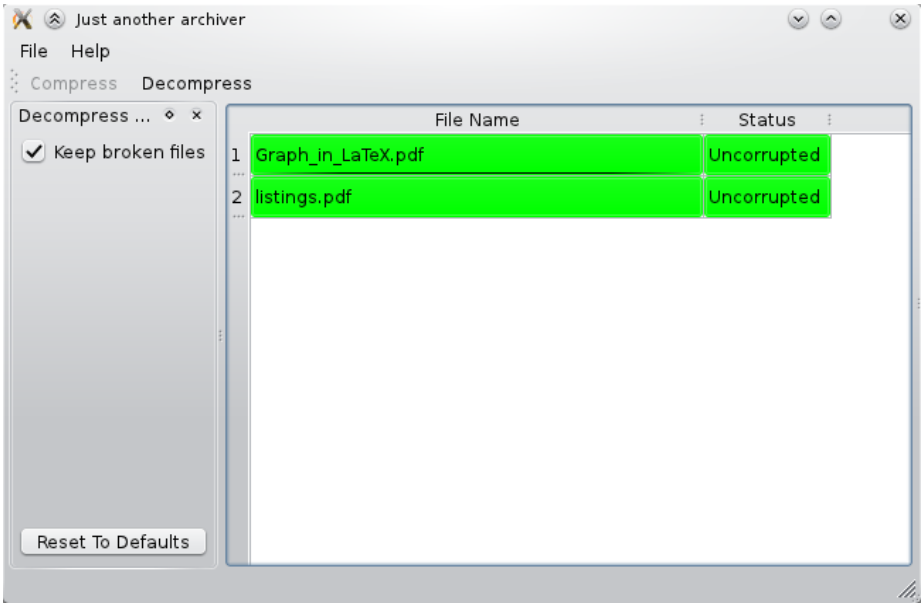


Рисунок 5.13 — Окно со списком файлов архива

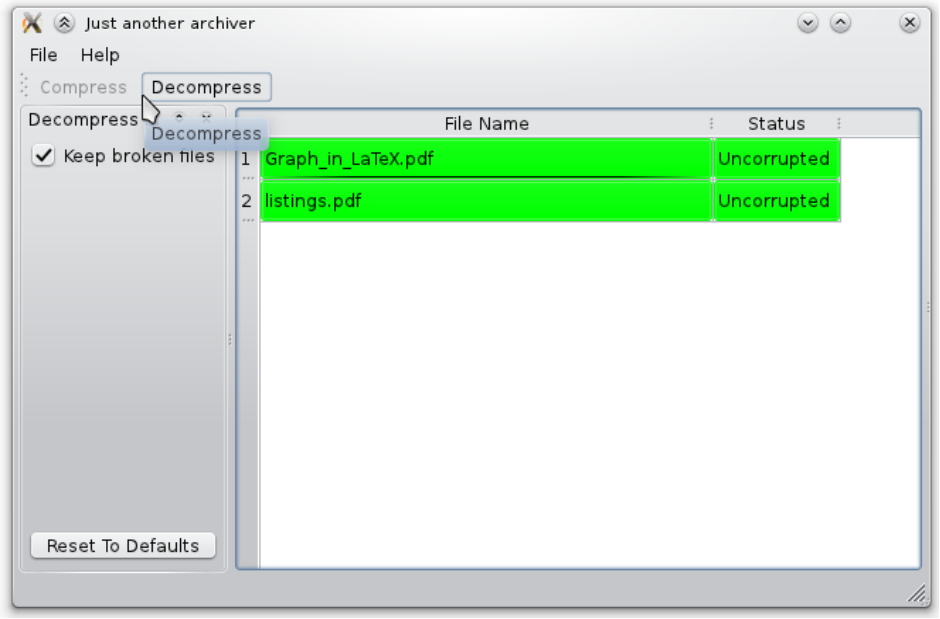


Рисунок 5.14 — Пункт панели инструментов Decompress

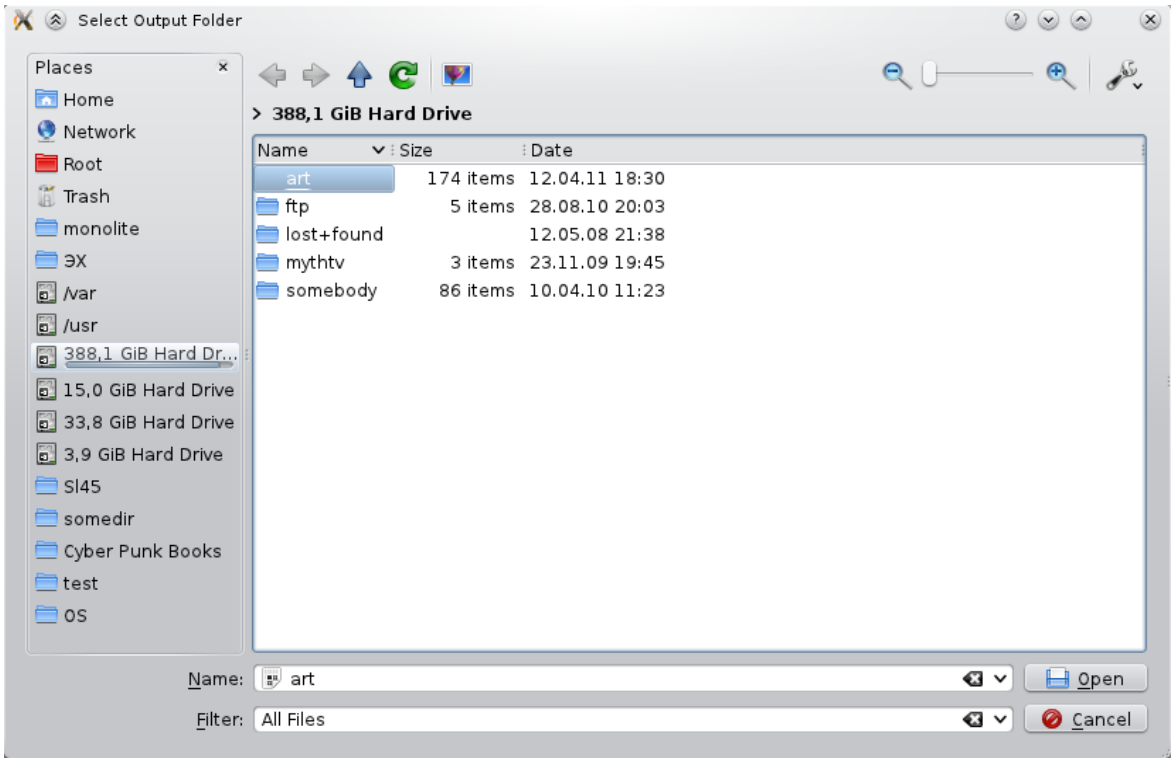


Рисунок 5.15 — Окно диалога выбора каталога назначения

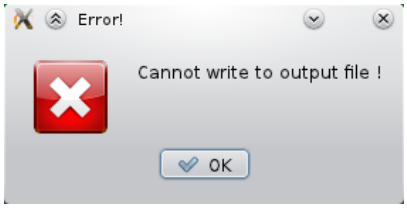


Рисунок 5.16 — Окно ошибки записи распакованных файлов

— объём распакованных файлов

— средняя скорость обработки данных

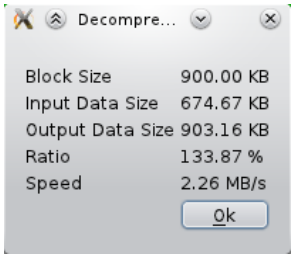


Рисунок 5.17 — Окно с информацией о результатах распаковки

В списке файлов архива будет сделана пометка о результатах распаковки (рисунок 5.18)

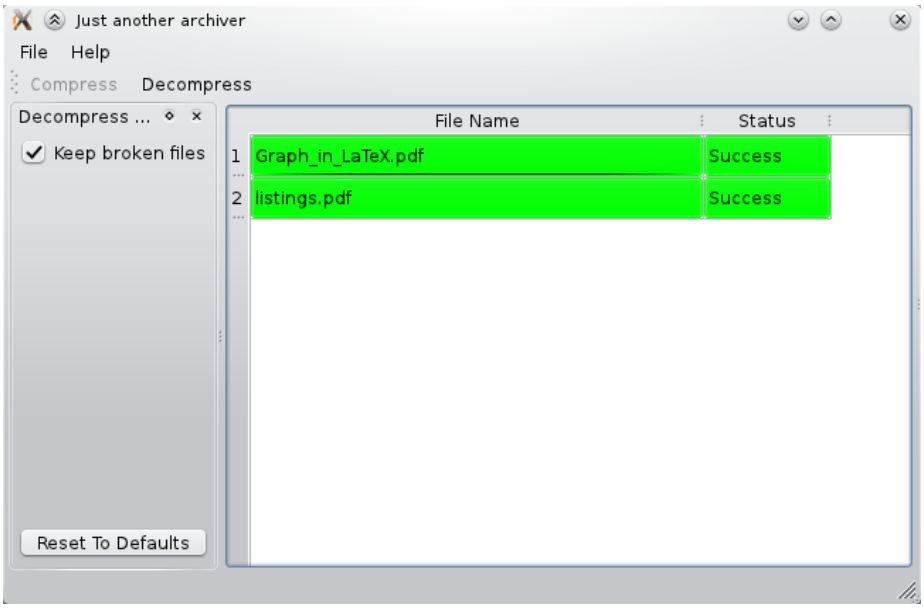


Рисунок 5.18 — Список файлов архива с результатами распаковки

5.2.3 Работа с повреждёнными архивами

В случае повреждения архива при его открытии для распаковки будет выведено предупреждение (рисунок 5.19). Так же в списке входящих в него файлов будет указаны повреждённые (рисунок 5.20).

Если в настройках распаковки указано оставлять повреждённые распакованные файлы (по-умолчанию), то при наличии в архиве хотя бы одного целого блока такого файла, неповреждённые данные будут записаны

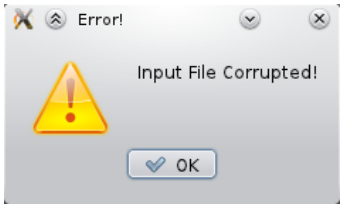


Рисунок 5.19 — Окно предупреждения о повреждённости архива

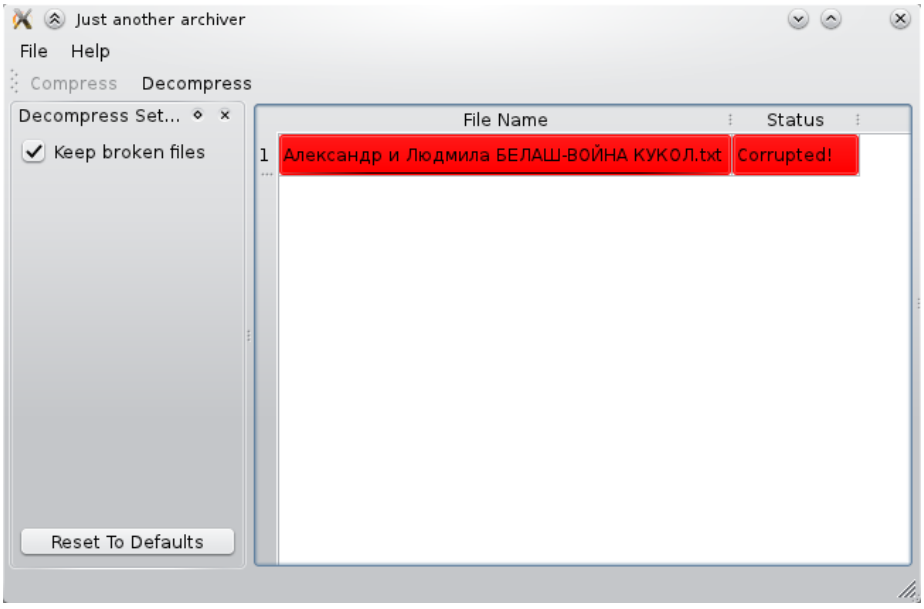


Рисунок 5.20 — Содержимое повреждённого архива

в выходной файл. Причём расположение уцелевших байт в распакованном файле будет совпадать с исходным. Вместо потерянных бит будет записан 0.

6 Результаты

Для проверки эффективности работы приложения было проведено тестирование коэффициента сжатия набора файлов Canterbury corpus [8].

Сжатие производилось разработанным приложением (JAA) с различной последовательностью кодирования и с использованием сторонних приложений, таких как:

— tar-1.26+bzip2-1.0.6 [3]

— tar-1.26+gzip-1.4

— WinRAR-3.80 [9]

Выбор приложений обусловлен их популярностью среди пользователей.

Результаты приведены в таблице 6.1

Из приведенных данных видно, что:

— степень сжатия данных разработанного приложения сопоставимо с результатами распространённых программных продуктов

— наилучшим коэффициент сжатия достигается при использовании связки методов кодирования RLE+BWT+MTF+RLE+Хаффмана кодирование

— предварительное кодирование методами BWT + MTF повышает эффективность сжатия в 2 раза (с 46.7 до 29.0%)

Таблица 6.1 — Результаты тестирования архиваторов

Архиватор	Параметры	Коэффициент сжатия, %
WinRAR	Максимальное сжатие	14.7
WinRAR	Обычное сжатие	18.4
tar+bzip2	-9	20.2
JAA	RLE+BWT+MTF+RLE+Huffman	20.4
JAA	BWT+MTF+RLE+Huffman	20.5
tar+gzip	-9	26.3
JAA	BWT+MTF+Huffman	29.0
JAA	MTF+Huffman	46.7
JAA	RLE+Huffman	46.8
JAA	Huffman	47.0
JAA	BWT + Huffman	47.1
JAA	RLE	91.7
JAA	Без сжатия	100.2

Заключение

На основе применённых алгоритмов кодирования информации было спроектировано и создано приложение, позволяющее проводить архивацию/компрессию и распаковку/декомпрессию данных. Степень сжатия сопоставимо с рядом распространённых архиваторов.

Достоинства разработанного приложения;

— были соблюдены принципы ООП, что в дальнейшем может облегчить использование всего кода приложения или его части при разработке новых программных продуктов

— были использованы кроссплатформенные библиотеки, что позволило создать исполнимые файлы для различных ОС с различной разрядностью (Gentoo Linux 64bit и MS Windows XP 32bit) компиляцией единого исходного кода

— UTF-8 кодировка имён файлов в архиве делает возможным обмен архивами между ОС с различной системной локалью

— возможность гибкой модификации алгоритма сжатия, что позволяет оценить эффективность различных последовательностей методов кодирования

— реализована возможность извлечения информации из повреждённых архивов

Были намечены пути совершенствования данного приложения:

— распараллелить алгоритмы (де-)компрессии для эффективного использования многопроцессорных систем

— реализовать возможность использования вычислительных мощностей современных видеокарт с помощью технологии OpenCL

— добавить новые, более эффективные методы кодирования, модифицировать имеющиеся

Список использованных источников

1. Snappy, compression/decompression library [Электронный ресурс]. — Режим доступа: http://code.google.com/p/snappy/. — Дата доступа: 03.04.2011.

2. Юкин В.А. Burrows wheeler transform faq [Электронный ресурс]. — Режим доступа: http://compression.ru/arctest/descript/bwt-faq.htm. — Дата доступа: 03.04.2011.

3. Julian Seward. bzip2 a program and library for data compression [Электронный ресурс]. — Режим доступа: http://bzip.org/. — Дата доступа: 03.04.2011.

4. Boost libraries for c++ [Электронный ресурс]. — Режим доступа: http://www.boost.org/. — Дата доступа: 09.04.2011.

5. Qt toolkit [Электронный ресурс]. — Режим доступа: http://qt.nokia.com/. — Дата доступа: 09.04.2011.

6. libdivsufsort project [Электронный ресурс]. — Режим доступа: http://code.google.com/p/libdivsufsort/. — Дата доступа: 09.04.2011.

7. Alexander Simakov. shcodec home page [Электронный ресурс]. — Режим доступа: http://webcenter.ru/~xander/. — Дата доступа: 09.04.2011.

8. Canterbury corpus [Электронный ресурс]. — Режим доступа: http://corpus.canterbury.ac.nz/index.html. — Дата доступа: 13.04.2011.

9. Winrar archiver [Электронный ресурс]. — Режим доступа: http://www.rarlab.com/. — Дата доступа: 13.04.2011.

Приложение А Содержимое компакт-диска

Содержимое компакт-диска:

Структура каталогов компакт-диска:

/**bin**/**win32**/**JAA.exe** — исполняемый win32 файл с необходимыми библиотеками

/**report**/**jaa.pdf** — электронный вариант пояснительной записки (pdf)

/**report**/**source**/ — электронный вариант пояснительной записки (LaTeX)

/**source**/ — исходный текст приложения

Приложение Б Листинги модулей

Б.1 Модуль DataBlock

Листинг Б.1 — **Compressor/DataBlock/dataBlock.h**

```
1  /*
2   * File:   dataBlock.h
3   * Author: art
4   *
5   * Created on 21 Январь 2011 г., 19:07
6   */
7
8  #ifndef DATABLOCK_H
9  #    define DATABLOCK_H
10
11 #    include <iostream>
12 #    include <sstream>
13 #    include <cstring>
14 #    include <vector>
15 #    include <stdint.h>
16 #    include <QFile>
17 #    include "../private/consts.h"
18 #    include "../CRC/crc.h"
19
20
21 using namespace std;
22
23 typedef vector < unsigned char > dataT;
24
25 class DataBlockHeader;
26
27 class DataBlock
28 {
29 private:
30     DataBlock( const DataBlock& );
31     void operator=(const DataBlock&) ;
32
33 public:
34     DataBlock( );
35     DataBlock( uint32_t block_size );
36     DataBlock( unsigned char *inData );
37     virtual         ~ DataBlock( );
38
39     /*read-write*/
40     DataBlockHeader *   readRAW( QFile &in );
41     int                 read( QFile &in );
42     int                 writeRAW( );
43     void                write( QFile &out );
44
45     /*data get*/
46     DataBlockHeader *   getHeader( );
47     unsigned char *     getData( );
48     void                setData( unsigned char* inData, uint32_t inDataSize );
49     void                setData( dataT* inData );
50     dataT               *getBlock( );
51     void                setBlock( unsigned char* inBlock );
52
```

```
53      /*Integrity check*/
54      int                     checkCRC( );
55      void                    recordCRC( );
56
57  private:
58      dataT                   *data;
59      uint32_t                dataSize;
60      uint32_t                nBytesToRead;
61      DataBlockHeader         *header;
62      dataT                   *outBlock;
63      bool                    recoveryMode;
64
65      uint32_t                calcCRC( );
66
67  };
68
69  #endif  /* DATABLOCK_H */
```

Листинг Б.2 — **Compressor/DataBlock/dataBlock.cpp**

```
1   /*
2    * File:   dataBlock.cpp
3    * Author: art
4    *
5    * Created on 21 Январь 2011 г., 19:07
6    */
7
8   #include <sstream>
9   #include <cstdlib>
10  #include <cstring>
11  #include <iostream>
12  #include <fstream>
13  #include "dataBlock.h"
14  #include "dataBlockHeader.h"
15  #include "readerDataBlockHeader.h"
16
17  #define MIN_RAW_BLOCK_SIZE (100000)
18
19  using namespace std;
20
21  /*
22   * TODO: setBlockSize*2 <-- !
23   */
24  DataBlock::DataBlock( unsigned int setBlockSize ) : data( new dataT( setBlockSize * 2 ) ),
25      dataSize( 0 ), nBytesToRead( setBlockSize ), header( new DataBlockHeader( ) ), outBlock(
            new dataT( setBlockSize * 2 ) ),
26      recoveryMode( false ) { }
27
28  DataBlock::DataBlock( ) : data( new vector < unsigned char > ), dataSize( 0 ), nBytesToRead(
        0 ),
29      header( new DataBlockHeader( ) ), outBlock( NULL ), recoveryMode( false )
30  {
31      data->reserve( MIN_RAW_BLOCK_SIZE );
32  }
33
34  DataBlock::DataBlock( unsigned char *inData ) : data( NULL ), dataSize( 0 ),
35      nBytesToRead( 0 ), header( NULL ), outBlock( NULL ), recoveryMode( false )
36  {
```

```
37      unsigned char in_header_data[HEADER_SIZE];
38      memcpy( in_header_data, inData, HEADER_SIZE );
39
40      header = new DataBlockHeader( in_header_data );
41      dataSize = header->getEncodedDataSize( );
42
43      data = new dataT( dataSize );
44      data->assign( inData + HEADER_SIZE, inData + HEADER_SIZE + dataSize );
45  }
46
47  DataBlock::~DataBlock( )
48  {
49      if ( header )
50          delete header;
51
52      if ( outBlock )
53          delete outBlock;
54
55      if ( data )
56          delete data;
57  }
58
59  DataBlockHeader *
60  DataBlock::readRAW( QFile &in )
61  {
62      uint64_t offset = in.pos( );
63      data->resize( nBytesToRead );
64      dataSize = in.read( ( char* ) data->data( ), nBytesToRead );
65      if ( !dataSize )
66          return NULL;
67
68      data->resize( dataSize );
69
70      header->initRAW( offset, dataSize );
71      recordCRC( );
72
73      return header;
74  }
75
76  int
77  DataBlock::read( QFile &in )
78  {
79      ReaderDataBlockHeader readerHeader;
80
81      switch ( readerHeader.read( header, in, recoveryMode ) )
82      {
83          case FILE_END: return FILE_END;
84              break;
85          case FILE_TOO_SMALL: return FILE_TOO_SMALL;
86              break;
87          case FILE_BROKEN:
88              recoveryMode = true;
89              return HEADER_CORRUPTED;
90              break;
91          default:;
92      }
93
94      data->clear( );
95
```

```
96      uint32_t origDataSize = header->getEncodedDataSize( );
97
98      data->resize( origDataSize );
99      dataSize = in.read( ( char* ) data->data( ), origDataSize );
100
101     if ( ( dataSize != origDataSize ) || ( checkCRC( ) ) )
102     {
103
104         return FILE_BROKEN;
105     }
106     return 0;
107 }
108
109 unsigned char *
110 DataBlock::getData( )
111 {
112     return data->data( );
113 }
114
115 void
116 DataBlock::setData( unsigned char* inData, unsigned int inDataSize )
117 {
118     dataSize = inDataSize;
119     data->assign( inData, inData + dataSize );
120 }
121
122 void
123 DataBlock::setData( dataT* inData )
124 {
125     data->assign( inData->begin( ), inData->end( ) );
126 }
127
128 void
129 DataBlock::write( QFile &out )
130 {
131     recordCRC( );
132     getBlock( );
133     out.write( ( char* ) outBlock->data( ), outBlock->size( ) );
134 }
135
136 DataBlockHeader *
137 DataBlock::getHeader( )
138 {
139     return header;
140 }
141
142 dataT *
143 DataBlock::getBlock( )
144 {
145     outBlock->assign( header->getData( ), header->getData( ) + HEADER_SIZE );
146     outBlock->insert( outBlock->end( ), data->begin( ), data->end( ) );
147
148     return outBlock;
149 }
150
151 void
152 DataBlock::setBlock( unsigned char* inBlock )
153 {
154     unsigned char in_header_data[HEADER_SIZE];
```

```
155     memcpy( in_header_data, inBlock, HEADER_SIZE );
156
157     header->setData( in_header_data );
158     dataSize = header->getEncodedDataSize( );
159     data->assign( inBlock + HEADER_SIZE, inBlock + HEADER_SIZE + dataSize );
160 }
161
162 unsigned int
163 DataBlock::calcCRC( )
164 {
165     unsigned int dataCRC = crc32( data->data( ), dataSize );
166     return dataCRC;
167 }
168
169 int
170 DataBlock::checkCRC( )
171 {
172     if ( calcCRC( ) != header->getDataCRC( ) )
173         return -1;
174     return 0;
175 }
176
177 void
178 DataBlock::recordCRC( )
179 {
180     header->setDataCRC( calcCRC( ) );
181     header->recordCRC( );
182 }
183
184 int
185 DataBlock::writeRAW( )
186 {
187     char * oFileName = header ->getFileName( );
188     QFile fout( QString::fromUtf8( oFileName, strlen( oFileName ) ) );
189
190     if ( !fout.open( QIODevice::ReadWrite ) )
191         return OUTPUT_ERROR;
192
193     fout.seek( header->getOffset( ) );
194
195     fout.write( ( char* ) data->data( ), data->size( ) );
196     fout.close( );
197
198     return 0;
199 }
```

Б.2 Модуль DataBlockHeader

Листинг Б.3 — **Compressor/DataBlock/dataBlockHeader.h**

```
1 /*
2  * File:   DataBlockHeader.h
3  * Author: art
4  *
5  * Created on 11 Март 2011 г., 17:00
6  */
```

```
7
8  #ifndef DATABLOCKHEADER_H
9  #    define DATABLOCKHEADER_H
10
11 #    include <stdint.h>
12 #    include <cstring>
13 #    include "../CRC/crc.h"
14 #    include "../private/consts.h"
15
16 #    define HEADER_SIZE (448)
17 #    define HEADER_DATA_SIZE (436)//After Header-CRC data size
18 #    define MAX_FILENAME_LENGTH (400)
19
20 class DataBlockHeader
21 {
22 public:
23     DataBlockHeader( );
24     virtual ~ DataBlockHeader( );
25
26     DataBlockHeader( unsigned char * in_header_data );
27
28     void        setCodecParams( uint32_t codec_params );
29     uint32_t    getCodecParams( );
30
31     void        setRAWDataSize( uint32_t RAWDataSize );
32     uint32_t    getRAWDataSize( ) const;
33
34     void        setEncodedDataSize( uint32_t encodedDataSize );
35     uint32_t    getEncodedDataSize( ) const;
36
37     void        setDecodedDataSize( uint32_t decodedDataSize );
38     uint32_t    getDecodedDataSize( ) const;
39
40     void        setId( uint64_t id );
41     uint64_t    getId( ) const;
42
43     void        setData( unsigned char *inHeaderData );
44     unsigned char * getData( );
45
46     void        setDataCRC( uint32_t crc );
47     uint32_t    getDataCRC( );
48
49     uint32_t    calcCRC( );
50     void        recordCRC( );
51     int         checkCRC( );
52
53     void        setFileName ( const char * fileName );
54     char *      getFileName ( );
55
56     void        setPart( const uint32_t part );
57     uint32_t    getPart( ) const;
58
59     void        setPartsCount( const uint32_t part );
60     uint32_t    getPartsCount( ) const;
61
62     void        setHeaderCRC( uint32_t headerCRC );
63     uint32_t    getHeaderCRC( ) const;
64
65     void        setOffset( uint64_t offset );
```

```
66      uint64_t    getOffset( );
67
68      void        clean( );
69      /**
70       * Init header of uncompressed data
71       * @param offset offset of the begin current block in source file
72       * @param in_data_size size of block
73       */
74      void        initRAW( uint64_t offset, const uint32_t rawDataSize );
75
76  private:
77
78      struct HeaderDataType
79      {
80          uint64_t id;
81          uint32_t headerCRC;
82          uint64_t offset;
83          uint32_t rawDataSize;
84          uint32_t decodedDataSize;
85          uint32_t encodedDataSize;
86          uint32_t encodedDataCRC;
87          uint32_t codecParams;
88          char     fileName[MAX_FILENAME_LENGTH];
89          uint32_t part;
90          uint32_t partsCount;
91
92      } __attribute__( ( packed ) ) headerData;
93
94      DataBlockHeader( const DataBlockHeader& );
95
96  };
97
98  #endif  /* DATABLOCKHEADER_H */
```

Листинг Б.4 — **Compressor/DataBlock/dataBlockHeader.cpp**

```
1   /*
2    * File:   DataBlockHeader.cpp
3    * Author: art
4    *
5    * Created on 11 Марта 2011 г., 17:00
6    */
7
8   #include "dataBlockHeader.h"
9
10  DataBlockHeader::DataBlockHeader( ) : headerData( ) { }
11
12  DataBlockHeader::~DataBlockHeader( ) { }
13
14  DataBlockHeader::DataBlockHeader( unsigned char * in_header_data ) : headerData( )
15  {
16      memcpy( &headerData, in_header_data, HEADER_SIZE );
17  }
18
19  void
20  DataBlockHeader::setCodecParams( uint32_t codec_params )
21  {
22      headerData.codecParams = codec_params;
```

```
23 }
24
25 uint32_t
26 DataBlockHeader::getCodecParams( )
27 {
28     return headerData.codecParams;
29 }
30
31 void
32 DataBlockHeader::setRAWDataSize( uint32_t RAWDataSize )
33 {
34     headerData.rawDataSize = RAWDataSize;
35 }
36
37 uint32_t
38 DataBlockHeader::getRAWDataSize( ) const
39 {
40     return headerData.rawDataSize;
41 }
42
43 void
44 DataBlockHeader::setEncodedDataSize( uint32_t encodedDataSize )
45 {
46     headerData.encodedDataSize = encodedDataSize;
47 }
48
49 uint32_t
50 DataBlockHeader::getEncodedDataSize( ) const
51 {
52     return headerData.encodedDataSize;
53 }
54
55 void
56 DataBlockHeader::setDecodedDataSize( uint32_t decodedDataSize )
57 {
58     headerData.decodedDataSize = decodedDataSize;
59 }
60
61 uint32_t
62 DataBlockHeader::getDecodedDataSize( ) const
63 {
64     return headerData.decodedDataSize;
65 }
66
67 void
68 DataBlockHeader::setId( uint64_t id )
69 {
70     this->headerData.id = id;
71 }
72
73 uint64_t
74 DataBlockHeader::getId( ) const
75 {
76     return headerData.id;
77 }
78
79 unsigned char *
80 DataBlockHeader::getData( )
81 {
```

```
82      return ( unsigned char* ) &headerData;
83  }
84
85  void
86  DataBlockHeader::setData( unsigned char *inHeaderData )
87  {
88      memcpy( &headerData, inHeaderData, HEADER_SIZE );
89  }
90
91  void
92  DataBlockHeader::clean( )
93  {
94      memset( &headerData, 0, HEADER_SIZE );
95  }
96
97  void
98  DataBlockHeader::setDataCRC( uint32_t crc )
99  {
100     headerData.encodedDataCRC = crc;
101 }
102
103 uint32_t
104 DataBlockHeader::getDataCRC( )
105 {
106     return headerData.encodedDataCRC;
107 }
108
109 void
110 DataBlockHeader::recordCRC( )
111 {
112     setHeaderCRC( calcCRC( ) );
113 }
114
115 int
116 DataBlockHeader::checkCRC( )
117 {
118     if ( calcCRC( ) != getHeaderCRC( ) )
119         return -1;
120     return 0;
121 }
122
123 void
124 DataBlockHeader::setFileName( const char * fileName )
125 {
126     strncpy( headerData.fileName, fileName, MAX_FILENAME_LENGTH );
127 }
128
129 char *
130 DataBlockHeader::getFileName( )
131 {
132     return headerData.fileName;
133 }
134
135 void
136 DataBlockHeader::setPart( const uint32_t part )
137 {
138     headerData.part = part;
139 }
140
```

```
141 uint32_t
142 DataBlockHeader::getPart( ) const
143 {
144     return headerData.part;
145 }
146
147 void
148 DataBlockHeader::setPartsCount( const uint32_t partsCount )
149 {
150     headerData.partsCount = partsCount;
151 }
152
153 uint32_t
154 DataBlockHeader::getPartsCount( ) const
155 {
156     return headerData.partsCount;
157 }
158
159 uint32_t
160 DataBlockHeader::calcCRC( )
161 {
162     getData( );
163     uint32_t headerCRC = crc32( ( unsigned char * ) ( &headerData.offset ), HEADER_DATA_SIZE
          );
164     return headerCRC;
165 }
166
167 void
168 DataBlockHeader::setHeaderCRC( uint32_t headerCRC )
169 {
170     this->headerData.headerCRC = headerCRC;
171 }
172
173 uint32_t
174 DataBlockHeader::getHeaderCRC( ) const
175 {
176     return headerData.headerCRC;
177 }
178
179 void
180 DataBlockHeader::setOffset( uint64_t offset )
181 {
182     headerData.offset = offset;
183 }
184
185 uint64_t
186 DataBlockHeader::getOffset( )
187 {
188     return headerData.offset;
189 }
190
191 void
192 DataBlockHeader::initRAW( uint64_t offset, const uint32_t rawDataSize )
193 {
194     memset( &headerData, 0, HEADER_SIZE );
195     headerData.id = RAW_ID;
196     headerData.decodedDataSize = rawDataSize;
197     headerData.encodedDataSize = rawDataSize; //входной и выходной размер равны
198     headerData.offset = offset;
```

```
199 }
```

Б.3 Модуль ReaderDataBlockHeader

Листинг Б.5 — **Compressor/DataBlock/readerDataBlockHeader.h**

```
1  /*
2   * File:   ReaderDataBlockHeader.h
3   * Author: art
4   *
5   * Created on 12 Марта 2011 г., 17:35
6   */
7
8  #ifndef READERDATABLOCKHEADER_H
9  #    define READERDATABLOCKHEADER_H
10
11 #    include <fstream>
12 #    include "QFile"
13
14 class DataBlockHeader;
15
16 class ReaderDataBlockHeader
17 {
18 public:
19     ReaderDataBlockHeader( );
20
21     ReaderDataBlockHeader( const ReaderDataBlockHeader& orig );
22
23     virtual ~ ReaderDataBlockHeader( );
24
25     int             read( DataBlockHeader * outHeader, QFile &in, bool searchHeader = false
           );
26
27 private:
28
29     bool            find ( DataBlockHeader *outHeader, QFile &fin);
30
31 } ;
32
33 #endif  /* READERDATABLOCKHEADER_H */
```

Листинг Б.6 — **Compressor/DataBlock/readerDataBlockHeader.cpp**

```
1  /*
2   * File:   ReaderDataBlockHeader.cpp
3   * Author: art
4   *
5   * Created on 12 Марта 2011 г., 17:35
6   */
7
8  #include "readerDataBlockHeader.h"
9  #include  "dataBlockHeader.h"
10
11 #include <algorithm>
12
13 #define BUFFER_SIZE (100000)
```

```
14
15 ReaderDataBlockHeader::ReaderDataBlockHeader( ) { }
16
17 ReaderDataBlockHeader::~ReaderDataBlockHeader( ) { }
18
19 int
20 ReaderDataBlockHeader::read( DataBlockHeader * outHeader, QFile &in, bool recoverMode )
21 {
22     unsigned char in_header_data[HEADER_SIZE];
23
24     if ( recoverMode )
25     {
26         if ( find( outHeader, in ) )
27         {
28             return FILE_END;
29         }
30     }
31     else
32     {
33         unsigned int receivedBytesCount = in.read( ( char* ) in_header_data, HEADER_SIZE );
34
35         if ( receivedBytesCount == 0 )
36             return FILE_END;
37
38         else if ( receivedBytesCount < HEADER_SIZE )
39             return FILE_TOO_SMALL;
40
41         outHeader->setData( in_header_data );
42
43         if ( outHeader->checkCRC( ) )
44             return FILE_BROKEN;
45     }
46
47     return 0;
48 }
49
50 bool
51 ReaderDataBlockHeader::find( DataBlockHeader *outHeader, QFile &fin )
52 {
53     unsigned char buffer[BUFFER_SIZE];
54
55     unsigned int nReadBytes;
56
57     while ( ( nReadBytes = fin.read( ( char* ) buffer, BUFFER_SIZE ) ) > HEADER_SIZE )
58     {
59         unsigned char * findPos = std::search( buffer, buffer + nReadBytes + 1, ARCHIVER_ID,
                ARCHIVER_ID + 7 );
60
61         if ( findPos == buffer + nReadBytes + 1 )
62         {
63             fin.seek( -( std::streamoff ) ( HEADER_SIZE - 1 )+ fin.pos() );
64         }
65         else
66         {
67             //found at findPos
68
69             if ( ( BUFFER_SIZE - ( findPos - buffer ) ) > HEADER_SIZE )
70             {
71                 //enough space in buffer
```

```
72
73                  outHeader->setData( findPos );
74
75                  if ( !outHeader->checkCRC( ) )
76                  {
77                      //valid CRC
78                      fin.seek( -( std::streamoff )( nReadBytes - ( findPos - buffer ) -
                            HEADER_SIZE ) + fin.pos()  );
79                      return 0;
80                  }
81                  else
82                  {
83                      //not valid CRC may be not header?
84                      fin.seek( -( std::streamoff )( nReadBytes - ( findPos - buffer ) - 2 )+
                            fin.pos() );
85                      //try again with offset
86                  }
87              }
88              else
89              {
90                  //not enough space
91                  fin.seek( -( nReadBytes - ( findPos - buffer ) - 1 )+fin.pos() );
92              }
93          }
94      }
95
96      return -1;
97  }
```

Б.4 Модуль crc

Листинг Б.7 — **Compressor/CRC/crc.h**

```
1  #ifndef CRC_H
2  #define CRC_H
3
4  #    include <stdint.h>
5  #    include <cstdlib>
6
7  uint_least32_t crc32(const unsigned char * buf, size_t len);
8
9
10 #endif /* CRC_H */
```

Листинг Б.8 — **Compressor/CRC/crc.cpp**

```
1  #include "crc.h"
2  #include <stddef.h>
3  #include <stdint.h>
4  /*
5    Name  : CRC-32
6    Poly  : 0x04C11DB7     x^32 + x^26 + x^23 + x^22 + x^16 + x^12 + x^11
7                          + x^10 + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1
8    Init  : 0xFFFFFFFF
9    Revert: true
10   XorOut: 0xFFFFFFFF
```

```
11    Check : 0xCBF43926 ("123456789")
12    MaxLen: 268 435 455 байт (2 147 483 647 бит) - обнаружение
13     одинарных, двойных, пакетных и всех нечетных ошибок
14  */
15  const uint_least32_t Crc32Table[256] = {
16      0x00000000, 0x77073096, 0xEE0E612C, 0x990951BA,
17      0x076DC419, 0x706AF48F, 0xE963A535, 0x9E6495A3,
18      0x0EDB8832, 0x79DCB8A4, 0xE0D5E91E, 0x97D2D988,
19      0x09B64C2B, 0x7EB17CBD, 0xE7B82D07, 0x90BF1D91,
20      0x1DB71064, 0x6AB020F2, 0xF3B97148, 0x84BE41DE,
21      0x1ADAD47D, 0x6DDDE4EB, 0xF4D4B551, 0x83D385C7,
22      0x136C9856, 0x646BA8C0, 0xFD62F97A, 0x8A65C9EC,
23      0x14015C4F, 0x63066CD9, 0xFA0F3D63, 0x8D080DF5,
24      0x3B6E20C8, 0x4C69105E, 0xD56041E4, 0xA2677172,
25      0x3C03E4D1, 0x4B04D447, 0xD20D85FD, 0xA50AB56B,
26      0x35B5A8FA, 0x42B2986C, 0xDBBBC9D6, 0xACBCF940,
27      0x32D86CE3, 0x45DF5C75, 0xDCD60DCF, 0xABD13D59,
28      0x26D930AC, 0x51DE003A, 0xC8D75180, 0xBFD06116,
29      0x21B4F4B5, 0x56B3C423, 0xCFBA9599, 0xB8BDA50F,
30      0x2802B89E, 0x5F058808, 0xC60CD9B2, 0xB10BE924,
31      0x2F6F7C87, 0x58684C11, 0xC1611DAB, 0xB6662D3D,
32      0x76DC4190, 0x01DB7106, 0x98D220BC, 0xEFD5102A,
33      0x71B18589, 0x06B6B51F, 0x9FBFE4A5, 0xE8B8D433,
34      0x7807C9A2, 0x0F00F934, 0x9609A88E, 0xE10E9818,
35      0x7F6A0DBB, 0x086D3D2D, 0x91646C97, 0xE6635C01,
36      0x6B6B51F4, 0x1C6C6162, 0x856530D8, 0xF262004E,
37      0x6C0695ED, 0x1B01A57B, 0x8208F4C1, 0xF50FC457,
38      0x65B0D9C6, 0x12B7E950, 0x8BBEB8EA, 0xFCB9887C,
39      0x62DD1DDF, 0x15DA2D49, 0x8CD37CF3, 0xFBD44C65,
40      0x4DB26158, 0x3AB551CE, 0xA3BC0074, 0xD4BB30E2,
41      0x4ADFA541, 0x3DD895D7, 0xA4D1C46D, 0xD3D6F4FB,
42      0x4369E96A, 0x346ED9FC, 0xAD678846, 0xDA60B8D0,
43      0x44042D73, 0x33031DE5, 0xAA0A4C5F, 0xDD0D7CC9,
44      0x5005713C, 0x270241AA, 0xBE0B1010, 0xC90C2086,
45      0x5768B525, 0x206F85B3, 0xB966D409, 0xCE61E49F,
46      0x5EDEF90E, 0x29D9C998, 0xB0D09822, 0xC7D7A8B4,
47      0x59B33D17, 0x2EB40D81, 0xB7BD5C3B, 0xC0BA6CAD,
48      0xEDB88320, 0x9ABFB3B6, 0x03B6E20C, 0x74B1D29A,
49      0xEAD54739, 0x9DD277AF, 0x04DB2615, 0x73DC1683,
50      0xE3630B12, 0x94643B84, 0x0D6D6A3E, 0x7A6A5AA8,
51      0xE40ECF0B, 0x9309FF9D, 0x0A00AE27, 0x7D079EB1,
52      0xF00F9344, 0x8708A3D2, 0x1E01F268, 0x6906C2FE,
53      0xF762575D, 0x806567CB, 0x196C3671, 0x6E6B06E7,
54      0xFED41B76, 0x89D32BE0, 0x10DA7A5A, 0x67DD4ACC,
55      0xF9B9DF6F, 0x8EBEEFF9, 0x17B7BE43, 0x60B08ED5,
56      0xD6D6A3E8, 0xA1D1937E, 0x38D8C2C4, 0x4FDFF252,
57      0xD1BB67F1, 0xA6BC5767, 0x3FB506DD, 0x48B2364B,
58      0xD80D2BDA, 0xAF0A1B4C, 0x36034AF6, 0x41047A60,
59      0xDF60EFC3, 0xA867DF55, 0x316E8EEF, 0x4669BE79,
60      0xCB61B38C, 0xBC66831A, 0x256FD2A0, 0x5268E236,
61      0xCC0C7795, 0xBB0B4703, 0x220216B9, 0x5505262F,
62      0xC5BA3BBE, 0xB2BD0B28, 0x2BB45A92, 0x5CB36A04,
63      0xC2D7FFA7, 0xB5D0CF31, 0x2CD99E8B, 0x5BDEAE1D,
64      0x9B64C2B0, 0xEC63F226, 0x756AA39C, 0x026D930A,
65      0x9C0906A9, 0xEB0E363F, 0x72076785, 0x05005713,
66      0x95BF4A82, 0xE2B87A14, 0x7BB12BAE, 0x0CB61B38,
67      0x92D28E9B, 0xE5D5BE0D, 0x7CDCEFB7, 0x0BDBDF21,
68      0x86D3D2D4, 0xF1D4E242, 0x68DDB3F8, 0x1FDA836E,
69      0x81BE16CD, 0xF6B9265B, 0x6FB077E1, 0x18B74777,
```

```
70      0x88085AE6, 0xFF0F6A70, 0x66063BCA, 0x11010B5C,
71      0x8F659EFF, 0xF862AE69, 0x616BFFD3, 0x166CCF45,
72      0xA00AE278, 0xD70DD2EE, 0x4E048354, 0x3903B3C2,
73      0xA7672661, 0xD06016F7, 0x4969474D, 0x3E6E77DB,
74      0xAED16A4A, 0xD9D65ADC, 0x40DF0B66, 0x37D83BF0,
75      0xA9BCAE53, 0xDEBB9EC5, 0x47B2CF7F, 0x30B5FFE9,
76      0xBDBDF21C, 0xCABAC28A, 0x53B39330, 0x24B4A3A6,
77      0xBAD03605, 0xCDD70693, 0x54DE5729, 0x23D967BF,
78      0xB3667A2E, 0xC4614AB8, 0x5D681B02, 0x2A6F2B94,
79      0xB40BBE37, 0xC30C8EA1, 0x5A05DF1B, 0x2D02EF8D
80  };
81
82  uint_least32_t crc32(const unsigned char * buf, size_t len)
83  {
84      uint_least32_t crc = 0xFFFFFFFF;
85      while (len--)
86          crc = (crc >> 8) ^ Crc32Table[(crc ^ *buf++) & 0xFF];
87      return crc ^ 0xFFFFFFFF;
88  }
```

Б.5 Модуль FilesTable

Листинг Б.9 — **Compressor/FilesTable/filesTable.h**

```
1   /*
2    * File:   DataBlocksTable.h
3    * Author: art
4    *
5    * Created on 12 Марта 2011 г., 9:10
6    */
7
8   #ifndef DATABLOCKSTABLE_H
9   #    define DATABLOCKSTABLE_H
10
11  #    include <stdint.h>
12  #    include <map>
13  #    include <string>
14  #    include <vector>
15  #    include "fileBlocksInfo.h"
16
17  using namespace std;
18
19  class DataBlockHeader;
20
21  class FilesTable
22  {
23  public:
24
25      struct FileInfo
26      {
27          string fileName;
28          bool corrupted;
29      } ;
30
31      FilesTable( );
32
```

```
33      FilesTable( const FilesTable& );
34      virtual                                 ~FilesTable( );
35
36      int                                     add( DataBlockHeader * inHeader, unsigned int id =
            0 );
37      void                                    remove( DataBlockHeader * inHeader );
38      map< string, vector <uint32_t> > *      getNonCompleteFilesBlocksInfo( );
39      vector<string> *                        getNonCompleteFilesNames( );
40      vector<FileInfo> *                      getArchiveContent( );
41      void                                    clean( );
42      unsigned int                            getId( DataBlockHeader * inHeader ) const;
43      unsigned int                            getId( string fileName ) const;
44
45  private:
46      map< string, FileBlocksInfo >           fileBlocksTable; //сведения о полученных блоках
            файла
47      map< string, vector <uint32_t> >        brokenFiletable;
48      vector <string>                         brokenFilesNames;
49      vector<FileInfo>                        archiveContent;
50
51  };
52
53  #endif  /* DATABLOCKSTABLE_H */
```

Листинг Б.10 — **Compressor/FilesTable/filesTable.cpp**

```
1   /*
2    * File:   DataBlocksTable.cpp
3    * Author: art
4    *
5    * Created on 12 Марта 2011 г., 9:10
6    */
7
8   #include "filesTable.h"
9   #include "../DataBlock/dataBlockHeader.h"
10
11  FilesTable::FilesTable( ) : fileBlocksTable( ), brokenFiletable( ), brokenFilesNames( ),
        archiveContent( ) { }
12
13  FilesTable::~FilesTable( ) { }
14
15  int
16  FilesTable::add( DataBlockHeader * inHeader, unsigned int id )
17  {
18      int result = 0;
19
20      map< string, FileBlocksInfo >::iterator pos = fileBlocksTable.find( string( inHeader->
            getFileName( ) ) );
21
22      if ( pos == fileBlocksTable.end( ) )
23      {
24          FileBlocksInfo blockInfo( inHeader->getPartsCount( ), id );
25
26          result = FIRST_RECIEVED_BLOCK;
27
28          switch ( blockInfo.setRecievedBlock( inHeader->getPart( ) ) )
29          {
30              case BLOCK_OUT_OF_RANGE: result = BLOCK_OUT_OF_RANGE;
```

```
31                  break;
32              case ALL_BLOCKS_RECIEVED: result = ALL_BLOCKS_RECIEVED;
33                  break;
34              case FIRST_AND_LAST_RECIEVED_BLOCK: result = FIRST_AND_LAST_RECIEVED_BLOCK;
35                  break;
36              default: break;
37          }
38
39          fileBlocksTable.insert( map< string, FileBlocksInfo >::value_type( inHeader->
                getFileName( ), blockInfo ) );
40      }
41      else
42      {
43          switch ( pos->second.setRecievedBlock( inHeader->getPart( ) ) )
44          {
45              case BLOCK_OUT_OF_RANGE: result = BLOCK_OUT_OF_RANGE;
46                  break;
47              case ALL_BLOCKS_RECIEVED: result = ALL_BLOCKS_RECIEVED;
48                  break;
49              case FIRST_AND_LAST_RECIEVED_BLOCK: result = FIRST_AND_LAST_RECIEVED_BLOCK;
50                  break;
51              default: break;
52          }
53      }
54
55      return result;
56  }
57
58  void
59  FilesTable::clean( )
60  {
61      fileBlocksTable.clear( );
62  }
63
64  unsigned int
65  FilesTable::getId( DataBlockHeader * inHeader ) const
66  {
67      map< string, FileBlocksInfo >::const_iterator i = fileBlocksTable.find( string( inHeader
            ->getFileName( ) ) );
68      if ( i == fileBlocksTable.end( ) )
69          return 0;
70      return i->second.getId( );
71  }
72
73  unsigned int
74  FilesTable::getId( string fileName ) const
75  {
76      map< string, FileBlocksInfo >::const_iterator i = fileBlocksTable.find( fileName );
77      if ( i == fileBlocksTable.end( ) )
78          return 0;
79      return i->second.getId( );
80  }
81
82  void
83  FilesTable::remove( DataBlockHeader * inHeader )
84  {
85      fileBlocksTable.erase( string( inHeader->getFileName( ) ) );
86  }
87
```

```
88  map< string, vector <uint32_t> > *
89  FilesTable::getNonCompleteFilesBlocksInfo( )
90  {
91      brokenFiletable.clear( );
92      for ( map< string, FileBlocksInfo >::iterator i = fileBlocksTable.begin( ), end =
            fileBlocksTable.end( ); i != end; ++i )
93          if ( i->second.getNonRecievedBlocksInfo( ) )
94              brokenFiletable.insert( make_pair( i->first, *( i->second.
                    getNonRecievedBlocksInfo( ) ) ) );
95      return &brokenFiletable;
96  }
97
98  vector<string> *
99  FilesTable::getNonCompleteFilesNames( )
100 {
101     brokenFilesNames.clear( );
102
103     for ( map< string, FileBlocksInfo >::iterator i = fileBlocksTable.begin( ), end =
            fileBlocksTable.end( ); i != end; ++i )
104         if ( !i->second.complete( ) )
105             brokenFilesNames.push_back( i->first );
106     if ( brokenFilesNames.empty( ) )
107         return NULL;
108
109     return &brokenFilesNames;
110 }
111
112 vector<FilesTable::FileInfo>*
113 FilesTable::getArchiveContent( )
114 {
115     archiveContent.clear( );
116     for ( map< string, FileBlocksInfo >::iterator i = fileBlocksTable.begin( ), end =
            fileBlocksTable.end( ); i != end; ++i )
117     {
118         FileInfo info;
119         info.fileName = i->first;
120         if ( i->second.complete( ) )
121             info.corrupted = false;
122         else
123             info.corrupted = true;
124         archiveContent.push_back( info );
125     }
126     return &archiveContent;
127 }
```

Б.6 Модуль FileBlocksInfo

Листинг Б.11 — **Compressor/FilesTable/fileBlocksInfo.h**

```
1  /*
2   * File:   FileBlocksInfo.h
3   * Author: art
4   *
5   * Created on 12 Март 2011 г., 11:12
6   */
7
```

```
8  #ifndef FILEBLOCKSINFO_H
9  #    define FILEBLOCKSINFO_H
10
11 #    include <string.h>
12 #    include <stdint.h>
13 #    include <boost/dynamic_bitset.hpp>
14 #    include <vector>
15
16 using namespace std;
17
18 class FileBlocksInfo
19 {
20 public:
21
22
23     FileBlocksInfo( const FileBlocksInfo& orig );
24
25     FileBlocksInfo( const uint32_t _totalBlocks, unsigned int _id = 0 );
26
27     virtual ~FileBlocksInfo( );
28
29     uint32_t                    getTotalBlocks( );
30     int                         setRecievedBlock( uint32_t blockN );
31     vector<uint32_t> *          getNonRecievedBlocksInfo( );
32     bool                        complete( );
33     unsigned int                getId( ) const;
34
35 private:
36     uint32_t                    blocksCount;
37     boost::dynamic_bitset<>     blocks; //array of block recieving status: false - recieved,
           true - not
38     vector< uint32_t>           nonRecievedblocks;
39     unsigned int                id; //file 's ID in archive
40
41 } ;
42
43 #endif  /* FILEBLOCKSINFO_H */
```

Листинг Б.12 — **Compressor/FilesTable/fileBlocksInfo.cpp**

```
1  /*
2   * File:   FileBlocksInfo.cpp
3   * Author: art
4   *
5   * Created on 12 Март 2011 г., 11:12
6   */
7
8  #include <iostream>
9  #include "fileBlocksInfo.h"
10 #include "../private/consts.h"
11
12
13 FileBlocksInfo::FileBlocksInfo( const FileBlocksInfo& orig ) : blocksCount( 0 ), blocks( ),
       nonRecievedblocks( ), id( )
14 {
15     blocks.resize( orig.blocks.size( ) );
16     blocks = orig.blocks;
17     blocksCount = orig.blocksCount;
```

```
18      id = orig.id;
19  }
20
21  FileBlocksInfo::FileBlocksInfo( const uint32_t _totalBlocks, unsigned int _id ) :
        blocksCount( _totalBlocks ), blocks( ), nonRecievedblocks( ), id( _id )
22  {
23      blocks.resize( _totalBlocks, true );
24      nonRecievedblocks.reserve( _totalBlocks );
25  }
26
27  FileBlocksInfo::~FileBlocksInfo( ) { }
28
29  bool
30  FileBlocksInfo::complete( )
31  {
32      return blocks.none( );
33  }
34
35  unsigned int
36  FileBlocksInfo::getId( ) const
37  {
38      return id;
39  }
40
41  uint32_t
42  FileBlocksInfo::getTotalBlocks( )
43  {
44      return blocksCount;
45  }
46
47  int
48  FileBlocksInfo::setRecievedBlock( uint32_t blockN )
49  {
50      if ( blockN > blocksCount )
51          return BLOCK_OUT_OF_RANGE;
52      if ( !blocks[blockN] )
53          return BLOCK_ALREADY_RECIEVED;
54      blocks[blockN] = false;
55
56      if ( blocksCount == 1 )/* when first block is the last too*/
57          return FIRST_AND_LAST_RECIEVED_BLOCK;
58
59      if ( complete( ) )
60          return ALL_BLOCKS_RECIEVED;
61      return 0;
62  }
63
64  vector<uint32_t> *
65  FileBlocksInfo::getNonRecievedBlocksInfo( )
66  {
67      nonRecievedblocks.clear( );
68      if ( blocks.any( ) )
69      {
70          for ( unsigned int i = 0; i < blocks.size( ); ++i )
71          {
72              if ( blocks[i] )
73                  nonRecievedblocks.push_back( i );
74          }
75          return &nonRecievedblocks;
```

```
76      }
77      return NULL;
78  }
```

Б.7 Модуль CodecAbstract

Листинг Б.13 — **Compressor/Codec/codecAbstract.h**

```
1   /*
2    * File:   Codec_abstract.h
3    * Author: art
4    *
5    * Created on 5 Февраль 2011 г., 19:16
6    */
7
8   #ifndef CODEC_ABSTRACT_H
9   #   define CODEC_ABSTRACT_H
10  #   include <stdint.h>
11
12  using namespace std;
13
14  class DataBlockHeader;
15  class DataBlock;
16
17  class Codec_abstract
18  {
19  public:
20      Codec_abstract( );
21      virtual ~ Codec_abstract( ) = 0;
22
23  protected:
24      unsigned char   *data;
25      unsigned int    decodedDataSize;
26      unsigned int    encodedDataSize;
27      uint32_t        codecParams;
28
29      void            initDecoder( DataBlock * in_block );
30      void            initEncoder( DataBlock * in_block );
31      void            recordOutHeader( DataBlockHeader *outHeader, const uint64_t id );
32
33  private:
34      Codec_abstract( const Codec_abstract& );
35      void operator=(const Codec_abstract&) ;
36  } ;
37
38  #endif  /* CODEC_ABSTRACT_H */
```

Листинг Б.14 — **Compressor/Codec/codecAbstract.cpp**

```
1   /*
2    * File:   Codec_abstract.cpp
3    * Author: art
4    *
5    * Created on 5 Февраль 2011 г., 19:16
6    */
7
```

```
8  #include "codecAbstract.h"
9  #    include "../DataBlock/dataBlock.h"
10 #include "../DataBlock/dataBlockHeader.h"
11
12 Codec_abstract::Codec_abstract( ) : data( 0 ), decodedDataSize( 0 ),
13     encodedDataSize( 0 ), codecParams( 0 ) { }
14
15 Codec_abstract::~Codec_abstract( ) { }
16
17 void
18 Codec_abstract::initDecoder( DataBlock * in_block )
19 {
20     DataBlockHeader *header = in_block->getHeader( );
21
22     data = in_block->getData( );
23     encodedDataSize = header->getEncodedDataSize( );
24     decodedDataSize = header->getDecodedDataSize( );
25     codecParams     = header->getCodecParams( );
26 }
27
28 void
29 Codec_abstract::initEncoder( DataBlock * in_block )
30 {
31     dataT * vect = in_block->getBlock( );
32     data = vect->data( );
33     decodedDataSize = vect->size( );
34 }
35
36 void
37 Codec_abstract::recordOutHeader( DataBlockHeader *outHeader, const uint64_t id )
38 {
39     outHeader->clean( );
40     outHeader->setId( id );
41     outHeader->setDecodedDataSize( decodedDataSize );
42     outHeader->setEncodedDataSize( encodedDataSize );
43     outHeader->setCodecParams( codecParams );
44 }
```

Б.8 Модуль Codec

Листинг Б.15 — **Compressor/Codec/codec.h**

```
1  /*
2   * File:   Codec.h
3   * Author: art
4   *
5   * Created on 5 Февраль 2011 г., 19:44
6   */
7
8  #ifndef CODEC_H
9  #define CODEC_H
10
11 #include "Codecs/BWT/codecBWT.h"
12 #include "Codecs/MTF/codecMTF.h"
13 #include "Codecs/RLE/codecRLE.h"
14 #include "Codecs/HUFF/codecHUFF.h"
```

```
15
16 class Codec : public virtual Codec_BWT, public virtual Codec_HUFF, public virtual Codec_MTF,
       public virtual Codec_RLE
17 {
18 public:
19     Codec();
20     virtual ~Codec();
21 private:
22
23 };
24
25 #endif  /* CODEC_H */
```

Листинг Б.16 — **Compressor/Codec/codec.cpp**

```
1  /*
2   * File:   Codec.cpp
3   * Author: art
4   *
5   * Created on 5 Февраль 2011 г., 19:44
6   */
7
8  #include "codec.h"
9
10 Codec::Codec()
11 {
12 }
13
14 Codec::~Codec()
15 {
16 }
```

Б.9 Модуль CodecBWT

Листинг Б.17 — **Compressor/Codec/Codecs/BWT/encoderBWT.h**

```
1  /*
2   * File:   codec_BWT.h
3   * Author: art
4   *
5   * Created on 23 Январь 2011 г., 12:47
6   */
7
8  #ifndef CODEC_BWT_H
9  #    define CODEC_BWT_H
10
11
12 #    include "divsuf/divsufsort.h"
13 #    include "../../../DataBlock/dataBlock.h"
14 #    include "../../codecAbstract.h"
15
16 class Codec_BWT : public virtual Codec_abstract
17 {
18 public:
19     Codec_BWT( );
20     ~ Codec_BWT( );
```

```
21
22      void        encode_BWT( DataBlock* inData );
23      void        decode_BWT( DataBlock* inData );
24  private:
25  } ;
26
27
28  #endif  /* CODEC_BWT_H */
```

Листинг Б.18 — **Compressor/Codec/Codecs/BWT/encoderBWT.cpp**

```
1   /*
2    * File:   codec_BWT.cpp
3    * Author: art
4    *
5    * Created on 23 Январь 2011 г., 12:47
6    */
7
8   #include "codecBWT.h"
9
10  Codec_BWT::Codec_BWT( ) { }
11
12  Codec_BWT::~Codec_BWT( ) { }
13
14  void
15  Codec_BWT::encode_BWT( DataBlock* inData )
16  {
17      initEncoder( inData );
18
19      unsigned int * index_table = new unsigned int[decodedDataSize];
20
21      encodedDataSize = decodedDataSize;
22
23      dataT buffer( encodedDataSize );
24
25      int32_t *SA = new int32_t [decodedDataSize];
26
27      codecParams = divbwt( data, buffer.data( ), SA, decodedDataSize );
28
29      inData->setData( buffer.data( ), buffer.size( ) );
30
31      recordOutHeader( inData->getHeader( ), BWT_ID );
32
33      delete[] SA;
34      delete[] index_table;
35  }
36
37  void
38  Codec_BWT::decode_BWT( DataBlock* inData )
39  {
40      initDecoder( inData );
41
42      unsigned int * index_table = new unsigned int [encodedDataSize];
43
44      int32_t *A = new int32_t [encodedDataSize];
45      inverse_bw_transform( data, data, A, encodedDataSize, codecParams );
46      delete[] A;
47
```

```
48      inData->setBlock( data );
49
50      delete[] index_table;
51  }
```

Б.10 Модуль CodecHUFF

Листинг Б.19 — **Compressor/Codec/Codecs/HUFF/codecHUFF.h**

```
1  /*
2   * File:   codec_HUFF.h
3   * Author: art
4   *
5   * Created on 24 Январь 2011 г., 20:32
6   */
7
8  #ifndef CODEC_HUFF_H
9  #    define CODEC_HUFF_H
10
11 #    include "../../../DataBlock/dataBlock.h"
12 #    include "../../codecAbstract.h"
13
14 class Codec_HUFF : public virtual Codec_abstract
15 {
16 public:
17     Codec_HUFF( );
18     virtual ~Codec_HUFF( );
19
20     void         decode_HUFF( DataBlock* inData );
21     void         encode_HUFF( DataBlock* inData );
22 private:
23
24 } ;
25
26 #endif  /* CODEC_HUFF_H */
```

Листинг Б.20 — **Compressor/Codec/Codecs/HUFF/codecHUFF.cpp**

```
1  /*
2   * File:   codec_HUFF.cpp
3   * Author: art
4   *
5   * Created on 24 Январь 2011 г., 20:32
6   */
7
8  #include "codecHUFF.h"
9  #include "shclib.h"
10
11 Codec_HUFF::Codec_HUFF()
12 {
13 }
14
15 Codec_HUFF::~Codec_HUFF()
16 {
17 }
18
```

```
19 void
20 Codec_HUFF::decode_HUFF( DataBlock* inData )
21 {
22     initDecoder( inData );
23
24     dataT buffer( decodedDataSize );
25
26     sh_DecodeBlock( data, buffer.data(), encodedDataSize );
27
28     inData->setBlock( buffer.data() );
29 }
30
31 void
32 Codec_HUFF::encode_HUFF( DataBlock* inData )
33 {
34     initEncoder( inData );
35     dataT buffer( decodedDataSize + 256 );
36
37     encodedDataSize = sh_EncodeBlock( data, buffer.data( ), decodedDataSize );
38
39     inData->setData( buffer.data( ), encodedDataSize );
40     recordOutHeader( inData->getHeader( ), HUFF_ID );
41 }
```

Б.11 Модуль CodecMTF

Листинг Б.21 — **Compressor/Codec/Codecs/MTF/codecMTF.h**

```
1  /*
2   * File:   codec_MTF.h
3   * Author: art
4   *
5   * Created on 23 Январь 2011 г., 20:50
6   */
7
8  #ifndef CODEC_MTF_H
9  #    define CODEC_MTF_H
10
11 #    include "../../../DataBlock/dataBlock.h"
12 #    include "../../codecAbstract.h"
13
14 class Codec_MTF : public virtual Codec_abstract
15 {
16 public:
17     Codec_MTF( );
18     virtual ~Codec_MTF( );
19
20     void         decode_MTF( DataBlock* inData );
21     void         encode_MTF( DataBlock* inData );
22 private:
23     int          mtf( int c );
24     int          get_mtf_c( int i );
25
26     typedef struct  mtf_list
27     {
28         int c;
```

```
29          struct mtf_list *prev;
30          struct mtf_list *next;
31      } mtf_list_t;
32
33      mtf_list_t  *p, *head, *tail, *table;
34
35      void        init_mtf( int tsize );
36
37  } ;
38
39  inline int
40  Codec_MTF::mtf( int c )
41  {
42      int i = 0;
43
44      /* find c. */
45      p = head;
46      while ( p->c != c )
47      {
48          ++ i;
49          p = p->next;
50      }
51      /* move-to-front. */
52      if ( p->prev )
53      {
54          if ( p->next )
55          {
56              p->prev->next = p->next;
57              p->next->prev = p->prev;
58          }
59          else
60          {
61              p->prev->next = NULL;
62              tail = p->prev;
63          }
64          p->prev = NULL;
65          p->next = head;
66          head->prev = p;
67          head = p;
68      } /* front, don't MTF! */
69
70      return i;
71  }
72
73  inline int
74  Codec_MTF::get_mtf_c( int i )
75  {
76      /* find c. */
77      p = head;
78      while ( i -- )
79      {
80          p = p->next;
81      }
82      /* move-to-front. */
83      if ( p->prev )
84      {
85          if ( p->next )
86          {
87              p->prev->next = p->next;
```

```
88              p->next->prev = p->prev;
89          }
90          else
91          {
92              p->prev->next = NULL;
93              tail = p->prev;
94          }
95          p->prev = NULL;
96          p->next = head;
97          head->prev = p;
98          head = p;
99      }
100     return p->c;
101 }
102
103 #endif  /* CODEC_MTF_H */
```

Листинг Б.22 — **Compressor/Codec/Codecs/MTF/codecMTF.cpp**

```
1   /*
2    * File:   codec_MTF.cpp
3    * Author: art
4    *
5    * Created on 23 Январь 2011 г., 20:50
6    */
7
8   #include "codecMTF.h"
9
10  Codec_MTF::Codec_MTF( ) { }
11
12  Codec_MTF::~Codec_MTF( ) { }
13
14  void
15  Codec_MTF::decode_MTF( DataBlock* inData )
16  {
17      initDecoder( inData );
18
19      dataT buffer;
20      buffer.reserve( decodedDataSize );
21
22      init_mtf( 256 );
23
24      for ( unsigned int i = 0; i < encodedDataSize; ++i )
25          buffer.push_back( get_mtf_c( data[i] ) );
26
27      inData->setBlock( buffer.data( ) );
28
29      delete[] table;
30  }
31
32  void
33  Codec_MTF::encode_MTF( DataBlock* inData )
34  {
35      initEncoder( inData );
36
37      encodedDataSize = decodedDataSize;
38
39      dataT buffer;
```

```
40      buffer.reserve( encodedDataSize );
41
42      init_mtf( 256 );
43
44
45      for ( unsigned int i = 0; i < decodedDataSize; ++i )
46          buffer.push_back( mtf( data[i] ) );
47
48      delete[] table;
49
50      inData->setData( buffer.data( ), buffer.size( ) );
51
52      recordOutHeader( inData->getHeader( ), MTF_ID );
53  }
54
55  void
56  Codec_MTF::init_mtf( int tsize )
57  {
58      p =
59          head =
60          tail =
61          table = NULL;
62
63      table = new mtf_list_t[tsize];
64      /* initialize the list. */
65      for ( int i = tsize; i--> 0; )
66      {
67          table[i].c      = i;
68          table[i].next   = &table[i - 1];
69          table[i].prev   = &table[i + 1];
70      }
71      table[tsize - 1].prev = NULL;
72      table[0].next = NULL;
73      head = &table[tsize - 1];
74      tail = &table[0];
75  }
```

Б.12 Модуль CodecRLE

Листинг Б.23 — **Compressor/Codec/Codecs/RLE/codecRLE.h**

```
1   /*
2    * File:   Codec_RLE.h
3    * Author: art
4    *
5    * Created on 31 Январь 2011 г., 14:24
6    */
7
8   #ifndef CODEC_RLE_H
9   #    define CODEC_RLE_H
10
11  #    include "../../../DataBlock/dataBlock.h"
12  #    include "../../codecAbstract.h"
13
14  class Codec_RLE : public virtual Codec_abstract
15  {
```

```
16 public:
17     Codec_RLE( );
18     virtual ~ Codec_RLE( );
19
20     void         decode_RLE( DataBlock* inData );
21     void         encode_RLE( DataBlock* inData );
22 private:
23
24 } ;
25
26 #endif  /* CODEC_RLE_H */
```

Листинг Б.24 — **Compressor/Codec/Codecs/RLE/codecRLE.cpp**

```
1  /*
2   * File:   Codec_RLE.cpp
3   * Author: art
4   *
5   * Created on 31 Январь 2011 г., 14:24
6   */
7
8  #include "codecRLE.h"
9
10 Codec_RLE::Codec_RLE( ) { }
11
12 Codec_RLE::~Codec_RLE( ) { }
13
14 void
15 Codec_RLE::decode_RLE( DataBlock* inData )
16 {
17     initDecoder( inData );
18
19     dataT buffer;
20     buffer.reserve( decodedDataSize );
21
22     int currChar, prevChar;
23     prevChar = 0xFFFFFF;
24
25     unsigned char count;
26
27     unsigned int i = 0;
28     while ( i < encodedDataSize )
29     {
30         currChar = data[i++];
31
32         buffer.push_back( currChar );
33
34         /* check for run */
35         if ( currChar == prevChar )
36         {
37             count = data[i++];
38             while ( count-- > 0 )
39             {
40                 buffer.push_back( currChar );
41             }
42
43             prevChar = 0xFFFFFF;
44         }
```

```
45          else
46          {
47              /* no run */
48              prevChar = currChar;
49          }
50      }
51
52      inData->setBlock( buffer.data( ) );
53  }
54
55  void
56  Codec_RLE::encode_RLE( DataBlock* inData )
57  {
58      initEncoder( inData );
59
60      /*
61       * TODO:
62       * RLE overhead?!
63       */
64
65      encodedDataSize = decodedDataSize * 2; //<----
66
67      dataT buffer;
68      buffer.reserve( encodedDataSize );
69
70      int currChar = EOF, prevChar = EOF; /* current and previous characters */
71      unsigned char count;
72      count = 0;
73
74      unsigned int index = 0;
75
76      while ( index != decodedDataSize )
77      {
78          currChar = data[index++];
79          buffer.push_back( currChar );
80
81          /* check for run */
82          if ( currChar == prevChar )
83          {
84              /* we have a run.  count run length */
85              count = 0;
86
87              while ( index != decodedDataSize )
88              {
89                  currChar = data[index++];
90                  if ( currChar == prevChar )
91                  {
92                      ++count;
93
94                      if ( count == UCHAR_MAX )
95                      {
96                          /* count is as long as it can get */
97                          buffer.push_back( count );
98
99                          /* force next char to be different */
100                         prevChar = EOF;
101                         break;
102                     }
103                 }
```

```
104                 else
105                 {
106                     /* run ended */
107                     buffer.push_back( count );
108                     buffer.push_back( currChar );
109 
110                     prevChar = currChar;
111                     break;
112                 }
113                 if ( index == decodedDataSize )
114                 {
115                     buffer.push_back( count );
116                     break;
117                 }
118             }
119         }
120         else
121         {
122             /* no run */
123             prevChar = currChar;
124         }
125     }
126     encodedDataSize = buffer.size( );
127     inData->setData( buffer.data( ), buffer.size( ) );
128 
129     recordOutHeader( inData->getHeader( ), RLE_ID );
130 }
```

Б.13 Модуль Compressor

Листинг Б.25 — **Compressor/Compressor.h**

```
1  /*
2   * File:   Compressor.h
3   * Author: art
4   *
5   * Created on 5 Февраль 2011 г., 20:56
6   */
7 
8  #ifndef COMPRESSOR_H
9  #    define COMPRESSOR_H
10 
11 #    include <list>
12 #    include "./Codec/codec.h"
13 #    include "FilesTable/filesTable.h"
14 #    include "DataBlock/dataBlockHeader.h"
15 
16 class CompressorStatus
17 {
18 public:
19 
20     CompressorStatus( ) : runStatus( RUN ) { };
21 
22     enum RunStatus
23     {
24         RUN, STOP
```

```
25      } ;
26
27      enum ErrorCode
28      {
29          SUCCESS, INPUT_FILE_OPEN_ERROR, OUTPUT_FILE_WRITE_ERROR, INPUT_FILE_CORRUPTED,
                INPUT_FILE_UNCORRUPTED, DECOMPRESS_FAIL, PROCEED, CANCELLED
30      } ;
31
32      void setRunStatus( RunStatus status )
33      {
34          runStatus = status;
35      }
36
37      RunStatus getRunStatus( ) const
38      {
39          return runStatus;
40      }
41
42      virtual void showProgress( float progress, const QString &fileName, float speed );
43      virtual void showInfo( ErrorCode errorCode, const QString &fileName, unsigned int id );
44  private:
45      RunStatus runStatus;
46  } ;
47
48  class Compressor
49  {
50  public:
51
52      Compressor( CompressorStatus *status  );
53      virtual
54      ~ Compressor( );
55
56      struct Stat
57      {
58          off_t decodedSize;
59          off_t encodedSize;
60          float speed; //enc-/dec- speed (mb/s)
61          vector <string> brokenFilesNames;
62      } ;
63
64      enum CoderTypes
65      {
66          NONE, RLE, BWT, MTF, HUFFMAN
67      } ;
68
69
70      /**
71       * Compress multiply files
72       * @param iFileNames     array with input file names
73       * @param iFilesCount    number of input files
74       * @param oFileName      output file name
75       * @param blocksize      block size
76       * @return               error code
77       */
78      CompressorStatus::ErrorCode      compress( const QStringList  &iFileNames, unsigned int
            iFilesCount, const QString &oFileName,
79                                                unsigned int blocksize, const list<
                                                      CoderTypes > *compressSequence = NULL );
80      /**
```

```
81          * Decompress archive
82          * @param iFileName      archive filename
83          * @param keepBroken     keep broken files
84          * @return               error code
85          */
86      CompressorStatus::ErrorCode     decompress( const QString &iFileName, bool keepBroken =
            false );
87      /**
88          * Get statistical info
89          * @param stat  point to info structure
90          */
91      void                            getStat( Stat *stat );
92
93      /**
94          * List archive contents
95          * @param iFileName archive filename
96          * @param contents  point to output vector with archive contents
97          * @return          error code
98          */
99      CompressorStatus::ErrorCode     listArchiveContents( const QString &iFileName, vector<
            FilesTable::FileInfo> * const contents = NULL );
100
101 private:
102     Codec                           codec;
103     DataBlock                       *block;
104     CompressorStatus                *status;
105     off_t                           currReadBytesCount;
106     off_t                           decodedDataSize;
107     off_t                           encodedDataSize;
108     uint32_t                        totalProcessingTime;
109     unsigned int                    source_data_size;
110     FilesTable                      blocksTable;
111     CompressorStatus::ErrorCode     error;
112     vector<FilesTable::FileInfo>    contents;
113     list< CoderTypes >              defaultCompressSequence;
114
115     void                            showEncodingProgress( const QString &currFileName, float
            speed );
116     void                            showDecodingProgress( const char * currFileName, float
            speed );
117     void                            showInfo( CompressorStatus::ErrorCode errorCode, const
            QString &currFileName = "", unsigned int id = 0 );
118     void                            showInfo( CompressorStatus::ErrorCode errorCode, const
            char * currFileName, unsigned int id = 0 );
119
120     void                            compress( const list< CoderTypes > *compressSequence =
            NULL );
121     bool                            decompress( );
122
123     bool                            createEmptyFile( const char * fileName );
124     void                            removeBrokenFiles( );
125     float                           speed( unsigned int nBytes, clock_t elapsedTime );
126
127 };
128
129 class Encoder
130 {
131 public:
132
```

```
133     Encoder( DataBlock * block, Codec *codec )
134     {
135         this->block = block;
136         this->codec = codec;
137     }
138 
139     virtual ~ Encoder( ) { };
140 
141     void operator( )( Compressor::CoderTypes coderType )
142     {
143         switch ( coderType )
144         {
145             case Compressor::BWT: codec->encode_BWT( block );
146                 break;
147             case Compressor::RLE: codec->encode_RLE( block );
148                 break;
149             case Compressor::MTF: codec->encode_MTF( block );
150                 break;
151             case Compressor::HUFFMAN: codec->encode_HUFF( block );
152                 break;
153             default:
154                 break;
155         }
156     }
157 private:
158 
159     DataBlock       *block;
160     Codec           *codec;
161 
162 } ;
163 
164 void inline     Compressor::compress( const list< CoderTypes > *compressSequence )
165 {
166     Encoder encoder( block, &codec );
167     const list< CoderTypes > *sequence = compressSequence ? compressSequence : &
        defaultCompressSequence;
168     std::for_each( sequence->begin( ), sequence->end( ), encoder );
169 }
170 
171 bool inline     Compressor::decompress( )
172 {
173     uint64_t id;
174     while ( ( id = block->getHeader( )->getId( ) ) != RAW_ID )
175     {
176         switch ( id )
177         {
178             case HUFF_ID: codec.decode_HUFF( block );
179                 break;
180             case RLE_ID: codec.decode_RLE( block );
181                 break;
182             case BWT_ID: codec.decode_BWT( block );
183                 break;
184             case MTF_ID: codec.decode_MTF( block );
185                 break;
186             default:return - 1;
187         }
188     }
189     return 0;
190 }
```

```
191
192 void inline         Compressor::showDecodingProgress( const char * currFileName, float speed
        )
193 {
194     if (status)
195         status->showProgress( ( encodedDataSize ) ? ( float ) currReadBytesCount /
              encodedDataSize : 0, QString::fromUtf8( currFileName, strlen( currFileName ) ),
              speed );
196 }
197
198 void inline         Compressor::showEncodingProgress( const QString &currFileName, const
      float speed )
199 {
200     if (status)
201         status->showProgress( ( decodedDataSize ) ? ( float ) currReadBytesCount /
              decodedDataSize : 0, currFileName, speed );
202 }
203
204 float inline        Compressor::speed( unsigned int nBytes, clock_t elapsedTime )
205 {
206     return ( elapsedTime ) ? nBytes / ( ( double ) elapsedTime / CLOCKS_PER_SEC ) : 0;
207 }
208
209 void inline         Compressor::showInfo( CompressorStatus::ErrorCode errorCode, const
      QString &currFileName , unsigned int id )
210 {
211     if (status)
212         status->showInfo( errorCode, currFileName, id );
213 }
214
215 void inline         Compressor::showInfo( CompressorStatus::ErrorCode errorCode, const char
      * currFileName , unsigned int id )
216 {
217     QString fileName;
218     if (! currFileName)
219         fileName = "";
220     else
221         fileName = QString::fromUtf8( currFileName, strlen( currFileName ) );
222     if (status)
223         status->showInfo( errorCode, fileName, id );
224 }
225
226
227 #endif  /* COMPRESSOR_H */
```

Листинг Б.26 — **Compressor/Compressor.cpp**

```
1  /*
2   * File:   Compressor.cpp
3   * Author: art
4   *
5   * Created on 5 Февраль 2011 г., 20:56
6   */
7
8  #include <sys/types.h>
9  #include <sys/stat.h>
10 #include <math.h>
11 #include <time.h>
```

```
12 #include <QFile>
13 #include <QStringList>
14 #include "compressor.h"
15
16 void
17 CompressorStatus::showProgress( float, const QString&, float ) { }
18
19 void
20 CompressorStatus::showInfo( ErrorCode, const QString&, unsigned int ) { }
21
22 Compressor::Compressor( CompressorStatus *_status ) : codec( ), block( NULL ), status(
       _status ), currReadBytesCount( 0 ), decodedDataSize( 0 ), encodedDataSize( 0 ),
       totalProcessingTime( 0 ),
23     source_data_size( 0 ), blocksTable( ), defaultCompressSequence( )
24 {
25     defaultCompressSequence.push_back( RLE );
26     defaultCompressSequence.push_back( BWT );
27     defaultCompressSequence.push_back( MTF );
28     defaultCompressSequence.push_back( RLE );
29     defaultCompressSequence.push_back( BWT );
30     defaultCompressSequence.push_back( RLE );
31     defaultCompressSequence.push_back( HUFFMAN );
32 }
33
34 Compressor::~Compressor( ) { }
35
36 /*
37  * Encoding
38  */
39
40 bool
41 Compressor::createEmptyFile( const char * fileName )
42 {
43     QFile fout( QString::fromUtf8( fileName, strlen( fileName ) ) );
44     if ( !fout.open( QIODevice::Truncate | QIODevice::WriteOnly ) )
45         return false;
46     fout.close( );
47     return true;
48 }
49
50 CompressorStatus::ErrorCode
51 Compressor::compress( const QStringList &iFileNames, unsigned int iFilesCount, const QString
       &oFileName, unsigned int blocksize, const list< CoderTypes > *compressSequence )
52 {
53     totalProcessingTime = 0;
54     decodedDataSize = 0;
55     currReadBytesCount = 0;
56     encodedDataSize = 0;
57     error = CompressorStatus::SUCCESS;
58
59     vector<int>proceedFiles;
60
61     for ( unsigned int i = 0; i < iFilesCount; ++i )
62     {
63         QFile fin( iFileNames[i] );
64         if ( !fin.open( QIODevice::ReadOnly ) )
65         {
66             showInfo( CompressorStatus::INPUT_FILE_OPEN_ERROR, iFileNames[i], i + 1 );
67             error = CompressorStatus::INPUT_FILE_OPEN_ERROR;
```

```
68              continue;
69          }
70          fin.close( );
71
72          proceedFiles.push_back( i );
73          decodedDataSize += fin.size( );
74      }
75
76      block = new DataBlock( blocksize );
77      DataBlockHeader * header = block->getHeader( );
78
79      QFile fout( oFileName );
80      if ( !fout.open( QIODevice::WriteOnly ) )
81      {
82          showInfo( CompressorStatus::OUTPUT_FILE_WRITE_ERROR );
83          return error = CompressorStatus::OUTPUT_FILE_WRITE_ERROR;
84      }
85
86      bool stop = false;
87      for ( vector<int>::iterator i = proceedFiles.begin( ); i != proceedFiles.end( ); ++i )
88      {
89          showInfo( CompressorStatus::PROCEED, iFileNames[*i], *i + 1 );
90
91          QFile fin( iFileNames[*i] );
92          if ( !fin.open( QIODevice::ReadOnly ) )
93          {
94              fout.close( );
95              showInfo( CompressorStatus::INPUT_FILE_OPEN_ERROR, iFileNames[*i], *i + 1 );
96              error = CompressorStatus::INPUT_FILE_OPEN_ERROR;
97              continue;
98          }
99          uint32_t partsCount = ceil( ( float ) fin.size( ) / blocksize );
100         uint32_t part = 0;
101         while ( 1 )
102         {
103             if ( status )
104             {
105                 if ( status->getRunStatus( ) == CompressorStatus::STOP )
106                 {
107                     stop = true;
108                     break;
109                 }
110             }
111
112             //compress current file
113             if ( !block->readRAW( fin ) )
114                 break;
115
116             source_data_size = header->getDecodedDataSize( );
117             currReadBytesCount += source_data_size;
118
119             header->setFileName( iFileNames[*i].toUtf8( ).constData( ) );
120             header->setPart( part++ );
121             header->setPartsCount( partsCount );
122
123             clock_t startTime = clock( );
124
125             compress( compressSequence );
126
```

```
127             clock_t stoptTime = clock( );
128             totalProcessingTime += stoptTime - startTime;
129
130             header->setFileName( iFileNames[*i].toUtf8( ).constData( ) );
131             encodedDataSize += header->getEncodedDataSize( );
132             header->setRAWDataSize( source_data_size );
133             header->setPart( part - 1 );
134             header->setPartsCount( partsCount );
135             block->write( fout );
136
137             showEncodingProgress( iFileNames[*i], speed( source_data_size, stoptTime -
                    startTime ) );
138         }
139         fin.close( );
140         if ( !stop )
141             showInfo( CompressorStatus::SUCCESS, iFileNames[*i], *i + 1 );
142         else
143         {
144             do
145             {
146                 showInfo( CompressorStatus::CANCELLED, iFileNames[*i], *i + 1 );
147             }
148             while ( ++i != proceedFiles.end( ) );
149             break;
150         }
151     }
152     delete block;
153     return error;
154 }
155
156 /*
157  * Decoding
158  */
159
160 CompressorStatus::ErrorCode
161 Compressor::decompress( const QString &iFileName, bool keepBroken )
162 {
163     blocksTable.clean( );
164     totalProcessingTime = 0;
165     decodedDataSize = 0;
166     currReadBytesCount = 0;
167     encodedDataSize = 0;
168     unsigned int nextId = 1;
169     error = CompressorStatus::SUCCESS;
170
171     QFile fin( iFileName );
172     if ( !fin.open( QIODevice::ReadOnly ) )
173     {
174         showInfo( CompressorStatus::INPUT_FILE_OPEN_ERROR );
175         return error = CompressorStatus::INPUT_FILE_OPEN_ERROR;
176     }
177
178     block = new DataBlock( );
179     DataBlockHeader * header = block->getHeader( );
180
181     encodedDataSize = fin.size( );
182
183     bool stop = false;
184     while ( !stop )
```

```
185     {
186         switch ( block->read( fin ) )
187         {
188             case FILE_END:
189                 stop = true;
190                 continue;
191 
192             case FILE_BROKEN:
193             {
194                 error = CompressorStatus::INPUT_FILE_CORRUPTED;
195                 unsigned id = blocksTable.getId( header );
196                 id = id ? id : nextId++;
197                 showInfo( CompressorStatus::INPUT_FILE_CORRUPTED, header->getFileName( ), id
                        );
198                 continue;
199             }
200             case HEADER_CORRUPTED:
201             {
202                 error = CompressorStatus::INPUT_FILE_CORRUPTED;
203                 showInfo( CompressorStatus::INPUT_FILE_CORRUPTED );
204                 continue;
205             }
206             case FILE_TOO_SMALL:
207             {
208                 if ( keepBroken == false )
209                     removeBrokenFiles( );
210                 showInfo( CompressorStatus::INPUT_FILE_CORRUPTED );
211                 return error = CompressorStatus::INPUT_FILE_CORRUPTED;
212             }
213             default: break;
214         }
215 
216         currReadBytesCount = fin.pos( );
217 
218         clock_t startTime = clock( );
219 
220         decompress( );
221 
222         clock_t stoptTime = clock( );
223         totalProcessingTime += stoptTime - startTime;
224 
225         decodedDataSize += header->getDecodedDataSize( );
226         showDecodingProgress( header->getFileName( ), speed( header->getEncodedDataSize( ),
                stoptTime - startTime ) );
227 
228         if ( block->checkCRC( ) )
229         {
230             delete block;
231             fin.close( );
232             showInfo( CompressorStatus::DECOMPRESS_FAIL );
233             return error = CompressorStatus::DECOMPRESS_FAIL;
234         }
235 
236         switch ( blocksTable.add( header, nextId ) )
237         {
238             case FIRST_AND_LAST_RECIEVED_BLOCK:
239                 blocksTable.remove( header );
240                 if ( !createEmptyFile( header->getFileName( ) ) )
241                 {
```

```
242                     showInfo( CompressorStatus::OUTPUT_FILE_WRITE_ERROR );
243                     return error = CompressorStatus::OUTPUT_FILE_WRITE_ERROR;
244                 }
245                 showInfo( CompressorStatus::SUCCESS, header->getFileName( ), nextId++ );
246                 break;
247             case FIRST_RECIEVED_BLOCK:
248                 createEmptyFile( header->getFileName( ) );
249                 showInfo( CompressorStatus::PROCEED, header->getFileName( ), nextId++ );
250                 break;
251             case ALL_BLOCKS_RECIEVED:
252                 showInfo( CompressorStatus::SUCCESS, header->getFileName( ), blocksTable.
                        getId( header ) );
253                 blocksTable.remove( header );
254                 break;
255             case BLOCK_OUT_OF_RANGE:
256             case BLOCK_ALREADY_RECIEVED: showInfo( CompressorStatus::INPUT_FILE_CORRUPTED );
257                 return error = CompressorStatus::INPUT_FILE_CORRUPTED;
258                 break;
259             default:break;
260         }
261
262         if ( block->writeRAW( ) )
263         {
264             delete block;
265             fin.close( );
266             showInfo( CompressorStatus::OUTPUT_FILE_WRITE_ERROR );
267             return error = CompressorStatus::OUTPUT_FILE_WRITE_ERROR;
268         }
269
270         if ( status->getRunStatus( ) == CompressorStatus::STOP )
271         {
272             stop = true;
273             showInfo( CompressorStatus::CANCELLED, header->getFileName( ), blocksTable.getId
                    ( header ) );
274         }
275     }
276
277     //Check for broken files
278
279     vector<string> * brokenFilesNames = blocksTable.getNonCompleteFilesNames( );
280
281     if ( brokenFilesNames )
282         for ( vector<string>::iterator i = brokenFilesNames->begin( ); i != brokenFilesNames
                ->end( ); ++i )
283         {
284             showInfo( CompressorStatus::INPUT_FILE_CORRUPTED, i->c_str( ), blocksTable.getId
                    ( *i ) );
285             if ( keepBroken == false )
286                 removeBrokenFiles( );
287         }
288
289     fin.close( );
290     delete block;
291     return error;
292 }
293
294 void
295 Compressor::getStat( Stat *stat )
296 {
```

```
297     vector<string> *nonCompleteFilesNames = blocksTable.getNonCompleteFilesNames( );
298
299     if ( nonCompleteFilesNames )
300         stat->brokenFilesNames = *nonCompleteFilesNames;
301     stat->decodedSize = decodedDataSize;
302     stat->encodedSize = encodedDataSize;
303     stat->speed = speed( decodedDataSize, totalProcessingTime );
304 }
305
306 void
307 Compressor::removeBrokenFiles( )
308 {
309     vector<string> *nonCompleteFilesNames = blocksTable.getNonCompleteFilesNames( );
310
311     if ( nonCompleteFilesNames )
312     {
313         for ( unsigned int i = 0; i < nonCompleteFilesNames->size( ); ++i )
314             QFile::remove( QString::fromUtf8( ( *nonCompleteFilesNames )[i].c_str( ), strlen
                    ( ( *nonCompleteFilesNames )[i].c_str( ) ) ) );
315     }
316 }
317
318 CompressorStatus::ErrorCode
319 Compressor::listArchiveContents( const QString &iFileName, vector<FilesTable::FileInfo> *
        const contents )
320 {
321     blocksTable.clean( );
322     unsigned int nextId = 1;
323
324     if ( contents )
325     {
326         contents->clear( );
327         contents->reserve( 1000 );
328     }
329
330     totalProcessingTime = 0;
331     decodedDataSize = 0;
332     currReadBytesCount = 0;
333     encodedDataSize = 0;
334     error = CompressorStatus::SUCCESS;
335
336     QFile fin( iFileName );
337     if ( !fin.open( QIODevice::ReadOnly ) )
338     {
339         error = CompressorStatus::INPUT_FILE_OPEN_ERROR;
340         showInfo( error );
341         return error;
342     }
343
344     block = new DataBlock( );
345     DataBlockHeader * header = block->getHeader( );
346     encodedDataSize = fin.size( );
347
348     bool stop = false;
349     while ( !stop )
350     {
351         if ( status )
352         {
353             if ( status->getRunStatus( ) == CompressorStatus::STOP )
```

```
354             {
355                 stop = true;
356             }
357         }
358 
359         clock_t startTime = clock( );
360         switch ( block->read( fin ) )
361         {
362             case FILE_END:
363                 stop = true;
364                 continue;
365 
366             case FILE_BROKEN:
367             {
368                 error = CompressorStatus::INPUT_FILE_CORRUPTED;
369 
370                 unsigned id = blocksTable.getId( header );
371                 id = id ? id : nextId++;
372                 showInfo( error, header->getFileName( ), id );
373 
374                 if ( contents )
375                 {
376                     //Add to archive contents broken 1-block-size file's info
377                     FilesTable::FileInfo fileInfo;
378                     fileInfo.corrupted = true;
379                     fileInfo.fileName = header->getFileName( );
380                     contents->push_back( fileInfo );
381                 }
382                 continue;
383             }
384             case HEADER_CORRUPTED:
385             {
386                 error = CompressorStatus::INPUT_FILE_CORRUPTED;
387                 showInfo( error );
388                 continue;
389             }
390             case FILE_TOO_SMALL:
391             {
392                 error = CompressorStatus::INPUT_FILE_CORRUPTED;
393                 showInfo( error );
394                 delete block;
395                 return error;
396             }
397             default: break;
398         }
399 
400         currReadBytesCount += header->getEncodedDataSize( );
401 
402         clock_t stoptTime = clock( );
403         totalProcessingTime += stoptTime - startTime;
404 
405         showDecodingProgress( header->getFileName( ), speed( header->getEncodedDataSize( ),
                stoptTime - startTime ) );
406 
407         switch ( blocksTable.add( header, nextId ) )
408         {
409             case FIRST_AND_LAST_RECIEVED_BLOCK:
410                 blocksTable.remove( header );
```

```
411                 showInfo( CompressorStatus::INPUT_FILE_UNCORRUPTED, header->getFileName( ),
                        nextId++ );
412                 break;
413             case FIRST_RECIEVED_BLOCK:
414                 showInfo( CompressorStatus::PROCEED, header->getFileName( ), nextId++ );
415                 break;
416             case ALL_BLOCKS_RECIEVED:
417                 showInfo( CompressorStatus::INPUT_FILE_UNCORRUPTED, header->getFileName( ),
                        blocksTable.getId( header ) );
418                 blocksTable.remove( header );
419                 break;
420             case BLOCK_OUT_OF_RANGE:
421             case BLOCK_ALREADY_RECIEVED: error = CompressorStatus::INPUT_FILE_CORRUPTED;
422                 showInfo( error );
423                 delete block;
424                 return error;
425                 break;
426             default:break;
427         }
428     }
429     fin.close( );
430     delete block;
431
432         vector<string> * brokenFilesNames = blocksTable.getNonCompleteFilesNames( );
433
434         if ( brokenFilesNames )
435             for ( vector<string>::iterator i = brokenFilesNames->begin( ); i !=
                    brokenFilesNames->end( ); ++i )
436             {
437                 showInfo( CompressorStatus::INPUT_FILE_CORRUPTED, i->c_str( ), blocksTable.
                        getId( *i ) );
438             }
439
440         if ( contents )
441         {
442             vector<FilesTable::FileInfo> *partialArchiveContents = blocksTable.
                    getArchiveContent( );
443             contents->insert( contents->end( ), partialArchiveContents->begin( ),
                    partialArchiveContents->end( ) );
444         }
445     return error;
446 }
```

Б.14 Модуль CompressorThread

Листинг Б.27 — **GUI/CompressorThread/CompressorThread.h**

```
1  /*
2   * File:   CompressorThread.h
3   * Author: art
4   *
5   * Created on 17 Марш 2011 г., 16:41
6   */
7
8  #ifndef COMPRESSORTHREAD_H
9  #    define COMPRESSORTHREAD_H
```

```
10
11 #    include <QThread>
12 #    include <QDir>
13 #    include <QVector>
14
15 #    include "Compressor/compressor.h"
16 #    include "Compressor/FilesTable/filesTable.h"
17
18 class QProgressDialog;
19 class CompressorThread;
20
21 class CTCompressorStatus : public CompressorStatus
22 {
23 public:
24
25     CTCompressorStatus( );
26     void showProgress( float progress, const QString&, float speed );
27     void showInfo( CompressorStatus::ErrorCode errorCode, const QString &currFileName = "",
           unsigned int id = 0 );
28     void setCompressorThread( CompressorThread *comprThread  );
29
30 private:
31
32     CompressorThread *comprThread;
33 } ;
34
35 class CompressorThread : public QThread
36 {
37     Q_OBJECT
38 public:
39     CompressorThread( );
40     virtual ~ CompressorThread( );
41
42     void run( );
43     void showProgress( float progress, const QString &fileName, float speed );
44     void showInfo( CompressorStatus::ErrorCode errorCode, const QString &fileName, unsigned
           int id );
45     void initCompress( const QStringList &iFileNames, const QString &destFileName, unsigned
           int blocksize, const QDir &compressBaseDir, const QList< Compressor::CoderTypes> &
           compressSequence );
46     void initDecompress( const QString &iFileName, bool keepBroken = false );
47     void initList( const QString &iFileName );
48     void compress( );
49     void decompress( );
50     void list( );
51
52 public  slots:
53     void    stop( );
54
55 signals:
56     void progressChanged( int progress, QString fileName, float speed);
57     void info( CTCompressorStatus::ErrorCode error, QString fileName, unsigned int id);
58     void statInfo( Compressor::Stat stat );
59 private:
60
61     QStringList                 iFileNames;
62     QString                     iFileName;
63     QString                     destFileName;
64     unsigned int                blocksize;
```

```
65      QDir                                compressBaseDir;
66      bool                                keepBroken;
67      QList< Compressor::CoderTypes>      compressSequence;
68      CTCompressorStatus *status;
69
70      enum Mode
71      {
72          NONE, COMPRESSING, DECOMPRESSING, LIST
73      } runMode;
74
75  };
76
77  #endif  /* COMPRESSORTHREAD_H */
```

Листинг Б.28 — **GUI/CompressorThread/CompressorThread.cpp**

```
1   /*
2    * File:   CompressorThread.cpp
3    * Author: art
4    *
5    * Created on 17 Марта 2011 г., 16:41
6    */
7
8   #include "compressorThread.h"
9
10  CompressorThread::CompressorThread( )
11  {
12      runMode = NONE;
13  }
14
15  CompressorThread::~CompressorThread( ) { }
16
17  void
18  CompressorThread::showProgress( float progress, const QString &fileName, float speed )
19  {
20      emit progressChanged( progress * 100, fileName, speed );
21  }
22
23  void
24  CompressorThread::compress( )
25  {
26      bool fail;
27
28      status = new CTCompressorStatus;
29      status->setCompressorThread( this );
30      Compressor compressor( status );
31
32      QStringList relIFileNames;
33
34      for ( QStringList::Iterator i = iFileNames.begin( ); i != iFileNames.end( ); ++i )
35          relIFileNames << compressBaseDir.relativeFilePath( *i );
36
37      std::list<Compressor::CoderTypes> sequence = compressSequence.toStdList( );
38      switch ( compressor.compress( relIFileNames, iFileNames.count( ), destFileName,
            blocksize, &sequence ) )
39      {
40          case CompressorStatus::OUTPUT_FILE_WRITE_ERROR: fail = true;
41              break;
```

```
42          default: fail = false;
43              break;
44      }
45
46      Compressor::Stat stat;
47      compressor.getStat( &stat );
48
49      if ( !fail ) emit statInfo( stat );
50
51      delete status;
52  }
53
54  void
55  CompressorThread::decompress( )
56  {
57      bool fail;
58
59      status = new CTCompressorStatus;
60      status->setCompressorThread( this );
61      Compressor compressor( status );
62
63      switch ( compressor.decompress( iFileName, keepBroken ) )
64      {
65          case CompressorStatus::OUTPUT_FILE_WRITE_ERROR:
66          case CompressorStatus::INPUT_FILE_OPEN_ERROR: fail = true;
67              break;
68          default: fail = false;
69              break;
70      }
71
72      Compressor::Stat stat;
73      compressor.getStat( &stat );
74      if ( !fail ) emit statInfo( stat );
75
76      delete status;
77  }
78
79  void
80  CompressorThread::initCompress( const QStringList &iFileNames, const QString &destFileName,
        unsigned int blocksize, const QDir &compressBaseDir, const QList< Compressor::CoderTypes
        > &compressSequence )
81  {
82      this->iFileNames = iFileNames;
83      this->destFileName = destFileName;
84      this->blocksize = blocksize;
85      this->compressBaseDir = compressBaseDir;
86      this->compressSequence = compressSequence;
87
88      runMode = COMPRESSING;
89  }
90
91  void
92  CompressorThread::initDecompress( const QString &iFileName, bool keepBroken )
93  {
94      this->iFileName = iFileName;
95      this->keepBroken = keepBroken;
96
97      runMode = DECOMPRESSING;
98  }
```

```
99
100 void
101 CompressorThread::initList( const QString &iFileName )
102 {
103     this->iFileName = iFileName;
104
105     runMode = LIST;
106 }
107
108 void
109 CompressorThread::list( )
110 {
111     status = new CTCompressorStatus;
112     status->setCompressorThread( this );
113     Compressor compressor( status );
114
115     compressor.listArchiveContents( iFileName );
116
117     delete status;
118 }
119
120 void
121 CompressorThread::run( )
122 {
123     switch ( runMode )
124     {
125         case COMPRESSING: compress( );
126             break;
127         case DECOMPRESSING: decompress( );
128             break;
129         case LIST: list( );
130             break;
131         default:break;
132     }
133 }
134
135 void
136 CompressorThread::showInfo( CompressorStatus::ErrorCode errorCode, const QString &fileName,
        unsigned int id )
137 {
138     emit info( errorCode, fileName, id );
139 }
140
141 void
142 CompressorThread::stop( )
143 {
144     status->setRunStatus( CompressorStatus::STOP );
145 }
146
147 /*
148  * CompressorThreadCompressorStatus
149  */
150
151 CTCompressorStatus::CTCompressorStatus( ) : comprThread( NULL ) { }
152
153 void
154 CTCompressorStatus::setCompressorThread( CompressorThread *comprThread )
155 {
156     this->comprThread = comprThread;
```

```
157 }
158
159 void
160 CTCompressorStatus::showProgress( float progress, const QString &fileName, float speed )
161 {
162     comprThread->showProgress( progress, fileName, speed );
163 }
164
165 void
166 CTCompressorStatus::showInfo( ErrorCode errorCode, const QString &fileName, unsigned int id
        )
167 {
168     comprThread->showInfo( errorCode, fileName, id );
169 }
```

Б.15 Модуль CompressSettingsPanel

Листинг Б.29 — **GUI/SettingsPanels/compressSettingsPanel.h**

```
1  /*
2   * File:   CompressSettingsPanel.h
3   * Author: art
4   *
5   * Created on 20 Март 2011 г., 18:47
6   */
7
8  #ifndef COMPRESSSETTINGSPANEL_H
9  #    define COMPRESSSETTINGSPANEL_H
10
11 #    include <QWidget>
12 #    include "Compressor/compressor.h"
13
14 class QSpinBox;
15 class QComboBox;
16
17 class CompressSettingsPanel : public QWidget
18 {
19     Q_OBJECT
20 public:
21     CompressSettingsPanel( );
22     virtual ~CompressSettingsPanel( );
23 public slots:
24     void        set( unsigned int blockSize, QList< Compressor::CoderTypes> compressSequence
          );
25 signals:
26     void        settingsChanged( unsigned int blockSize, QList< Compressor::CoderTypes>
          compressSequence );
27     void        resetToDefaults( void );
28 private:
29     QSpinBox*   blockSizeSpinBox;
30     QComboBox   *encoder1ComboBox;
31     QComboBox   *encoder2ComboBox;
32     QComboBox   *encoder3ComboBox;
33     QComboBox   *encoder4ComboBox;
34     QComboBox   *encoder5ComboBox;
35     QComboBox   *encoder6ComboBox;
```

```
36      QComboBox    *encoder7ComboBox;
37      QComboBox    *encoder8ComboBox;
38
39      void         setupWidgetsConnections( );
40
41  private slots:
42      void         get( );
43      void         disableEncoders( );
44
45  } ;
46
47  #endif  /* COMPRESSSETTINGSPANEL_H */
```

Листинг Б.30 — **GUI/SettingsPanels/compressSettingsPanel.cpp**

```
1   /*
2    * File:    CompressSettingsPanel.cpp
3    * Author: art
4    *
5    * Created on 20 Марш 2011 г., 18:47
6    */
7
8   #include <QLabel>
9   #include <QSpinBox>
10  #include <QGridLayout>
11  #include <QComboBox>
12  #include <QPushButton>
13  #include "compressSettingsPanel.h"
14
15  CompressSettingsPanel::CompressSettingsPanel( ) : QWidget(  )
16  {
17      QStringList encoderTypes = QStringList( ) << "" << "RLE" << "BWT" << "MTF" << "Huffman";
18
19      QLabel *encoder1Label = new QLabel( tr( "1 Encoder" ) );
20      QLabel *encoder2Label = new QLabel( tr( "2 Encoder" ) );
21      QLabel *encoder3Label = new QLabel( tr( "3 Encoder" ) );
22      QLabel *encoder4Label = new QLabel( tr( "4 Encoder" ) );
23      QLabel *encoder5Label = new QLabel( tr( "5 Encoder" ) );
24      QLabel *encoder6Label = new QLabel( tr( "6 Encoder" ) );
25      QLabel *encoder7Label = new QLabel( tr( "7 Encoder" ) );
26      QLabel *encoder8Label = new QLabel( tr( "8 Encoder" ) );
27
28
29      QHBoxLayout *blockSizeHbox = new QHBoxLayout;
30      QHBoxLayout *encoder1hbox = new QHBoxLayout;
31      QHBoxLayout *encoder2hbox = new QHBoxLayout;
32      QHBoxLayout *encoder3hbox = new QHBoxLayout;
33      QHBoxLayout *encoder4hbox = new QHBoxLayout;
34      QHBoxLayout *encoder5hbox = new QHBoxLayout;
35      QHBoxLayout *encoder6hbox = new QHBoxLayout;
36      QHBoxLayout *encoder7hbox = new QHBoxLayout;
37      QHBoxLayout *encoder8hbox = new QHBoxLayout;
38
39
40      encoder1ComboBox = new QComboBox;
41      encoder2ComboBox = new QComboBox;
42      encoder3ComboBox = new QComboBox;
43      encoder4ComboBox = new QComboBox;
```

```
44      encoder5ComboBox = new QComboBox;
45      encoder6ComboBox = new QComboBox;
46      encoder7ComboBox = new QComboBox;
47      encoder8ComboBox = new QComboBox;
48
49      encoder1ComboBox->insertItems( 0, encoderTypes );
50      encoder2ComboBox->insertItems( 0, encoderTypes );
51      encoder3ComboBox->insertItems( 0, encoderTypes );
52      encoder4ComboBox->insertItems( 0, encoderTypes );
53      encoder5ComboBox->insertItems( 0, encoderTypes );
54      encoder6ComboBox->insertItems( 0, encoderTypes );
55      encoder7ComboBox->insertItems( 0, encoderTypes );
56      encoder8ComboBox->insertItems( 0, encoderTypes );
57
58      QLabel *blockSizeLabel = new QLabel( tr( "Block Size" ) );
59      blockSizeSpinBox = new QSpinBox;
60      blockSizeSpinBox->setRange( 10, 10000 );
61      blockSizeSpinBox->setSingleStep( 100 );
62      blockSizeSpinBox->setSuffix( " kb" );
63      blockSizeHbox->addWidget( blockSizeLabel );
64      blockSizeHbox->addWidget( blockSizeSpinBox );
65
66
67      encoder1hbox->addWidget( encoder1Label );
68      encoder1hbox->addWidget( encoder1ComboBox );
69
70      encoder2hbox->addWidget( encoder2Label );
71      encoder2hbox->addWidget( encoder2ComboBox );
72
73      encoder3hbox->addWidget( encoder3Label );
74      encoder3hbox->addWidget( encoder3ComboBox );
75
76      encoder4hbox->addWidget( encoder4Label );
77      encoder4hbox->addWidget( encoder4ComboBox );
78
79      encoder5hbox->addWidget( encoder5Label );
80      encoder5hbox->addWidget( encoder5ComboBox );
81
82      encoder6hbox->addWidget( encoder6Label );
83      encoder6hbox->addWidget( encoder6ComboBox );
84
85      encoder7hbox->addWidget( encoder7Label );
86      encoder7hbox->addWidget( encoder7ComboBox );
87
88      encoder8hbox->addWidget( encoder8Label );
89      encoder8hbox->addWidget( encoder8ComboBox );
90
91      QPushButton *resetEncodersButton = new QPushButton( tr( "Disable All Encoders" ) );
92      connect( resetEncodersButton, SIGNAL( clicked( ) ), this, SLOT( disableEncoders( ) ) );
93
94      QPushButton *resetToDefaultsButton = new QPushButton( tr( "Reset To Defaults" ) );
95      connect( resetToDefaultsButton, SIGNAL( clicked( ) ), this, SIGNAL( resetToDefaults( ) )
            );
96
97      QGridLayout *grid = new QGridLayout;
98
99      grid->addLayout( blockSizeHbox, 0, 0, Qt::AlignTop );
100     grid->addLayout( encoder1hbox, 1, 0, Qt::AlignTop );
101     grid->addLayout( encoder2hbox, 2, 0, Qt::AlignTop );
```

```
102     grid->addLayout( encoder3hbox, 3, 0, Qt::AlignTop );
103     grid->addLayout( encoder4hbox, 4, 0, Qt::AlignTop );
104     grid->addLayout( encoder5hbox, 5, 0, Qt::AlignTop );
105     grid->addLayout( encoder6hbox, 6, 0, Qt::AlignTop );
106     grid->addLayout( encoder7hbox, 7, 0, Qt::AlignTop );
107     grid->addLayout( encoder8hbox, 8, 0, Qt::AlignTop );
108     grid->addWidget( resetEncodersButton, 9, 0, Qt::AlignTop );
109     grid->addWidget( resetToDefaultsButton, 10, 0, Qt::AlignTop );
110
111     setupWidgetsConnections( );
112
113     setLayout( grid );
114 }
115
116 CompressSettingsPanel::~CompressSettingsPanel( ) { }
117
118 void
119 CompressSettingsPanel::disableEncoders( )
120 {
121     encoder1ComboBox->setCurrentIndex( 0 );
122     encoder2ComboBox->setCurrentIndex( 0 );
123     encoder3ComboBox->setCurrentIndex( 0 );
124     encoder4ComboBox->setCurrentIndex( 0 );
125     encoder5ComboBox->setCurrentIndex( 0 );
126     encoder6ComboBox->setCurrentIndex( 0 );
127     encoder7ComboBox->setCurrentIndex( 0 );
128     encoder8ComboBox->setCurrentIndex( 0 );
129 }
130
131 void
132 CompressSettingsPanel::set( unsigned int blockSize, QList< Compressor::CoderTypes>
        compressSequence )
133 {
134     this->blockSizeSpinBox->setValue( blockSize );
135
136     encoder1ComboBox->setCurrentIndex( compressSequence[0] );
137     encoder2ComboBox->setCurrentIndex( compressSequence[1] );
138     encoder3ComboBox->setCurrentIndex( compressSequence[2] );
139     encoder4ComboBox->setCurrentIndex( compressSequence[3] );
140     encoder5ComboBox->setCurrentIndex( compressSequence[4] );
141     encoder6ComboBox->setCurrentIndex( compressSequence[5] );
142     encoder7ComboBox->setCurrentIndex( compressSequence[6] );
143     encoder8ComboBox->setCurrentIndex( compressSequence[7] );
144 }
145
146 void
147 CompressSettingsPanel::setupWidgetsConnections( )
148 {
149     connect( blockSizeSpinBox, SIGNAL( valueChanged( int ) ), this, SLOT( get( ) ) );
150
151     connect( encoder1ComboBox, SIGNAL( currentIndexChanged( int ) ), this, SLOT( get( ) ) );
152     connect( encoder2ComboBox, SIGNAL( currentIndexChanged( int ) ), this, SLOT( get( ) ) );
153     connect( encoder3ComboBox, SIGNAL( currentIndexChanged( int ) ), this, SLOT( get( ) ) );
154     connect( encoder4ComboBox, SIGNAL( currentIndexChanged( int ) ), this, SLOT( get( ) ) );
155     connect( encoder5ComboBox, SIGNAL( currentIndexChanged( int ) ), this, SLOT( get( ) ) );
156     connect( encoder6ComboBox, SIGNAL( currentIndexChanged( int ) ), this, SLOT( get( ) ) );
157     connect( encoder7ComboBox, SIGNAL( currentIndexChanged( int ) ), this, SLOT( get( ) ) );
158     connect( encoder8ComboBox, SIGNAL( currentIndexChanged( int ) ), this, SLOT( get( ) ) );
159 }
```

```
160
161 void
162 CompressSettingsPanel::get( )
163 {
164     QList< Compressor::CoderTypes> compressSequence;
165     compressSequence << static_cast < Compressor::CoderTypes > ( encoder1ComboBox->
        currentIndex( ) )
166         << static_cast < Compressor::CoderTypes > ( encoder2ComboBox->currentIndex( ) )
167         << static_cast < Compressor::CoderTypes > ( encoder3ComboBox->currentIndex( ) )
168         << static_cast < Compressor::CoderTypes > ( encoder4ComboBox->currentIndex( ) )
169         << static_cast < Compressor::CoderTypes > ( encoder5ComboBox->currentIndex( ) )
170         << static_cast < Compressor::CoderTypes > ( encoder6ComboBox->currentIndex( ) )
171         << static_cast < Compressor::CoderTypes > ( encoder7ComboBox->currentIndex( ) )
172         << static_cast < Compressor::CoderTypes > ( encoder8ComboBox->currentIndex( ) );
173
174     unsigned int blockSize = blockSizeSpinBox->value( );
175
176     emit settingsChanged( blockSize, compressSequence );
177 }
```

Б.16 Модуль DecompressSettingsPanel

Листинг Б.31 — **GUI/SettingsPanels/decompressSettingsPanel.h**

```
1  /*
2   * File:   DecompressSetingsPanel.h
3   * Author: art
4   *
5   * Created on 21 Мартъ 2011 г., 13:10
6   */
7
8  #ifndef DECOMPRESSSETTINGSPANEL_H
9  #    define DECOMPRESSSETTINGSPANEL_H
10
11 #    include <QWidget>
12 #    include "Compressor/compressor.h"
13
14 class QCheckBox;
15
16 class DecompressSettingsPanel : public QWidget
17 {
18     Q_OBJECT
19 public:
20     DecompressSettingsPanel( );
21     virtual ~ DecompressSettingsPanel( );
22
23 public slots:
24     void        set( bool keepBroken );
25 signals:
26     void        settingsChanged( bool keepBroken );
27     void        resetToDefaults( void );
28 private:
29     QCheckBox   *keepBrokenFilesCheckbox;
30
31     void        setupWidgetsConnections( );
32
```

```
33 private slots:
34     void         get( );
35
36 } ;
37
38 #endif  /* DECOMPRESSSETTINGSPANEL_H */
```

Листинг Б.32 — **GUI/SettingsPanels/decompressSettingsPanel.cpp**

```
1  /*
2   * File:   DecompressSetingsPanel.cpp
3   * Author: art
4   *
5   * Created on 21 Мартъ 2011 г., 13:10
6   */
7
8  #include <QCheckBox>
9  #include <QGridLayout>
10 #include <QPushButton>
11 #include "decompressSetingsPanel.h"
12
13 DecompressSettingsPanel::DecompressSettingsPanel( )
14 {
15     keepBrokenFilesCheckbox = new QCheckBox( "Keep broken files", this );
16     QGridLayout *grid = new QGridLayout;
17
18     QPushButton *resetToDefaultsButton = new QPushButton( tr( "Reset To Defaults" ) );
19     connect( resetToDefaultsButton, SIGNAL( clicked( ) ), this, SIGNAL( resetToDefaults( ) )
           );
20
21     grid->addWidget( keepBrokenFilesCheckbox, 0, 0, Qt::AlignTop );
22     grid->addWidget( resetToDefaultsButton, 1, 0, Qt::AlignBottom );
23
24     setLayout( grid );
25     setupWidgetsConnections( );
26 }
27
28 DecompressSettingsPanel::~DecompressSettingsPanel( ) { }
29
30 void
31 DecompressSettingsPanel::get( )
32 {
33     bool keepBroken = keepBrokenFilesCheckbox->isChecked( );
34     emit settingsChanged( keepBroken );
35 }
36
37 void
38 DecompressSettingsPanel::set( bool keepBroken )
39 {
40     keepBrokenFilesCheckbox->setChecked( keepBroken );
41 }
42
43 void
44 DecompressSettingsPanel::setupWidgetsConnections( )
45 {
46     connect( keepBrokenFilesCheckbox, SIGNAL( stateChanged( int ) ), this, SLOT( get( ) ) );
47 }
```

Б.17 Модуль FileList

Листинг Б.33 — **GUI/FileList/fileList.h**

```
/*
 * File:   FileList.h
 * Author: art
 *
 * Created on 14 Марта 2011 г., 18:38
 */

#ifndef FILELIST_H
#    define FILELIST_H

#    include <QTableWidget>
#    include <QDir>

#    include "Compressor/FilesTable/filesTable.h"
#    include "../CompressorThread/compressorThread.h"

class FileList : public QTableWidget
{
    Q_OBJECT
public:
    FileList( );
    virtual ~ FileList( );
    void                setFileList( const QStringList & fileList, QDir & basedir );

public slots:
    void                init( );
    void                setProceedFileStatus( const QString &fileName, unsigned int id );
    void                setFailFileStatus( const QString &fileName, unsigned int id );
    void                setSuccessFileStatus( const QString &fileName, unsigned int id );
    void                setCancelledFilesStatus( const QString &fileName, unsigned int id )
        ;
    void                setCorruptedFileStatus( const QString &fileName, unsigned int id );
    void                setUnCorruptedFileStatus( const QString &fileName, unsigned int id
        );

    void                showInfo( CTCompressorStatus::ErrorCode error, QString fileName,
        unsigned int id);
private:

    void                initTable( int rowCount, int columnCount );
    QTableWidgetItem*   getItemById( int id );

};

inline QTableWidgetItem*    FileList::getItemById( int id )
{
    int index = id - 1;
    if (id > rowCount( ))
        setRowCount( id );

    setItem( index, 0, new QTableWidgetItem( ) );
    setItem( index, 1, new QTableWidgetItem( ) );
    return item( index, 0 );
}
```

```
52
53 #endif  /* FILELIST_H */
```

Листинг Б.34 — **GUI/FileList/fileList.cpp**

```
1  /*
2   * File:   FileList.cpp
3   * Author: art
4   *
5   * Created on 14 Марта 2011 г., 18:38
6   */
7
8  #include "fileList.h"
9  #include "Compressor/FilesTable/filesTable.h"
10 #include "Compressor/compressor.h"
11
12 FileList::FileList( )
13 {
14     init( );
15     setColumnWidth( 0, 320 );
16     setColumnWidth( 1, 90 );
17 }
18
19 FileList::~FileList( ) { }
20
21 void
22 FileList::init( )
23 {
24     QTableWidget::clear( );
25     initTable( 0, 2 );
26 }
27
28 void
29 FileList::initTable( int rowCount, int columnCount )
30 {
31     setRowCount( rowCount );
32     setColumnCount( columnCount );
33     setHorizontalHeaderLabels( QStringList( ) << "File Name" << "Status" );
34 }
35
36 void
37 FileList::setFileList( const QStringList & fileList, QDir & basedir )
38 {
39     initTable( fileList.count( ), 2 );
40
41     for ( int i = 0; i < rowCount( ); ++i )
42     {
43         setItem( i, 0, new QTableWidgetItem( basedir.relativeFilePath( fileList[i] ) ) );
44         setItem( i, 1, new QTableWidgetItem( ) );
45     }
46 }
47
48 void
49 FileList::setFailFileStatus( const QString &fileName, unsigned int id )
50 {
51     QTableWidgetItem *failedFileCell = getItemById( id );
52
53     failedFileCell->setBackgroundColor( Qt::red );
```

```
54      failedFileCell->setText( fileName );
55      item( failedFileCell->row( ), failedFileCell->column( ) + 1 )->setBackgroundColor( Qt::
            red );
56      item( failedFileCell->row( ), failedFileCell->column( ) + 1 )->setText( "Fail!" );
57  }
58
59  void
60  FileList::setProceedFileStatus( const QString &fileName, unsigned int id )
61  {
62      QTableWidgetItem *proceedFileCell = getItemById( id );
63
64      proceedFileCell->setBackgroundColor( Qt::yellow );
65      proceedFileCell->setText( fileName );
66      item( proceedFileCell->row( ), proceedFileCell->column( ) + 1 )->setBackgroundColor( Qt
            ::yellow );
67      item( proceedFileCell->row( ), proceedFileCell->column( ) + 1 )->setText( "Proceed..." )
            ;
68  }
69
70  void
71  FileList::setSuccessFileStatus( const QString &fileName, unsigned int id )
72  {
73      QTableWidgetItem *succesedFileCell = getItemById( id );
74
75      succesedFileCell->setBackgroundColor( Qt::green );
76      succesedFileCell->setText( fileName );
77      item( succesedFileCell->row( ), succesedFileCell->column( ) + 1 )->setBackgroundColor(
            Qt::green );
78      item( succesedFileCell->row( ), succesedFileCell->column( ) + 1 )->setText( "Success" );
79  }
80
81  void
82  FileList::setCancelledFilesStatus( const QString &fileName, unsigned int id )
83  {
84      QTableWidgetItem *cancelledFileCell = getItemById( id );
85
86      cancelledFileCell->setText( fileName );
87      cancelledFileCell->setBackgroundColor( Qt::red );
88      item( cancelledFileCell->row( ), cancelledFileCell->column( ) + 1 )->setBackgroundColor(
             Qt::red );
89      item( cancelledFileCell->row( ), cancelledFileCell->column( ) + 1 )->setText( "Cancelled
            " );
90  }
91
92  void
93  FileList::setCorruptedFileStatus( const QString &fileName, unsigned int id )
94  {
95      QTableWidgetItem *corruptedFileCell = getItemById( id );
96
97      corruptedFileCell->setText( fileName );
98      corruptedFileCell->setBackgroundColor( Qt::red );
99      item( corruptedFileCell->row( ), corruptedFileCell->column( ) + 1 )->setBackgroundColor(
             Qt::red );
100     item( corruptedFileCell->row( ), corruptedFileCell->column( ) + 1 )->setText( "Corrupted
            !" );
101 }
102
103 void
104 FileList::setUnCorruptedFileStatus( const QString &fileName, unsigned int id )
```

```
105 {
106     QTableWidgetItem *corruptedFileCell = getItemById( id );
107 
108     corruptedFileCell->setText( fileName );
109     corruptedFileCell->setBackgroundColor( Qt::green );
110     item( corruptedFileCell->row( ), corruptedFileCell->column( ) + 1 )->setBackgroundColor(
          Qt::green );
111     item( corruptedFileCell->row( ), corruptedFileCell->column( ) + 1 )->setText( "
        Uncorrupted" );
112 }
113 
114 void
115 FileList::showInfo( CTCompressorStatus::ErrorCode error, QString fileName, unsigned int id )
116 {
117     if ( fileName.isEmpty( ) )
118         return;
119 
120     switch ( error )
121     {
122         case CTCompressorStatus::INPUT_FILE_OPEN_ERROR: setFailFileStatus( fileName, id );
123             break;
124         case CTCompressorStatus::SUCCESS: setSuccessFileStatus( fileName, id );
125             break;
126         case CTCompressorStatus::PROCEED: setProceedFileStatus( fileName, id );
127             break;
128         case CTCompressorStatus::INPUT_FILE_CORRUPTED: setCorruptedFileStatus( fileName, id
              );
129             break;
130         case CTCompressorStatus::INPUT_FILE_UNCORRUPTED: setUnCorruptedFileStatus( fileName,
               id );
131             break;
132         case CTCompressorStatus::CANCELLED: setCancelledFilesStatus( fileName, id );
133             break;
134         default: break;
135     }
136 }
```

Б.18 Модуль StatInfoW

Листинг Б.35 — **GUI/StatInfoW/statInfoW.h**

```
1  /*
2   * File:   StatInfo.h
3   * Author: art
4   *
5   * Created on 21 Марта 2011 г., 14:04
6   */
7 
8  #ifndef STATINFOW_H
9  #    define STATINFOW_H
10 
11 #    include "QDialog"
12 #    include "Compressor/compressor.h"
13 
14 class QTableWidget;
15 
```

```
16 class QLabel;
17
18 class StatInfoW : public QDialog
19 {
20
21     Q_OBJECT
22 public:
23     struct StatInfo
24     {
25         QList< Compressor::CoderTypes> compressSequence;
26         unsigned int blockSize;
27         off_t inputSize;
28         off_t outputSize;
29         float speed;
30     } ;
31     StatInfoW( QWidget *parent = NULL );
32     virtual ~ StatInfoW( );
33 public slots:
34     void            showInfo( StatInfo info );
35 private:
36     QLabel          *blockSizeLabel;
37     QLabel          *inputDataSizeLabel;
38     QLabel          *outputDataSizeLabel;
39     QLabel          *ratioLabel;
40     QLabel          *speedLabel;
41     QLabel          *encodingSequenceLabel;
42     QLabel          *encodingSequenceColumnLabel;
43
44
45 } ;
46
47 #endif  /* STATINFOW_H */
```

Листинг Б.36 — **GUI/StatInfoW/statInfoW.cpp**

```
1  /*
2   * File:   StatInfo.cpp
3   * Author: art
4   *
5   * Created on 21 Марта 2011 г., 14:04
6   */
7
8  #include <QLabel>
9  #include <QGridLayout>
10 #include <QPushButton>
11 #include "statInfoW.h"
12 #include "../DataUnitsToQString/dataUnitsToQString.h"
13
14 StatInfoW::StatInfoW( QWidget * parent ) : QDialog( parent, Qt::WindowTitleHint | Qt::
       WindowSystemMenuHint | Qt::WindowCloseButtonHint )
15 {
16     encodingSequenceColumnLabel = new QLabel( tr( "Encoding Sequence" ) );
17     encodingSequenceLabel = new QLabel( );
18
19     QLabel *blockSizeColumnLabel = new QLabel( tr( "Block Size" ) );
20     blockSizeLabel = new QLabel( );
21
22     QLabel *inputDataSizeColumnLabel = new QLabel( tr( "Input Data Size" ) );
```

```
23      inputDataSizeLabel = new QLabel( );
24
25      QLabel *outputDataSizeColumnLabel = new QLabel( tr( "Output Data Size" ) );
26      outputDataSizeLabel = new QLabel( );
27
28      QLabel *ratioColumnLabel = new QLabel( tr( "Ratio" ) );
29      ratioLabel = new QLabel( );
30
31      QLabel *speedColumnLabel = new QLabel( tr( "Speed" ) );
32      speedLabel = new QLabel( );
33
34      QPushButton *okButton = new QPushButton( "&Ok" );
35      connect( okButton, SIGNAL( clicked( ) ), this, SLOT( accept( ) ) );
36
37      QGridLayout *grid = new QGridLayout;
38
39      grid->setRowMinimumHeight( 0, 10 );
40
41      grid->addWidget( encodingSequenceColumnLabel, 1, 0, Qt::AlignTop );
42      grid->addWidget( encodingSequenceLabel, 1, 1, Qt::AlignTop );
43
44      grid->addWidget( blockSizeColumnLabel, 2, 0, Qt::AlignTop );
45      grid->addWidget( blockSizeLabel, 2, 1, Qt::AlignTop );
46
47      grid->addWidget( inputDataSizeColumnLabel, 3, 0, Qt::AlignTop );
48      grid->addWidget( inputDataSizeLabel, 3, 1, Qt::AlignTop );
49
50      grid->addWidget( outputDataSizeColumnLabel, 4, 0, Qt::AlignTop );
51      grid->addWidget( outputDataSizeLabel, 4, 1, Qt::AlignTop );
52
53      grid->addWidget( ratioColumnLabel, 5, 0, Qt::AlignTop );
54      grid->addWidget( ratioLabel, 5, 1, Qt::AlignTop );
55
56      grid->addWidget( speedColumnLabel, 6, 0, Qt::AlignTop );
57      grid->addWidget( speedLabel, 6, 1, Qt::AlignTop );
58
59      grid->addWidget( okButton, 7, 1, Qt::AlignBottom );
60
61      setLayout( grid );
62  }
63
64  StatInfoW::~StatInfoW( ) { }
65
66  void
67  StatInfoW::showInfo( StatInfo info )
68  {
69      blockSizeLabel->setText( DataUnitsToQString::convertDataSize( info.blockSize, 2 ) );
70      inputDataSizeLabel->setText( DataUnitsToQString::convertDataSize( info.inputSize, 2 ) );
71      outputDataSizeLabel->setText( DataUnitsToQString::convertDataSize( info.outputSize, 2 )
            );
72
73      speedLabel->setText( DataUnitsToQString::convertDataSpeed( info.speed, 2 ) );
74
75      encodingSequenceLabel->setText( "" );
76
77      QList< Compressor::CoderTypes> &compressSequence = info.compressSequence;
78      if ( !compressSequence.empty( ) )//Compress Mode
79      {
```

```
80          ratioLabel->setText( QString::number( ( float ) 100 * info.outputSize / info.
                inputSize, 'f', 2 ) + " %" );
81          for ( QList< Compressor::CoderTypes >::iterator i = compressSequence.begin( ); i !=
                compressSequence.end( ); ++i )
82          {
83              switch ( *i )
84              {
85                  case Compressor::BWT: encodingSequenceLabel->setText( encodingSequenceLabel
                        ->text( ) + "BWT " );
86                      break;
87                  case Compressor::HUFFMAN:encodingSequenceLabel->setText(
                        encodingSequenceLabel->text( ) + "HUFFMAN " );
88                      break;
89                  case Compressor::MTF:encodingSequenceLabel->setText( encodingSequenceLabel->
                        text( ) + "MTF " );
90                      break;
91                  case Compressor::RLE:encodingSequenceLabel->setText( encodingSequenceLabel->
                        text( ) + "RLE " );
92                      break;
93                  case Compressor::NONE:
94                      break;
95              }
96          }
97      }
98      else//Decompress Mode
99      {
100         ratioLabel->setText( QString::number( ( float ) 100 * info.outputSize  / info.
                inputSize, 'f', 2 ) + " %" );
101         encodingSequenceColumnLabel->hide( );
102         encodingSequenceLabel->hide( );
103     }
104
105     adjustSize( );
106     setFixedSize( width( ), height( ) );
107 }
```

Б.19 Модуль Face

Листинг Б.37 — **GUI/Face/face.h**

```
1  /*
2   * File:   face.h
3   * Author: art
4   *
5   * Created on 7 Мартъ 2011 г., 22:17
6   */
7
8  #ifndef _FACE_H
9  #    define _FACE_H
10
11 #    include <QFileDialog>
12 #    include "ui_face.h"
13 #    include "../Compressor/compressor.h"
14 #    include "../CompressorThread/compressorThread.h"
15
16 class FileList;
```

```
17 class QProgressDialog;
18 class CompressSettingsPanel;
19 class DecompressSettingsPanel;
20 class StatInfoW;
21
22 class Face : public QMainWindow
23 {
24     Q_OBJECT
25 public:
26
27     Face( );
28     virtual ~ Face( );
29
30 public slots:
31     void                        about( );
32     void                        selectFilesToCompress( );
33     void                        selectFileToDecompress( );
34     void                        compress( );
35     void                        decompress( );
36     void                        listArchiveContents( );
37     void                        displayCompressStatus( int progress, QString fileName, float
           speed );
38     void                        displayDecompressStatus( int progress, QString fileName,
           float speed );
39     void                        displayListStatus( int progress, QString fileName, float
           speed );
40     void                        showInfo( CTCompressorStatus::ErrorCode error, QString
           fileName);
41     void                        showCompressStatInfo( Compressor::Stat stat );
42     void                        showDecompressStatInfo( Compressor::Stat stat );
43     void                        setCompressSettings( unsigned int blockSize, QList<
           Compressor::CoderTypes> compressSequence );
44     void                        setDecompressSettings( bool keepBrokenFiles );
45 private:
46
47     CompressSettingsPanel       *compressSettingsPanel;
48     DecompressSettingsPanel     *decompressSettingsPanel;
49
50     StatInfoW                   *statInfoW;
51
52     CompressorThread            *compressThread;
53     CompressorThread            *decompressThread;
54     CompressorThread            *listArchiveThread;
55     Ui::face                    widget;
56     Compressor                  *compressor;
57
58     QToolBar                    *actionToolBar;
59
60     FileList                    *filelist;
61
62     QStringList                 sourceFileNames;
63     QStringList                 encodedfileNames;
64
65     QString                     destFileName;
66     QString                     destDirName;
67
68     QDir                        compressBaseDir;
69     QDir                        decompressBaseDir;
70
```

```
71      QDockWidget                     *compressSettingsDock;
72      QDockWidget                     *decompressSettingsDock;
73
74      QAction                         *selectFilesToCompressAction;
75      QAction                         *selectFileToDecompressAction;
76      QAction                         *compressAction;
77      QAction                         *decompressAction;
78      QAction                         *exitAction;
79      QAction                         *aboutAction;
80      QAction                         *aboutQtAction;
81
82      QMenu                           *fileMenu;
83      QMenu                           *helpMenu;
84
85      QProgressDialog                 *compressingProgressDialog;
86      QProgressDialog                 *decompressingProgressDialog;
87      QProgressDialog                 *listArchiveContentsProgressDialog;
88
89      bool                            brokenFileWarningShown;
90
91      QList< Compressor::CoderTypes> compressSequence;
92      unsigned int                    blockSize;
93      bool                            keepBrokenFiles;
94
95      void                            createActions( );
96      void                            createToolBars( );
97      void                            createMenus( );
98      void                            setupWidgetsConnections( );
99      void                            activateCompressMode( );
100     void                            activateDecompressMode( );
101
102 private slots:
103     void                            initSettings( );
104
105 };
106
107
108 #endif  /* _FACE_H */
```

Листинг Б.38 — **GUI/Face/face.cpp**

```
1  /*
2   * File:   face.cpp
3   * Author: art
4   *
5   * Created on 7 Мартм 2011 г., 22:17
6   */
7
8  #include <QFileDialog>
9  #include <QToolBar>
10 #include <QMessageBox>
11 #include <QProgressDialog>
12 #include <QDockWidget>
13 #include "face.h"
14 #include "../FileList/fileList.h"
15 #include "../CompressorThread/compressorThread.h"
16 #include "../SettingsPanels/compressSettingsPanel.h"
17 #include "../SettingsPanels/decompressSetingsPanel.h"
```

```
18 #include "../StatInfoW/statInfoW.h"
19 #include "../DataUnitsToQString/dataUnitsToQString.h"
20
21 Face::Face( ) : sourceFileNames( )
22 {
23     widget.setupUi( this );
24
25     filelist = new FileList;
26     setCentralWidget( filelist );
27
28     compressSettingsPanel = new CompressSettingsPanel;
29     compressSettingsDock = new QDockWidget( tr( "Compress Settings" ), this );
30     compressSettingsDock->setAllowedAreas( Qt::LeftDockWidgetArea );
31     compressSettingsDock->setWidget( compressSettingsPanel );
32     addDockWidget( Qt::LeftDockWidgetArea, compressSettingsDock );
33
34     compressSettingsDock->hide( );
35
36     decompressSettingsPanel = new DecompressSettingsPanel;
37     decompressSettingsDock = new QDockWidget( tr( "Decompress Settings" ), this );
38     decompressSettingsDock->setAllowedAreas( Qt::LeftDockWidgetArea );
39     decompressSettingsDock->setWidget( decompressSettingsPanel );
40     addDockWidget( Qt::LeftDockWidgetArea, decompressSettingsDock );
41
42     decompressSettingsDock->hide( );
43
44     statInfoW = new StatInfoW( this );
45
46     compressThread = new CompressorThread;
47     decompressThread = new CompressorThread;
48     listArchiveThread = new CompressorThread;
49
50     compressingProgressDialog = new QProgressDialog( "Compressing files...", "Abort", 0,
           100, this );
51     decompressingProgressDialog = new QProgressDialog( "Decompressing files...", "Abort", 0,
           100, this );
52     listArchiveContentsProgressDialog = new QProgressDialog( "Reading archive contents...",
           "Abort", 0, 100, this );
53
54     qRegisterMetaType< CTCompressorStatus::ErrorCode > ( "CTCompressorStatus::ErrorCode" );
55     qRegisterMetaType< Compressor::Stat > ( "Compressor::Stat" );
56
57     createActions( );
58     createMenus( );
59     createToolBars( );
60     setupWidgetsConnections( );
61     initSettings( );
62 }
63
64 Face::~Face( )
65 {
66     delete compressingProgressDialog;
67     delete decompressingProgressDialog;
68     delete statInfoW;
69 }
70
71 void
72 Face::about( )
73 {
```

```
74      const char *htmlText =
75          "<HTML>"
76          "<p>JAA - <b>J</b>ust <b>A</b>nother <b>A</b>rchiver</p>"
77          "<p>Author: Artur Molchanov <a href=""mailto:arturmolchanov@gmail.com"">
                arturmolchanov@gmail.com</a> </p>"
78          "</HTML>";
79      QMessageBox::about( this, tr( "About YAA" ), htmlText );
80  }
81
82  void
83  Face::activateCompressMode( )
84  {
85      decompressAction->setEnabled( false );
86      compressAction->setEnabled( true );
87      compressSettingsDock->show( );
88      decompressSettingsDock->hide( );
89  }
90
91  void
92  Face::activateDecompressMode( )
93  {
94      compressAction->setEnabled( false );
95      decompressAction->setEnabled( true );
96      decompressSettingsDock->show( );
97      compressSettingsDock->hide( );
98  }
99
100 void
101 Face::createActions( )
102 {
103     selectFilesToCompressAction = new QAction( tr( "Select files to compress" ), this );
104     connect( selectFilesToCompressAction, SIGNAL( triggered( ) ), this, SLOT(
            selectFilesToCompress( ) ) );
105
106     selectFileToDecompressAction = new QAction( tr( "Select file to decompress" ), this );
107     connect( selectFileToDecompressAction, SIGNAL( triggered( ) ), this, SLOT(
            selectFileToDecompress( ) ) );
108
109     compressAction = new QAction( tr( "Compress" ), this );
110     compressAction->setEnabled( false );
111     connect( compressAction, SIGNAL( triggered( ) ), this, SLOT( compress( ) ) );
112
113     decompressAction = new QAction( tr( "Decompress" ), this );
114     decompressAction->setEnabled( false );
115     connect( decompressAction, SIGNAL( triggered( ) ), this, SLOT( decompress( ) ) );
116
117     exitAction = new QAction( tr( "E&xit" ), this );
118     connect( exitAction, SIGNAL( triggered( ) ), qApp, SLOT( quit( ) ) );
119
120     aboutQtAction = new QAction( QIcon( ":/images/qt.png" ), tr( "About &Qt" ), this );
121     connect( aboutQtAction, SIGNAL( triggered( ) ), qApp, SLOT( aboutQt( ) ) );
122
123     aboutAction = new QAction( QIcon( ":/images/qt.png" ), tr( "About" ), this );
124     connect( aboutAction, SIGNAL( triggered( ) ), this, SLOT( about( ) ) );
125 }
126
127 void
128 Face::createToolBars( )
129 {
```

```
130     actionToolBar = addToolBar( tr( "Action" ) );
131     actionToolBar->addAction( compressAction );
132     actionToolBar->addAction( decompressAction );
133 }
134
135 void
136 Face::createMenus( )
137 {
138     fileMenu = menuBar( )->addMenu( tr( "File" ) );
139     fileMenu->addAction( selectFilesToCompressAction );
140     fileMenu->addAction( selectFileToDecompressAction );
141     fileMenu->addSeparator( );
142     fileMenu->addAction( exitAction );
143
144     helpMenu = menuBar( )->addMenu( tr( "Help" ) );
145     helpMenu->addAction( aboutAction );
146     helpMenu->addAction( aboutQtAction );
147 }
148
149 void
150 Face::compress( )
151 {
152     QFileDialog destFileDialog;
153     destFileDialog.setWindowTitle( "Select Output File" );
154
155     if ( destFileDialog.exec( ) )
156         destFileName = destFileDialog.selectedFiles( )[0];
157     else return;
158
159     compressingProgressDialog->setWindowModality( Qt::WindowModal );
160     compressingProgressDialog->setAutoClose( false );
161     compressingProgressDialog->reset( );
162
163     compressThread->initCompress( sourceFileNames, destFileName, blockSize, compressBaseDir,
          compressSequence );
164     compressingProgressDialog->show( );
165     compressThread->start( );
166 }
167
168 void
169 Face::decompress( )
170 {
171     QFileDialog destDirDialog;
172     destDirDialog.setFileMode( QFileDialog::DirectoryOnly );
173     destDirDialog.setWindowTitle( "Select Output Folder" );
174
175     if ( destDirDialog.exec( ) )
176         destDirName = destDirDialog.selectedFiles( )[0];
177     else return;
178
179     while ( !QDir::setCurrent( destDirName ) )
180     {
181         QMessageBox::critical( this, "Error!", "Cannot change folder to " + destDirName + "\
              nPlease select other folder!" );
182         if ( destDirDialog.exec( ) )
183             destDirName = destDirDialog.selectedFiles( )[0];
184         else return;
185     }
186     decompressingProgressDialog->setWindowModality( Qt::WindowModal );
```

```
187     decompressingProgressDialog->setAutoClose( false );
188     decompressingProgressDialog->reset( );
189
190     decompressThread->initDecompress( encodedfileNames[0], keepBrokenFiles );
191     decompressingProgressDialog->show( );
192     decompressThread->start( );
193 }
194
195 void
196 Face::displayCompressStatus( int progress, QString fileName, float speed )
197 {
198     compressingProgressDialog->setValue( progress );
199     compressingProgressDialog->setLabelText( "Current file: " + fileName + "\n" + "Speed: "
        + DataUnitsToQString::convertDataSpeed( speed, 2 ) );
200 }
201
202 void
203 Face::displayDecompressStatus( int progress, QString fileName, float speed )
204 {
205     decompressingProgressDialog->setValue( progress );
206     decompressingProgressDialog->setLabelText( "Current file: " + fileName + "\n" + "Speed:
        " + DataUnitsToQString::convertDataSpeed( speed, 2 ) );
207 }
208
209 void
210 Face::displayListStatus( int progress, QString fileName, float speed )
211 {
212     listArchiveContentsProgressDialog->setValue( progress );
213     listArchiveContentsProgressDialog->setLabelText( "Current file: " + fileName + "\n" + "
        Speed: " + DataUnitsToQString::convertDataSpeed( speed, 2 ) );
214 }
215
216 void
217 Face::initSettings( )
218 {
219     compressSequence.clear( );
220     compressSequence << Compressor::RLE <<
221         Compressor::BWT <<
222         Compressor::MTF <<
223         Compressor::RLE <<
224         Compressor::HUFFMAN <<
225         Compressor::NONE <<
226         Compressor::NONE <<
227         Compressor::NONE;
228     blockSize = 900000;
229     compressSettingsPanel->set( blockSize / 1000, compressSequence );
230     keepBrokenFiles = true;
231     decompressSettingsPanel->set( keepBrokenFiles );
232 }
233
234 void
235 Face::listArchiveContents( )
236 {
237     brokenFileWarningShown = false;
238     listArchiveContentsProgressDialog->setWindowModality( Qt::WindowModal );
239     listArchiveContentsProgressDialog->setAutoClose( false );
240
241     listArchiveThread->initList( encodedfileNames[0] );
242     listArchiveContentsProgressDialog->show( );
```

```
243     listArchiveThread->start( );
244 }
245
246 void
247 Face::selectFilesToCompress( )
248 {
249     QFileDialog openFileToCompressDialog;
250     openFileToCompressDialog.setFileMode( QFileDialog::ExistingFiles );
251     openFileToCompressDialog.setWindowTitle( "Select Files To Compress" );
252
253     if ( openFileToCompressDialog.exec( ) )
254     {
255         sourceFileNames = openFileToCompressDialog.selectedFiles( );
256         compressBaseDir = openFileToCompressDialog.directory( );
257     }
258     else return;
259
260     while ( !QDir::setCurrent( compressBaseDir.absolutePath( ) + "/" ) )
261     {
262         QMessageBox::critical( this, "Error!", "Folder " + compressBaseDir.path( ) + " is
                not accessible" + "\nPlease select other files!" );
263         if ( openFileToCompressDialog.exec( ) )
264         {
265             sourceFileNames = openFileToCompressDialog.selectedFiles( );
266             compressBaseDir = openFileToCompressDialog.directory( );
267         }
268         else return;
269     }
270
271     filelist->setFileList( sourceFileNames, compressBaseDir );
272     activateCompressMode( );
273 }
274
275 void
276 Face::selectFileToDecompress( )
277 {
278     QFileDialog openFileToDecompressDialog;
279     openFileToDecompressDialog.setWindowTitle( tr( "Select File To Decompress" ) );
280     if ( openFileToDecompressDialog.exec( ) )
281     {
282         encodedfileNames = openFileToDecompressDialog.selectedFiles( );
283         decompressBaseDir = openFileToDecompressDialog.directory( );
284         listArchiveContents( );
285         activateDecompressMode( );
286     }
287 }
288
289 void
290 Face::setCompressSettings( unsigned int blockSize, QList< Compressor::CoderTypes>
        compressSequence )
291 {
292     this->compressSequence = compressSequence;
293     this->blockSize = 1000 * blockSize;
294 }
295
296 void
297 Face::setDecompressSettings( bool keepBrokenFiles )
298 {
299     this->keepBrokenFiles = keepBrokenFiles;
```

```
300 }
301
302 void
303 Face::setupWidgetsConnections( )
304 {
305     connect( compressSettingsPanel, SIGNAL( settingsChanged( unsigned int, QList< Compressor
        ::CoderTypes> ) ), this, SLOT( setCompressSettings( unsigned int, QList< Compressor
        ::CoderTypes> ) ) );
306     connect( compressSettingsPanel, SIGNAL( resetToDefaults( ) ), this, SLOT( initSettings(
        ) ) );
307
308     connect( decompressSettingsPanel, SIGNAL( settingsChanged( bool ) ), this, SLOT(
        setDecompressSettings( bool ) ) );
309     connect( decompressSettingsPanel, SIGNAL( resetToDefaults( ) ), this, SLOT( initSettings
        ( ) ) );
310
311     connect( compressThread, SIGNAL( progressChanged( int, QString, float ) ), this, SLOT(
        displayCompressStatus( int, QString, float ) ) );
312     connect( compressThread, SIGNAL( info( CTCompressorStatus::ErrorCode, QString, unsigned
        int ) ), this, SLOT( showInfo( CTCompressorStatus::ErrorCode, QString ) ) );
313     connect( compressingProgressDialog, SIGNAL( canceled( ) ), compressThread, SLOT( stop( )
         ) );
314     connect( compressThread, SIGNAL( finished( ) ), compressingProgressDialog, SLOT( reject(
         ) ) );
315     connect( compressThread, SIGNAL( info( CTCompressorStatus::ErrorCode, QString, unsigned
        int ) ), filelist, SLOT( showInfo( CTCompressorStatus::ErrorCode, QString, unsigned
        int ) ) );
316     connect( compressThread, SIGNAL( statInfo( Compressor::Stat ) ), this, SLOT(
        showCompressStatInfo( Compressor::Stat ) ) );
317
318     connect( decompressThread, SIGNAL( progressChanged( int, QString, float ) ), this, SLOT(
         displayDecompressStatus( int, QString, float ) ) );
319     connect( decompressingProgressDialog, SIGNAL( canceled( ) ), decompressThread, SLOT(
        stop( ) ) );
320     connect( decompressThread, SIGNAL( finished( ) ), decompressingProgressDialog, SLOT(
        reject( ) ) );
321     connect( decompressThread, SIGNAL( info( CTCompressorStatus::ErrorCode, QString,
        unsigned int ) ), filelist, SLOT( showInfo( CTCompressorStatus::ErrorCode, QString,
        unsigned int ) ) );
322     connect( decompressThread, SIGNAL( info( CTCompressorStatus::ErrorCode, QString,
        unsigned int ) ), this, SLOT( showInfo( CTCompressorStatus::ErrorCode, QString ) ) )
        ;
323     connect( decompressThread, SIGNAL( statInfo( Compressor::Stat ) ), this, SLOT(
        showDecompressStatInfo( Compressor::Stat ) ) );
324
325     connect( listArchiveThread, SIGNAL( progressChanged( int, QString, float ) ), this, SLOT
        ( displayListStatus( int, QString, float ) ) );
326     connect( listArchiveContentsProgressDialog, SIGNAL( canceled( ) ), listArchiveThread,
        SLOT( stop( ) ) );
327     connect( listArchiveThread, SIGNAL( finished( ) ), listArchiveContentsProgressDialog,
        SLOT( reject( ) ) );
328     connect( listArchiveThread, SIGNAL( info( CTCompressorStatus::ErrorCode, QString,
        unsigned int ) ), this, SLOT( showInfo( CTCompressorStatus::ErrorCode, QString ) ) )
        ;
329     connect( listArchiveThread, SIGNAL( info( CTCompressorStatus::ErrorCode, QString,
        unsigned int ) ), filelist, SLOT( showInfo( CTCompressorStatus::ErrorCode, QString,
        unsigned int ) ) );
330     connect( listArchiveThread, SIGNAL( started( ) ), filelist, SLOT( init( ) ) );
331 }
```

```
332
333 void
334 Face::showCompressStatInfo( Compressor::Stat stat )
335 {
336     if ( !compressingProgressDialog->wasCanceled( ) )
337     {
338         StatInfoW::StatInfo statInfo;
339         statInfo.speed = stat.speed;
340         statInfo.blockSize = blockSize;
341         statInfo.compressSequence = compressSequence;
342         statInfo.inputSize = stat.decodedSize;
343         statInfo.outputSize = stat.encodedSize;
344
345         statInfoW->showInfo( statInfo );
346         statInfoW->setWindowTitle( "Compressing Statistics" );
347
348         statInfoW->show( );
349     }
350 }
351
352 void
353 Face::showDecompressStatInfo( Compressor::Stat stat )
354 {
355     if ( !decompressingProgressDialog->wasCanceled( ) )
356     {
357         StatInfoW::StatInfo statInfo;
358         statInfo.speed = stat.speed;
359         statInfo.blockSize = blockSize;
360
361         QList< Compressor::CoderTypes> emptySeq;
362
363         statInfo.compressSequence = emptySeq;
364         statInfo.inputSize = stat.encodedSize;
365         statInfo.outputSize = stat.decodedSize;
366
367         statInfoW->showInfo( statInfo );
368         statInfoW->setWindowTitle( "Decompressing Statistics" );
369
370         statInfoW->show( );
371     }
372 }
373
374 void
375 Face::showInfo( CTCompressorStatus::ErrorCode error, QString fileName )
376 {
377     switch ( error )
378     {
379         case CTCompressorStatus::INPUT_FILE_OPEN_ERROR:
380             if ( fileName.isEmpty( ) )
381             {
382                 QMessageBox::critical( this, "Error!", "Cannot open input file!" ); //
                        archive file
383             }
384             else
385                 QMessageBox::warning( this, "Error!", "Cannot open input file " + fileName +
                        "!" ); //raw file
386             break;
387         case CTCompressorStatus::INPUT_FILE_CORRUPTED:
388         {
```

```
389             if ( !brokenFileWarningShown )
390             {
391                 brokenFileWarningShown = true;
392                 QMessageBox::warning( this, "Error!", "Input File Corrupted!" );
393             }
394         }
395             break;
396         case CTCompressorStatus::OUTPUT_FILE_WRITE_ERROR:QMessageBox::critical( this, "Error
                !", "Cannot write to output file " + fileName + "!" );
397             break;
398         case CTCompressorStatus::DECOMPRESS_FAIL:QMessageBox::critical( this, "Error!", "
                Decompressing failed!" );
399             break;
400         default:break;
401     }
402 }
```

Б.20 Модуль DataUnitsToQString

Листинг Б.39 — **GUI/DataUnitsToQString/dataUnitsToQString.h**

```
1  /*
2   * File:   DataSizeToQString.h
3   * Author: art
4   *
5   * Created on 22 Март 2011 г., 15:46
6   */
7
8  #ifndef DATAUNITSTOQSTRING_H
9  #    define DATAUNITSTOQSTRING_H
10
11 #    include <QString>
12
13 class DataUnitsToQString
14 {
15 public:
16     DataUnitsToQString( );
17     DataUnitsToQString( const DataUnitsToQString& orig );
18     virtual ~ DataUnitsToQString( );
19     static QString convertDataSize( off_t dataSize, int prec) ;
20     static QString convertDataSpeed( int dataSpeed, int prec) ;
21 private:
22
23 } ;
24
25 #endif  /* DATAUNITSTOQSTRING_H */
```

Листинг Б.40 — **GUI/DataUnitsToQString/dataUnitsToQString.cpp**

```
1  /*
2   * File:   DataSizeToQString.cpp
3   * Author: art
4   *
5   * Created on 22 Март 2011 г., 15:46
6   */
7
```

```
8  #include "dataUnitsToQString.h"
9
10 DataUnitsToQString::DataUnitsToQString( ) { }
11
12 DataUnitsToQString::DataUnitsToQString( const DataUnitsToQString& ) { }
13
14 DataUnitsToQString::~DataUnitsToQString( ) { }
15
16 QString
17 DataUnitsToQString::convertDataSize( off_t dataSize, int prec )
18 {
19     if ( dataSize > 1E9 )
20         return QString::number( ( float ) dataSize / 1E9, 'f', prec ) + " GB";
21     else if ( dataSize > 1E6 )
22         return QString::number( ( float ) dataSize / 1E6, 'f', prec ) + " MB";
23     else if ( dataSize > 1E3 )
24         return QString::number( ( float ) dataSize / 1E3, 'f', prec ) + " KB";
25     else
26         return QString::number( dataSize ) + " B";
27 }
28
29 QString
30 DataUnitsToQString::convertDataSpeed( int dataSpeed, int prec )
31 {
32     if ( dataSpeed > 1E9 )
33         return QString::number( ( float ) dataSpeed / 1E9, 'f', prec ) + " GB/s";
34     else if ( dataSpeed > 1E6 )
35         return QString::number( ( float ) dataSpeed / 1E6, 'f', prec ) + " MB/s";
36     else if ( dataSpeed > 1E3 )
37         return QString::number( ( float ) dataSpeed / 1E3, 'f', prec ) + " KB/s";
38     else
39         return QString::number( dataSpeed ) + " B/s";
40 }
```

Б.21 Модуль Main

Листинг Б.41 — **main.cpp**

```
1  /*
2   * File:   main.cpp
3   * Author: art
4   *
5   * Created on 7 Март 2011 г., 22:14
6   */
7
8  #include <QApplication>
9  #include <QTextCodec>
10 #include "GUI/Face/face.h"
11
12 int
13 main( int argc, char *argv[] )
14 {
15     QTextCodec *codec = QTextCodec::codecForName( "UTF-8" );
16     QTextCodec::setCodecForTr( codec );
17
18     QApplication app( argc, argv );
```

```
19
20      Face * mainw = new Face( );
21
22      mainw->setWindowTitle( QObject::tr( "Just another archiver" ) );
23      mainw->show( );
24
25      return app.exec( );
26  }
```