Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет

информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Базы данных знаний и экспертные системы (ОАиП)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему

“Программное Средство Сжатия Данных На Основе

Алгоритма Хаффмана”

БГУИР КР 1-40 01 01 627 ПЗ

Студент: 751006 Шило А.В.

Руководитель:

асс. Фадеева Е.Е.

Минск 2018

**СОДЕРЖАНИЕ**

СОДЕРЖАНИЕ ..…………………………………………………………………4

ВВЕДЕНИЕ ...……………………………………………………………………. 5

1. АНАЛИЗ ПРОТОТИПОВ, ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОЕКТИРУЕМОМУ ПРОГРАММНОМУ СРЕДСТВУ ……………………………………………… 6

1.1 Сжатие Хаффмана …………………………………………………… 6

1.2 Сравнение аналогов............................................................................. 9

1.3 Постановка задачи ……………………….………………………….. 12

1.4Входные данные.................................................................................. 12

2. МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ ...…………………………………… 13

2.1 Возможные динамические структуры ……………………………... 13

2.2 Описание функциональности программного средства …………… 15

2.3 Спецификация функциональных требований …………………….. 16

3. ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА …………………. 17

4. СОЗДАНИЕ ПРОГРАММНОГО СРЕДСТВА ……………………………. 19

4.1 Описание классов и методов ………………………………………. 19

4.2 Программирование отдельных модулей ………………………….. 21

5. Тестирование, проверка работоспособности и анализ полученных результатов ..……………………………………………………………………. 29

6. РУКОВОДСТВО ПО УСТАНОВКЕ И ИСПОЛЬЗОВАНИЮ …………… 36

6.1 Руководство по установке .…………………………………………. 36

6.2 Пример использования программы …………………………... 38

ЗАКЛЮЧЕНИЕ …………………………………………………………... 40

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ ……………………........ 41

ПРИЛОЖЕНИЯ .…………………………………………………………. 42

**ВВЕДЕНИЕ**

Сжатие данных – это их преобразование для того, чтобы уменьшить занимаемый ими объем. Это нужно для рационального использования объема запоминающих устройств. Сжатие основывается на избыточности информации. Такая избыточность бывает двух видов:

* Повторение в файле фрагментов;
* Некоторые фрагменты в файле встречаются чаще других.

Так же сжатие делится на:

* Без потерь;
* С потерями.

Сжатие без потерь позволяет полностью восстанавливать исходные данные, а сжатие с потерями дает возможность восстановить данные с небольшими искажениями.

В этой курсовой работе я создам программное средство сжатия на основе алгоритма Хаффмана.

Данная пояснительная записка содержит следующие основные этапы, в ходе следования которым я постараюсь создать программное средство сжатия:

1. Анализ прототипов, литературных источников и формирование требований к проектируемому программному средству;
2. Анализ требований к программному средству и разработка функциональных требований;
3. Проектирование программного средства;
4. Создание программного средства;
5. Тестирование, проверка работоспособности и анализ полученных результатов;
6. Руководство по установке и использованию.

**1. АНАЛИЗ ПРОТОТИПОВ, ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ И ФОРМИРОВАНИЕ**

**ТРЕБОВАНИЙ К ПРОЕКТИРУЕМОМУ ПРОГРАММНОМУ СРЕДСТВУ**

* 1. **Сжатие Хаффмана**

Сжатие Хаффмана – это алгоритм эффективного кодирования информации без потерь. Идея, положенная в основу кодировании Хаффмана, основана на частоте появления символа в последовательности. Символ, который встречается в последовательности чаще всего, получает новый очень маленький код, а символ, который встречается реже всего, получает, наоборот, очень длинный код. Это нужно, так как мы хотим, чтобы, когда мы обработали весь ввод, самые частотные символы заняли меньше всего места (и меньше, чем они занимали в оригинале), а самые редкие — побольше (но так как они редкие, это не имеет значения) [5].

Так же код Хаффмана обладает свойством префиксности (то есть ни одно кодовое слово не является префиксом другого), что позволяет однозначно их декодировать [1].

Допустим, у меня есть строка символов, тогда алгоритм сжатия выглядит так:

* Создаю статистику встречаемости(вероятности) каждого символа в строке;
* Создаю бинарное дерево, в котором корень – это сумма вероятностей предыдущих двух его потомков, в свою очередь составленных из сумм их потомков и так далее;
* Прохожу по дереву от корня до каждого символа, записывая «повороты», например, при переходе в левую ветку я записываю ноль, в правую – единицу. Как только я дохожу до конечного символа, записанная мною последовательность из нулей и единиц будет кодом данного символа.

Пример: строка “Мама мыла раму”

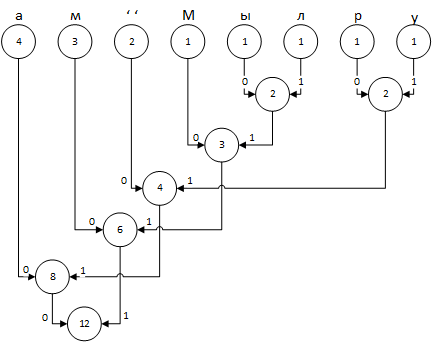
Создаю таблицу встречаемости:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| М | а | м | ы | л | р | у | ‘ ‘ |
| 1 | 4 | 3 | 1 | 1 | 1 | 1 | 2 |

Теперь сортирую данную таблицу по убыванию весов:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| а | м | ‘ ‘ | М | ы | л | р | у |
| 4 | 3 | 2 | 1 | 1 | 1 | 1 | 1 |

Теперь построю дерево по данной таблице, всегда объединяя два корня с наименьшими вероятностями:



*Рисунок 1 – Построение кодового дерева Хаффмана.*

Отсюда мы получим таблицы кодов:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| а | м | ‘ ‘ | М | ы | л | р | у |
| 00 | 10 | 010 | 110 | 1110 | 1111 | 0110 | 0111 |

Текст, зашифрованный в двоичном коде по ASCII таблице:

110011001110000011101101110000000100000111011001111101111101011111000000010000011110000111000001110110011110011 – 112 бит.

Текст, зашифрованный кодами:

110001000010101110111100010011000100111 – 39 бит.

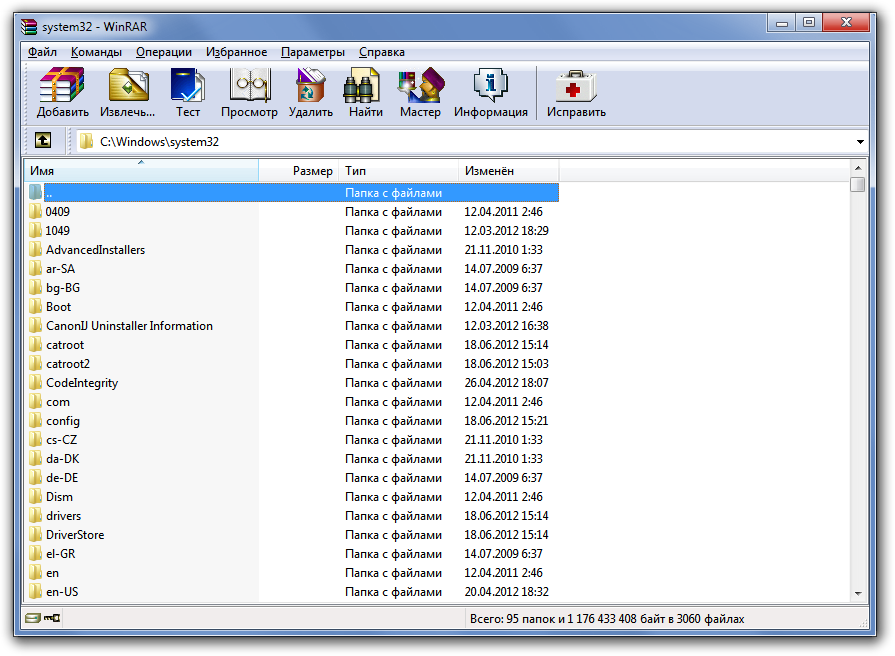
Создаётся впечатление, что данный алгоритм очень эффективен, но на практике, для расшифровки такого сообщения в его начало прибавляется таблица вероятностей (для построения дерева), ее длина и длина сообщения, что прибавляет еще 20-30 % к весу выходного файла. И тогда на практике, эффективность алгоритма сжатия Хаффмана составляет, в среднем, 20-30%[3].

* 1. **Сравнение аналогов**

Так как задание звучит как “Программное средство сжатия данных на основе

алгоритма Хаффмана.”, то я принял решение не рассматривать различные схожие алгоритмы сжатия (Арифметическое кодирование, LZW и другие), а сразу рассматривать популярные программы, использующие заданный алгоритм сжатия.

**1.2.1 WinRAR** – популярный архиватор файлов в форматы RAR и ZIP.



*Рисунок 2 – Внешний вид WinRAR*

В данном архиваторе при процессе архивации используется алгоритм Хаффмана для сжатия информации. Нужно понимать, что кроме данного алгоритма, в зависимости от настроек архивации параллельно используются и другие методы, для увеличения эффективности архивации.

**Плюсы:**

+ Скорость сжатия;

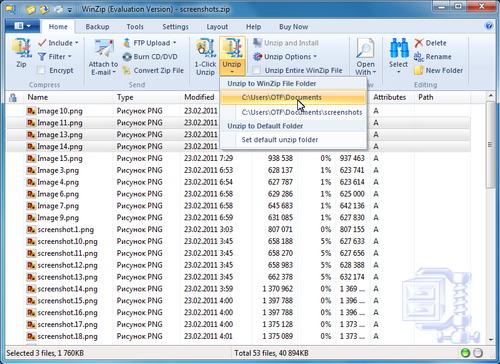
+ Вес выходного файла;

+ Минимальный, но оптимальный набор функций.

**Минусы:**

- Распространяется на платной основе.

В целом, я считаю, что это самый удобный архиватор, направленный на потребности обычных пользователей.

**1.2.2 WinZIP** – более свежий конкурент рассмотренного выше архиватора **WinRAR**, произведен компанией Corel с плотной интеграцией в оболочку Microsoft Windows.

*Рисунок 3 – Внешний вид WinZIP*

В данном архиваторе так же при архивации файлов используется метод сжатия Хаффмана. Нужно понимать, что кроме данного алгоритма, в зависимости от настроек архивации параллельно используются и другие методы, для увеличения эффективности архивации.

**Плюсы:**  
+ Многофункционален;

+ Эффективность архивации.

**Минусы:**

- Высокая стоимость;

- “Захламленный” функционал;

- Недружелюбный пользовательский интерфейс.

Я не использую данный архиватор потому что он неудобен, а так же из-за его большой стоимости.

* 1. **Постановка задачи**

Проектируемое программное средство сжатия должно эффективно уменьшать вес входных данных; так же оно должно никак не повреждать входные данные; программное средство должно уметь разжимать файлы, сжатые в данной программе; должно быть удобным и иметь максимально понятный для пользователя интерфейс; так же необходима возможность вызова помощи по данной программе; важно и отсутствие лишних функций, присутствующих в различных аналогах; необходимо наличие минимальных настроек, хотя бы возможность удалять уже заархивированные файлы. Для того, чтобы программное средство отвечало данным требованиям, в нем должны быть реализованы следующие функции:

* Сжатие файлов;
* Разжатие файлов;
* Справка по данному программному средству;
* Отображение прогресса в процессе сжатия и разжатия.

Так как данная программа не требует продвинутых функций, для ее написания был выбран язык Delphi, так как данным языком программирования я владею на достаточном уровне.

**1.4 Входные данные**

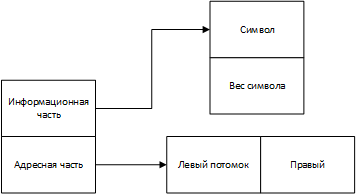
Входными данными для проектируемой программы может быть любой файл, если речь не идет о разжатии файла, так как планируемая структура данных в сжатом файле не может быть построена в других программах или получена случайно. Но, важно, чтобы файл не был изначально сжат, потому что тогда нету никакого смысла в том, чтобы сжимать его еще раз.

**2. МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ**

**2.1 Возможные динамические структуры**

Любой файл – это, по сути, массив байтов. При этом, каждый элемент данного массива содержит значение от 0 до 255. Поэтому, для алгоритма Хаффмана не важно, файл какого типа я имею на входе (текстовый, изображение и т.д.).

Мною было принято решение в качестве структуры, которая будет хранить информацию о вероятности символа и его самого, использовать дерево. В информативной части будут находится сам символ (можно использовать число типа Byte, потому что в таблице каждому символу соответствует код), и его вес (вероятность, встречаемость). В адресной части будут содержатся две ссылки: на правого и левого потомка, необходимыми для построения кодового дерева.



*Рисунок 4 – Структура данных*

Всего таких деревьев может быть (максимум) 256, то есть символ 00 – минимальный, а символ FF – максимальный (в шестнадцатеричной системе счисления).

|  |  |  |  |
| --- | --- | --- | --- |
| Node[0] | Node[1] | … | Node[255] |

*Таблица 1 – Массив деревьев*

**2.1.1 Структура нетипизированных файлов**

В ходе анализа алгоритмов и исходя из аналогов мною было принято решение принять следующую структуру сжимаемых файлов:

* + - 1. Первые четыре байта содержат размер исходного файла;
      2. Следующий байт содержит количество уникальных байтов или длину таблицу встречаемости;
      3. Далее идет таблица встречаемости – один байт на символ (его код), затем 4 байта на его «вес» в файле;
      4. Как только заканчивается таблица встречаемости – начинается зашифрованная последовательность байтов;
      5. Так как коды Хаффмана обычно меньше одного байта, то при записи закодированного потока байтов нужно использовать битовую арифметику, а именно логические операции и логические сдвиги.

**2.2 Описание функциональности программного средства**

Данное программное средство должно предоставлять возможность пользователю выбора действия: сжатия и разжатия. Так же, при нажатии необходимых кнопок (сжатия или разжатия) должно открываться диалоговое окно, в котором пользователь сможет выбрать необходимый файл. Должна присутствовать вкладка “Настройки”, в которой пользователь сможет выбрать режим работы программы, а именно – удалять ли файлы после сжатия/разжатия для экономии места на диске.

Для того, чтобы пользователю не казалось, что программа “зависла” в результате выполнения операций над файлами необходимо, чтобы в процессе сжатия или разжатия на экран выводился прогресс. Удобнее всего сделать это в виде так называемой строки прогресса (Progress Bar).

При невозможности сжатия или разжатия файла, а также после успешного завершения данных процессов уведомлять пользователя об этом.

Так же, для помощи пользователю должна быть возможность вызвать справку по данному приложению. Это должно происходить при нажатии на стандартную кнопку F1 или же при нажатии кнопки “Помощь” в меню.

Тогда:

* По нажатию на кнопку “Сжать” открывается диалоговое окно, в котором пользователь выбираем нужный ему файл;
* По нажатию на кнопку “Разжать” открывается диалоговое окно, в котором пользователь выбирает нужный ему файл;
* По нажатию на кнопку “Настройки” открывается новое окно, в котором есть Checkbox, который позволяет включить или выключить режим экономии места;
* По нажатию на клавишу F1 или на кнопку “Помощь” открывается текстовое окно с информацией, которая может помочь пользователю;
* Уведомление об ошибках или об успешном завершении процессов сжатия и разжатия.

**2.3. Спецификация функциональных требований**

Во время разработки данного программного средства должны быть реализованы следующие функции:

2.3.1. Функция сжатия файлов;

2.3.2. Функция разжатия файлов, сгенерированных в данном программном средстве;

2.3.3. Функция настроек, в которой будет присутствовать пункт, отвечающий за экономию места на хранилище;

2.3.4. Функция помощи, в которой будет описан базовый алгоритм взаимодействия пользователя с программой;

2.3.4 Функция отображения достигнутого прогресса в процессе выполнения главных операций (сжатие и разжатие);

2.3.5. Функция отображения ошибок, полученных в ходе выполнения операций;

2.3.6. Функция отображения отчета об успешном завершении главных процессов;

**3. ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА**

Схема алгоритма программы представлен в *приложении 1.* Так же он выполнен на листе, формата А1.

Ниже представлен словесный алгоритм:

3.1 Программа начинается с создания интерфейса программы;

3.2 Пользователь может выбрать один из процессов: сжатие, разжатие, настройки и помощь;

3.3 После выбора операции «сжатие» пользователь увидит диалоговое окно, в котором он сможет выбрать файл для сжатия;

3.4 Если файл занят другой программой или же недоступен для чтения, то на экран необходимо вывести сообщение об ошибке и вернутся к выбору операции;

3.5 Если файл доступен, тогда начинается процесс сжатия. Сначала собирается статистика встречаемости байт в файле. Затем собранная статистика помещается в массив. В этом массиве строится дерево кодирования, по алгоритму, расположенному в *приложении 2.* После построения дерево, производится кодировка каждого символа и запись сжатого потока байтов в файл. После завершения сжатия на экран выводится сообщение об успешном сжатии

3.6. После выбора операции «разжатие» пользователь так же увидит диалоговое окно, в котором он сможет выбрать необходимый файл;

3.7 Если файл занят другой программой или же он поврежден, на экран выведется сообщение об ошибке;

3.8 Если файл доступен, тогда начинается процесс разжатия. Сначала читается заголовок файла (О нем подробнее дальше). Заголовок записывается в таблицу встречаемости. Затем по ней так же строится кодовое дерево и начинается разжатие файла по алгоритму, расположенному в *приложении 3.* После разжатия поток байтов сохраняется в файл, с именем и разрешением, сгенерированным по исходному файлу.

3.9 После выбора операции «Настройки» должно появляться новое окно, в котором будет настройка, отвечающая за режим экономии памяти.

3.10 После выбора операции «Помощь» на экране должна появится текстовая справка по данному программному средству.

Далее будут рассмотрены некоторые словесные описания алгоритмов модулей, которые требуют пояснений:

**Алгоритм сжатия файла:**

3.1 Передаем имя сжимаемого файла в программу;

3.2 Запускаю процедуру создания статистики появления символа в файле;

3.3 Запуск процедуры создания сжимаемого файла;

3.3.1 Сортирую массив деревьев по убыванию весов;

3.3.2 Подсчет количества уникальных байт (из 256 возможных);

3.3.3 Создаем дерево (*Приложение 2*);

3.3.4 Создаем кодовые слова для кодирования входного файла (*Приложение 6*);

3.3.5 Записываем сам файл (*Приложение 4)*;

3.3.6 Очищаем дерево;

3.3.7 Очищаем статистику файла;

**Алгоритм разжатия файла:**

3.1 Чтение заголовка файла (*Приложение 5*);

3.2 Инициализирую объект со статистикой файла;

3.3 Включаю частоты байт в массив;

3.4 Создаю дерево;

3.5 Вызываю процедуру разжатия файла;

3.6 Очищаю дерево.

Подробнее данный алгоритм в *приложении 3*.

1. **Создание программного средства**

Спецификация функциональных требований и спроектированная архитектура программного средства служат фундаментом, на котором основывается выбор наиболее подходящих технологий для разработки программного средства. Успешное и обоснованное завершение данных этапов позволит создать расширяемое, надежное и функциональное приложение, призванное решать поставленные задачи.

* 1. **Описание классов и методов:**

**TParrot** – основной класс программы сжатия; название выбрано в соответсвии с общей логикой программы.

4.1.1. CreateArchClick – начать сжатие файла;

4.1.2. DeArchClick – начать разжатие файла;

4.1.3. SettingsClick – открыть настройки;

4.1.4. HelpClick – открыть помощь;

4.1.5. FormCreate – инициализация помощи.

TParrotPB – класс, необходимый для отображения строки прогресса.

TParrotSet – класс, необходимый для отображения настроек.

Решение вынести ProgressBar и Settings в разные модули было принято мною для того, чтобы в будущем я мог доработать обе эти функции, например, добавить больше настроек, изменить внешний вид ProgressBar’а.

**TWeightTable –** объект, в котором содержатся:

* Массив элементов, который содержит их количество, которые встречаются хотя бы раз;
* Процедура инициализации таблицы встречаемости;
* Процедура увеличения веса элемента.

**TFileName\_** - объект, в котором содержатся:

* Имя файла;
* Размер сжимаемого файла;
* Массив статистки частот байтов;
* Дерево кодов;
* Функцию генерации имени сжатого файла по имени исходного файла;
* Функцию генерацию имени файла по имени сжатого файла;
* Функцию определения размера файла без заголовка.

**Buffer\_ -** объект, в котором содержатся:

* Массив байт;
* Счетчик байт;
* Процедура инициализации буфера;
* Процедура вставки разжатых байтов в файл;
* Процедура записи остаточной цепочки в файл.

**4.2 Программирование отдельных модулей**

4.2.1 Процедура чтения заголовка файла:

(Блок-схема алгоритма в *приложении 9*)

**procedure** ReadHead;

**var**

b: Integer\_;

SymbolSt: **Integer**;

count\_, SymbolId, i: **Byte**;

**begin**

**try**

BlockRead(FileIn, b, 4); {Читаю в b первые четыре байта, которые должны содержать длину исходного файла}

ByteToInteger(b, ArchFile.FSize);{Переводим массив из 4 байтов в число типа Integer}

BlockRead(FileIn, count\_, 1);{Читаю в count\_ 1 байт, который должен содержать длину таблицы встречаемости}

ArchFile.FStat.create;{инициализирую статистику файлов}

ArchFile.FStat.ByteCount := count\_;{передаю в объект длину массива}

**for** i := 0 **to** ArchFile.FStat.ByteCount **do**

**begin**

BlockRead(FileIn, SymbolId, 1);{читаю в переменную символ 1 байт}

ArchFile.FStat.MainArray[i]^.Symbol := SymbolId; {заношу в таблицу символ}

BlockRead(FileIn, b, 4); {читаю в переменную 4 байта}

ByteToInteger(b, SymbolSt);{перевожу 4 байта в число типа Integer}

ArchFile.FStat.MainArray[i]^.Weight := SymbolSt; {заношу в таблицу встречаемость символа}

**end**;

CreateNode(ArchFile.Node, ArchFile.FStat.MainArray, ArchFile.FStat.ByteCount); {когда таблица построена – запускаю построение дерево по этой таблице}

CreateDeArc; {запускаю процесс разжатия файла}

DeleteNode(ArchFile.Node); {зачищаю дерево}

**except**

ShowMessage(*'С архивом что-то не так...'*);

**end**;

**end**;

4.2.2 Процедура перевода числа типа Integer в массив из 4 байт:

(Type Integer\_ = array[1..4] of Byte)

**procedure** IntegerToByte(i: **Integer**; **var** mass: Integer\_);

**var**

a: **Integer**;

b: ^Integer\_;

**begin**

b := @a;{соединяю адрес переменной а с b}

a := i;{в а помещаю значение типа integer}

mass := b^;{разыменовываю b и соединяю результат с mass}

**end**;

4.2.3 Процедура перевода массива из 4 байт в число типа Integer:

(Type Integer\_ = array[1..4] of Byte)

**procedure** ByteToInteger(mass: Integer\_; **var** i: **Integer**);

**var**

a: ^**Integer**;

b: Integer\_;

**begin**

a := @b;{соединяем адрес переменной b c a}

b := mass;{b присваиваю значение массива}

i := a^;{разыменовываю а и соединяю результат с i}

**end**;

4.2.4 Процедура записи кодов в файл:

**procedure** WriteInFile(**var** buffer: **string**);

**var**

i, j: **Integer**;

k: **Byte**;

buf: **array**[1..2 \* SizeOfBuffer] **of** **byte**;

**begin**

i := Length(buffer) **div** 8; {столько будет байтов в последовательности}

**for** j := 1 **to** i **do {**работаю с байтами от первого до последнего**}**

**begin**

buf[j] := 0;{зануляю байт, в который буду записывать сжатый код}

**for** k := 1 **to** 8 **do {**начинаю работать с битами**}**

**begin**

**if** buffer[(j - 1) \* 8 + k] = *'1'* **then**

buf[j] := buf[j] **or** (1 **shl** (8 - k));

**{**если байт в буфере =1, тогда в вспомогательном буфере с помощью логической операции “или” загоням в первый бит этого буфера 1, например – вспомогательный буфер изначально – 00000000; встречаю байт во входном буфере, равный 1, тогда 00000000 **или** 10000000 = 10000000 и так, пока не запишем в выходной буфер все целые байты**}**

**end**;

**end**;

BlockWrite(FileEx, buf, i);

Delete(buffer, 1, i \* 8);

**end**;

4.2.4 Внешняя процедура записи полного и остаточного буфера в файл:

(для того, чтобы не вызывать две процедуры, одна за одной в основном коде, я помещу в код данной процедуры вызов записи целых байт, а затем запишу остаточную цепочку)

**procedure** WriteInTFileName\_(**var** buffer: **string**);

**var**

a, k: **byte**;

**begin**

WriteInFile(buffer);{вызываю процедуру записи целого количества байт}

**if** length(buffer) >= 8 **then**

ShowMessage(*'С буффером что-то не так...'*)

**else** **if** Length(buffer) <> 0 **then**

**begin**

a := $FF; {присваиваем переменной значение 11111111}

**for** k := 1 **to** Length(buffer) **do {**просматриваю все байты, оставшиеся в буффере**}**

**if** buffer[k] = *'0'* **then**

a := a **xor** (1 **shl** (8 - k));{если встречаю 0, то с помощью логической операции “сложение по модулю два” я обнуляю необходимый бит}

BlockWrite(FileEx, a, 1);{записываю остаточную цепочку в байт}

**end**;

**end**;

Так как в начале файла записывается длина исходного файла, то во время распаковки, оставшиеся побочные 1(если, например после основной записи в буфере осталось 4 байта и они равны 1010, то в последнем байте выходного буфера будет записано следующее: 10101111, то есть, данные единицы будут лишним, но мы не можем от них избавится, так как байт не может содержать меньше 8 бит) просто не будут учтены, так как цикл распаковки будет идти ровно до последнего значащего бита.

4.2.5 Процедура сортировки массива с деревьями:

**procedure** SortMainArray(**var** a: BytesWithStat; LengthOfMass: **byte**);

**var**

i, j: **Byte**;

b: TPNode;

**begin**

**if** LengthOfMass <> 0 **then**

**for** j := 0 **to** LengthOfMass - 1 **do**

**begin**

**for** i := 0 **to** LengthOfMass - 1 **do**

**begin**

**if** a[i]^.Weight < a[i + 1]^.Weight **then**

**begin**

b := a[i];

a[i] := a[i + 1];

a[i + 1] := b;

**end**;

**end**;

**end**;

**end**;

4.2.6 Функция генерации имени сжатого файла по имени исходного файла:

**function** **TFileName\_**.ArcName: **string**;

**var**

name\_: **string**;

**const**

Exten = *'Loss'*;

**begin**

name\_ := name;{присваиваем локальной переменной значение имени файла (хранится в объекте)}

ArcName := name\_ + *'.'* + Exten;{добавляем к имени файла свое расширение}

**end**;

4.2.7 Функция генерации имени выходного файла по имени разжимаемого файла:

**function** **TFileName\_**.DeArcName: **string**;

**var**

i: **Integer**;

Name\_: **string**;

**const**

dot = *'.'*;

Exten = *'Loss'*;

**begin**

Name\_ := Name;

**if** pos(dot + Exten, Name\_) = 0 **then**

**begin**

ShowMessage(*'Имя архива должно заканчиваться на '* + dot + Exten);

**end**

**else**

**begin**

i := Length(Name\_);

**while** (i > 0) **and** (Name\_[i] <> *'.'*) **do**

**begin**

Dec(i);

**end**;

**if** i = 0 **then**

**begin**

Name\_ := copy(Name\_, 1, pos(dot + Exten, Name\_) - 1);

**if** Name\_ = *''* **then**

**begin**

ShowMessage(*'Неверное имя архива.'*);

**end**

**else**

DeArcName := Name\_;

**end**

**else**

**begin**

Delete(Name\_, pos(dot + Exten, Name\_), 5);

DeArcName := Name\_;

**end**;

**end**;

**end**;

**Список все процедур, их параметры и назначение:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя подпрограммы** | **Описание** | **Заголовок подпрограммы** | **Имя параметра** | **Назначение параметра** |
| Create; | Инициализация таблицы | procedure TWeightTable.Create; | - | - |
| Inc | Для подсчета статистики | procedure TWeightTable.Inc(i: Byte); | i | Байт (искомый) |
| ArcName: string; | функция создания имени сжатого файла по имени исходного файла | function TFileName\_.ArcName: string; | - | - |
| DeArcName: string; | функция создания имени исходного файла по имени сжатого файла | function TFileName\_.DeArcName: string; | - | - |
| FSizeWithoutHead: Integer; | функция вычисления размера файла без заголовка для сжатого файла | function TFileName\_.FSizeWithoutHead: Integer; | - | - |
| SortMainArray | обычная сортировка перестановки; соритруем по убыванию  встречаемости | procedure SortMainArray(var a: BytesWithStat; LengthOfMass: byte); | a | Таблица встречаемости |
| LengthOfMass | Длина массива |
| DeleteNode | процедура очистки дерева | procedure DeleteNode(Root: TPNode); | Root | Корень дерева |
| CreateNode | процедура создания дерева кодирования | procedure CreateNode(var Root: TPNode; MainArray: BytesWithStat;last: byte); | Root | Корневой узел |
| MainArray | Таблица со статистикой |
| last | Длина массива |
| CharStatictic | процедура подсчета статистика для байтов, встречающихся в файле хотя бы раз | procedure CharStatictic(fname: string); | fname | Название |
| Found | Поиск элементов в дереве кодирования | function Found(Node: TPNode; i: byte): Boolean; | Node | Дерево кодирования |
| i | Байт (искомый) |
| HuffCodeHelp | функция для создания кодов Хаффмана | function HuffCodeHelp(Node: TPNode; i: Byte): string; | Node | Дерево кодирования |
| i | Байт (искомый) |
| HuffCode | Поиск элементов в массиве | function HuffCode(Node: TPNode; i: Byte): string; | Node | Дерево кодирования |
| i | Байт (искомый) |
| WriteInFile | процедура записи в файл | procedure WriteInFile(var buffer: string); | buffer | Выходящий буфер закодированных байтов |
| WriteInTFileName\_ | процедура записи в файл | procedure WriteInTFileName\_(var buffer: string); | buffer | Выходящий буфер закодированных байтов |
| IntegerToByte | Перевод числа типа Integer В массив из 4 байтов | procedure IntegerToByte(i: Integer; var mass: Integer\_); | i | число |
| mass | Массив из 4-ых байт |
| ByteToInteger | Перевод массива из 4 байтов в число типа integer | procedure ByteToInteger(mass: Integer\_; var i: Integer); | mass | Массив из 4-ых байт |
| i | Число |
| CreateHead; | процедура создания заголовка файла | procedure CreateHead; | - | - |
| buffer\_.CreateBuf; | инициализация переменных | procedure buffer\_.CreateBuf; | - | - |
| buffer\_.InsertByte | процедура вставки разжатых файлов в файл | procedure buffer\_.InsertByte(a: Byte); | a | Разжатый байт |
| Buffer\_.FlushBuf; | запись остаточных байтов в файл | procedure Buffer\_.FlushBuf; | - | - |
| CreateDeArc; | процедура создания разжатого файла | procedure CreateDeArc; | - | - |
| ReadHead; | процедура чтения заголовка сжатого файла | procedure ReadHead; | - | - |
| ExtractFile; | внешняя процедура разжатия файла | procedure ExtractFile; | - | - |
| CreateArchiv; | процедура создания сжатого файла | procedure CreateArchiv; | - | - |
| CreateFile; | внешняя процедура создания сжатого файла | procedure CreateFile; | - | - |
| RunEncodeHaff | конечная процедура по созаднию сжимаемого файла | procedure RunEncodeHaff(FileName\_: string); | FileName\_ | Название |
| RunDecodeHaff | конечная процедура по созданию разжатого файла | procedure RunDecodeHaff(FileName\_: string); | FileName\_ | Название |
| CreateArchClick | реакция программы на нажатие кнопки меню "Сжать файл" | procedure TParrot.CreateArchClick(Sender: TObject); | Sender | Объект, который сгенерировал событие |
| DeArchClick | реакция программы на нажатие кнопки меню "Разжать файл" | procedure TParrot.DeArchClick(Sender: TObject); | Sender | Объект, который сгенерировал событие |
| SettingsClick | реакция программы на нажатие кнопки меню "Настройки" | procedure TParrot.SettingsClick(Sender: TObject); | Sender | Объект, который сгенерировал событие |
| HelpClick | реакция программы на нажатие кнопки меню "Помощь" | procedure TParrot.HelpClick(Sender: TObject); | Sender | Объект, который сгенерировал событие |
| FormCreate | создание интерфейса помощи | procedure TParrot.FormCreate(Sender: TObject); | Sender | Объект, который сгенерировал событие |

**5. Тестирование, проверка работоспособности и анализ полученных результатов**

Для того, чтобы соответствовать требованиям к проектируемому программному средству, необходимо, чтобы оно прошло некоторое тестирование, способное выявить его недостатки и проверить его на устойчивость к неправильным действиям пользователя.

Ниже будут приведены различные тесты, которые приближены к действиям пользователя.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| №  Теста | Тестируемая функциональность | Последовательность  действий | Ожидаемый результат | Полученный результат |
| 1. | Сжатие файла. | Запустить приложение. Нажать на кнопку меню ”Сжать”. Выбрать файл, не занятый в другой программе и открыть его. | На экране появляется строка состояния процесса. После достижения ею 100% появляется сообщение об успешном завершении процесса. В директории, где находился сжимаемый файл должен появится новый файл с разрешением \*.Loss, размер которого будет отличатся от размера исходного файла на 0-80% в зависимости от структуры и содержания исходного файла. Если в настройках выставлена функция экономии места, то исходный файл после сжатия должен быть удален. | Тест пройден успешно. |
| 2. | Разжатие файла. | Запустить приложение. Нажать на кнопку меню ”Разжать”. Выбрать файл, не занятый в другой программе; так же файл должен быть создан в данном программном средстве, а не просто переименован из другого файла. | На экране появляется строка состояния процесса. После достижения ею 100% появляется сообщение об успешном завершении процесса. В директории, где находился разжимаемый файл должен появится новый файл с разрешением исходного файла, размер которого будет равен размеру исходного файла. Если в настройках выставлена функция экономии места, то исходный файл после разжатия должен быть удален. | Тест пройден успешно. |
| 3. | Открытие и изменение настроек. Функция сжатия с измененными настройками. | Запустить приложение. Нажать на кнопку меню ”Настройки”.  По умолчанию настройка экономии места должна быть включен. Выключить данную функцию. Закрыть окно настроек и запустить процесс сжатия файла. | На экране должно появится окно настроек, в котором будет распологатся компонент CheckBox, и небольшое пояснение к работе функции. После изменения настройки закрыть окно. После нажатия на кнопку меню “Сжать файл” начинается процесс сжатия. После завершения на экран выводится сообщение об успешном завершении процесса. В директории с исходным файлом появляется сжатый файл, но оригинальный файл остается на месте. | Тест пройден успешно. |
| 4. | Открытие и изменение настроек. Функция разжатия с измененными настройками. | Запустить приложение. Нажать на кнопку меню ”Настройки”.  По умолчанию настройка экономии места должна быть включен. Выключить данную функцию. Закрыть окно настроек и запустить процесс разжатия файла. | На экране должно появится окно настроек, в котором будет распологатся компонент CheckBox, и небольшое пояснение к работе функции. После изменения настройки закрыть окно. После нажатия на кнопку меню “Разжать файл” начинается процесс разжатия. После завершения на экран выводится сообщение об успешном завершении процесса. В директории с исходным файлом появляется разжатый файл, но оригинальный файл остается на месте. | Тест пройден успешно. |
| 5. | Открытие помощи по нажатию кнопки меню “Помощь”. | Запустить приложение.  Нажать на кнопку меню “Помощь”. | На экран выводится текст, в котором кратко описывается руководство пользователя.  Оно легко пролистывается вниз с помощью ScrollBar, находящегося слева от окна с текстовой информацией. | Тест пройден успешно. |
| 6. | Открытие помощи с помощью клавиши клавиатуры F1. | Запустить приложение. Нажать на кнопку клавиатуры F1. | На экран выводится текст, в котором кратко описывается руководство пользователя.  Оно легко пролистывается вниз с помощью ScrollBar, находящегося слева от окна с текстовой информацией. | Тест пройден успешно. |
| 7. | Открытие помощи. Ее закрытие. Сжатие файла. Повторное открытие и закрытие помощи. | Запустить приложение. Вызвать помощь любым из возможных способов. Затем нажать на кнопку меню “Сжать файл”. Идет процесс сжатия файла, описанный выше. Затем снова открыть помощь. | На экран выводится текст, в котором кратко описывается руководство пользователя.  Оно легко пролистывается вниз с помощью ScrollBar, находящегося слева от окна с текстовой информацией.  После нажатия кнопки “Сжать файл” начинается процесс сжатия, а окно с текстом исчезает. После успешного завершения сжатия и вызова помощи текстовое окно снова появляется без каких-либо изменений. | Тест пройден успешно. |
| 8. | Открытие помощи. Ее закрытие. Разжатие файла. Повторное открытие и закрытие помощи. | Запустить приложение. Вызвать помощь любым из возможных способов. Затем нажать на кнопку меню “Разжать файл”. Идет процесс разжатия файла, описанный выше. Затем снова открыть помощь. | На экран выводится текст, в котором кратко описывается руководство пользователя.  Оно легко пролистывается вниз с помощью ScrollBar, находящегося слева от окна с текстовой информацией.  После нажатия кнопки “Разжать файл” начинается процесс Разжатия, а окно с текстом исчезает. После успешного завершения разжатия и вызова помощи текстовое окно снова появляется без каких-либо изменений. | Тест пройден успешно. |
| 9. | Открытие диалогового окна и выход из него без выбора файла. | Запустить программу. Нажать на кнопку меню “Сжатие файла”. После появления диалогового окна не выбирать файл, а выйти из него. | Программа будет работать в нормальном ключе. | Тест пройден успешно. |
| 10. | Открытие диалогового окна и выход из него без выбора файла. | Запустить программу. Нажать на кнопку меню “Разжатие файла”. После появления диалогового окна не выбирать файл, а выйти из него. | Программа будет работать в нормальном ключе. | Тест пройден успешно. |
| 11. | Открытие файла, открытого в другое программе. | Запустить программу. Нажать на кнопку меню “Сжатие файла”. В диалоговом окне выбора файлов выбрать файл, занятый в другой программе. | После выбора файла на экран должно вывестись соответствующее сообщение об ошибке. | Тест пройден успешно. |
| 12. | Открытие файла для процедуры разжатия, с неверным разрешением (не \*.Loss). | Запустить программу. Нажать на кнопку “Разжатие файла”. Выбрать нужный файл. | После выбора файла на экран должно вывестись соответствующее сообщение об ошибке. | Тест пройден успешно. |
| 13. | Попытка сжатия файла с большим весом.(100 Мб < Size < 3.5 Гб). | Запустить программу. Нажать на кнопку “Сжать файл”. Открыть необходимый файл. | После выбора файла должна начаться процедура сжатия, описанная выше.  Одновременно с этим должен появится ProgressBar и отображать прогресс процесса. После достижения им 100% должно выводится сообщение о успешном завершении процесса. | Тест пройден успешно, но время занимаемое процессом сжатия удручает. |
| 14. | Попытка разжатия файла с большим весом.(100 Мб < Size < 3.5 Гб). | Запустить программу. Нажать на кнопку “Разжать файл”. Открыть необходимый файл. | После выбора файла должна начаться процедура разжатия, описанная выше.  Одновременно с этим должен появится ProgressBar и отображать прогресс процесса. После достижения им 100% должно выводится сообщение о успешном завершении процесса. | Тест пройден успешно, но время занимаемое процессом разжатия лучше, чем у сжатия, но все равно далеко от идеала. |

*Таблица 2 – Тестирование программы.*

Программа довольно неплохо прошла процесс тестирования, но последнее выявило некоторые недостатки, которые будут исправлены в будущих версиях. Программа соответствует требованиям, выставленных ранее и имеет полный функционал, в соответствии с поставленной задачей.

**6. Руководство по установке и использованию**

6.1 Основные требования для запуска данного программного средства:

* ОС: Версия Microsoft Windows от XP и выше;
* Процессор: Pentium® III 800 МГц или AMD Athlon;
* RAM: От 128 Мб;
* Место на диске: от 3 Мб свободного места.

Исходя из данных требований следует, что данная программа может запускаться практически на любом компьютере.

**6.1 Руководство по установке**

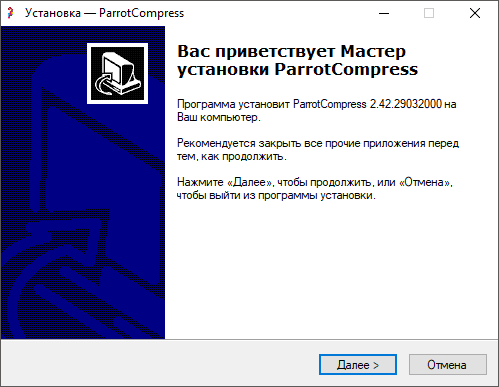
**6.1.1 Открытие файла**

Для установки данной программы на компьютер сначала необходимо запустить установочный файл “*ParrotSetup.exe*”.

После запуска необходимо пройти по всем этапам установки: от принятия лицензии (Все права на данную программу принадлежат БГУИР) до выбора директории, куда установить данную программу.

После завершения установки на рабочем столе появится ярылк приложения, нажав на который два раза левой кнопкой мышки запустится приложение.

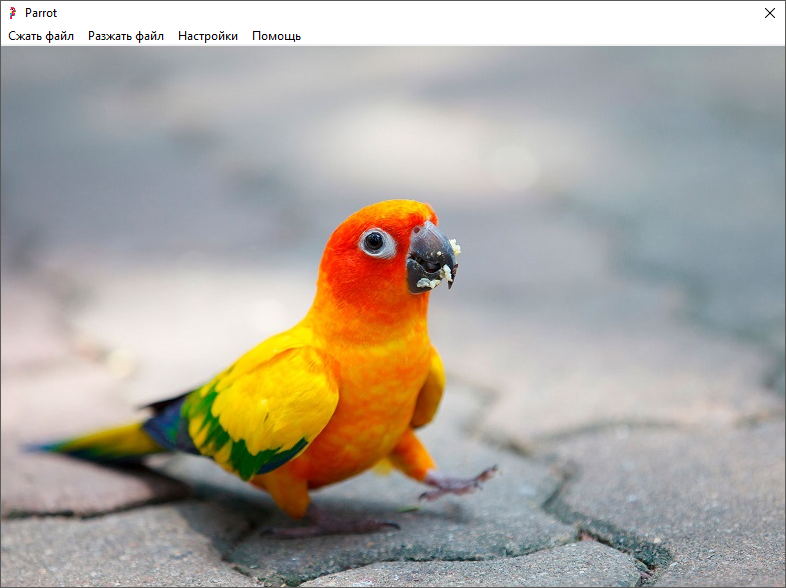
*Рисунок 5 – Окно установщика программы.*



**6.1.2 Внешний вид программы**

После запуска программы на экране появится интерфейс этого приложения:

*Рисунок 6 – Интерфейс программы*



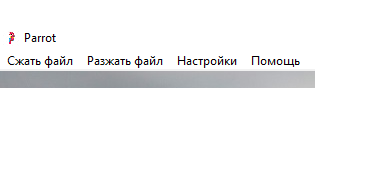
После открытия данного интерфейса программа готова к работе. Для получения полной справки по использованию этой программы необходимо нажать на кнопку “Помощь” среди кнопок главного меню приложения.

Данный интерфейс был выбран исходя из эстетических соображений автора программы, для того, чтобы сделать его более дружелюбным для пользователя, а также сделать его максимально простым и удобным в использовании.

**6.2 Пример использования программы**

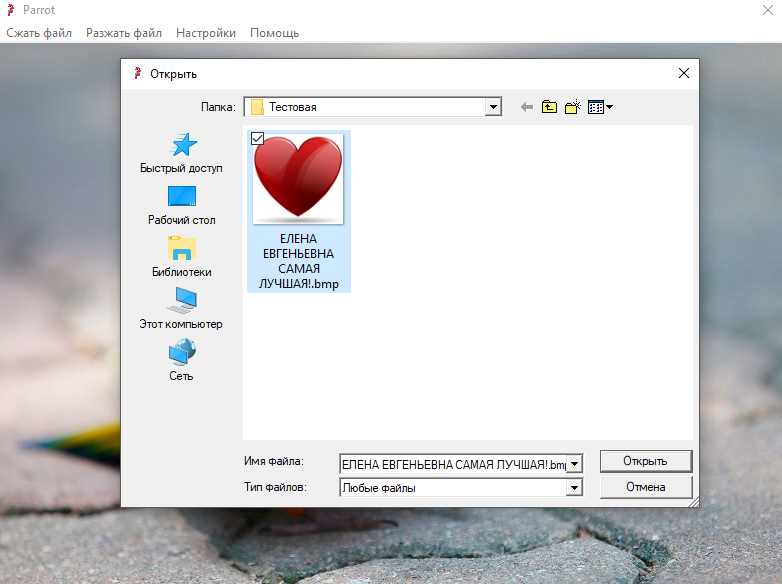
Например, если я хочу сжать файл, то следующие действия буду таковыми:

6.2.1. Нажать на кнопку “Сжать файл”;



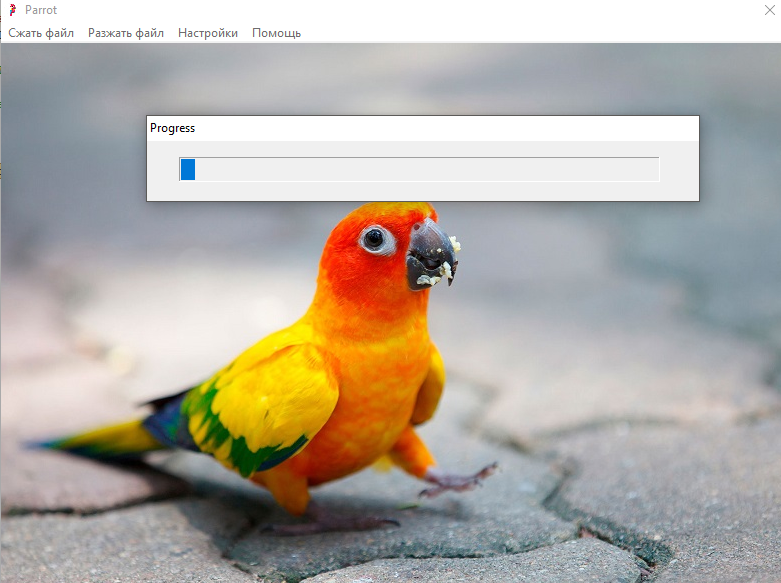
*Рисунок 7 – Кнопки главного меню*

6.2.2. В открывшемся диалоговом окне открыть нужный файл:



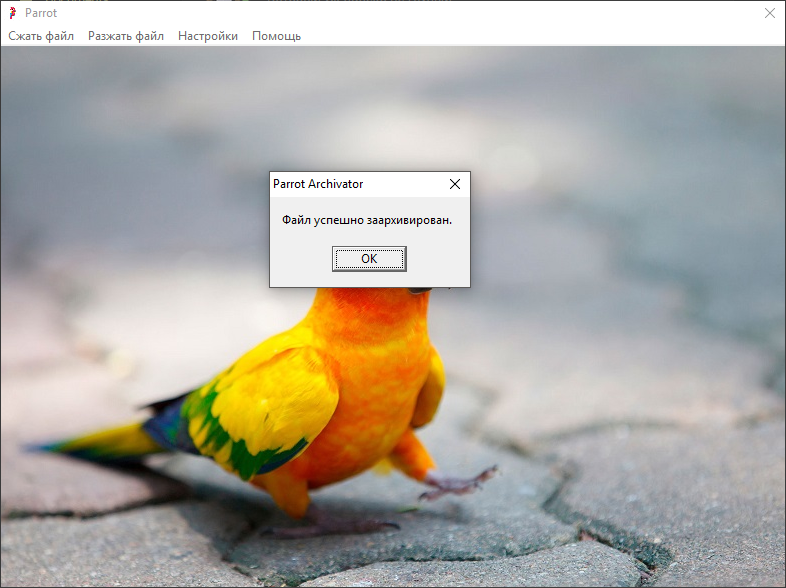
*Рисунок 8 – Выбор файла в диалоговом окне*

6.2.3. Начинается процесс сжатия, а его прогресс отображается в отдельном окне:



*Рисунок 9 – Отображение прогресс процесса*

6.2.4. После успешного завершения сжатия об этом появляется сообщение:



*Рисунок 10 – Отображение сообщения об успешном завершении процесса*

**ЗАКЛЮЧЕНИЕ**

В ходе анализа существующих аналогов, их недостатков и достоинств, изучения алгоритмов, необходимых для реализации программного средства, были установлены необходимые требования и определенный функционал для проектируемой программы, которые должны быть учтены при проектировании программного средства.

На основе функциональных требований было проведено проектирование программного средства, в котором представлена разработка алгоритмов отдельных модулей программы и программы в целом.

После построения самого программного средства были сформулированы и проведены необходимые тесты, целью которых было выявление недостатков в программе, а также устойчивость к неверным действиям пользователя.

На основе тестов мною было принято решение в будущем доработать программу и исправить такие вещи, как: скорость работы алгоритма для больших данных; большее количество настроек, позволяющих оптимизировать процесс работы с данными.

Так же в будущем будет внедрен дополнительные алгоритмы сжатия, для того, чтобы улучшить эффективность данного программного средства и довести до уровня существующих аналогов.

Главной целью при разработке программы было поставлено покрытие главного недостатка существующих средств сжатия данных – недружелюбный пользовательский интерфейс и “захламленность” ненужными функциями, а также создание универсального программного средства сжатия на основе алгоритма Хаффмана.

В ходе работы над курсовым проектом была достигнута большая часть поставленных задач, но, без сомнений, данное программное средство требует небольших доработок и улучшений, которые будут внесены мною в ходе сопровождения данной программы в будущем.

**СПИСОК ЛИТЕРАТУРЫ**

[1] Wikipedia.org [Электронный ресур]. – Электронные данные. – Режим доступа: <https://ru.wikipedia.org/wiki/Код_Хаффмана>

[2] Delphibasics.ru [Электронный портал]. – Электронные данные. – Режим доступа: <http://www.delphibasics.ru/>

[3] А. Я. Архангельский. 100 Компонентов Delphi

[4] Сэломон Д. Сжатие данных, изображений и звука

[5] habr.com [Электронный ресур]. – Электронные данные. – Режим доступа:

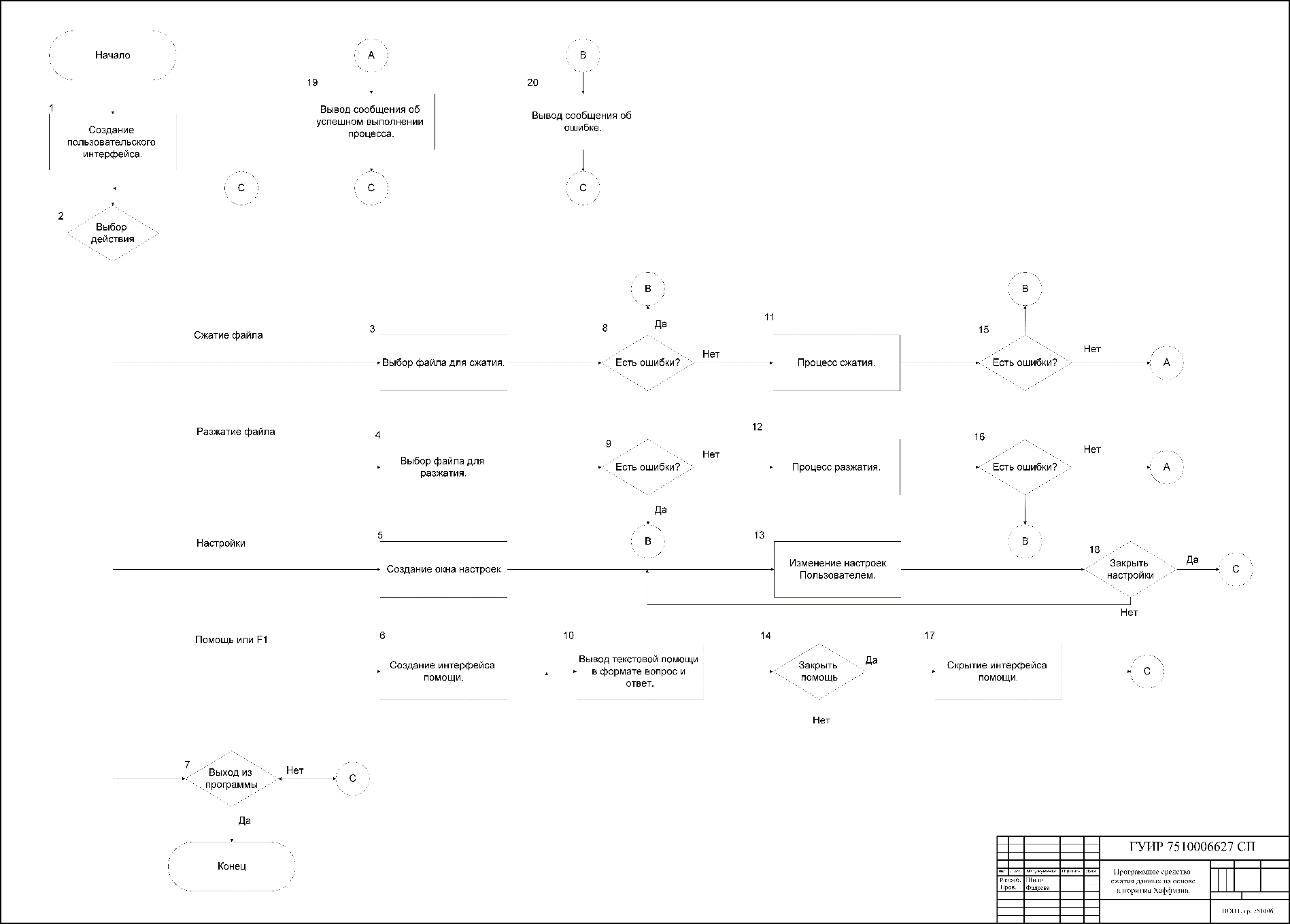
<https://habr.com/post/144200/>

[6] wikiznanie.ru [Электронный ресур]. – Электронные данные. – Режим доступа: <http://www.wikiznanie.ru/wikipedia/index.php/Код_Хаффмана>

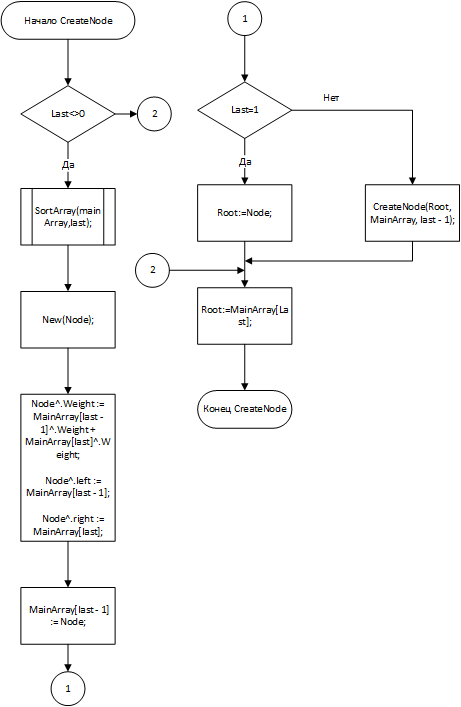
[7] Симаков А. Код Хафммана [Статья]

**ПРИЛОЖЕНИЯ**

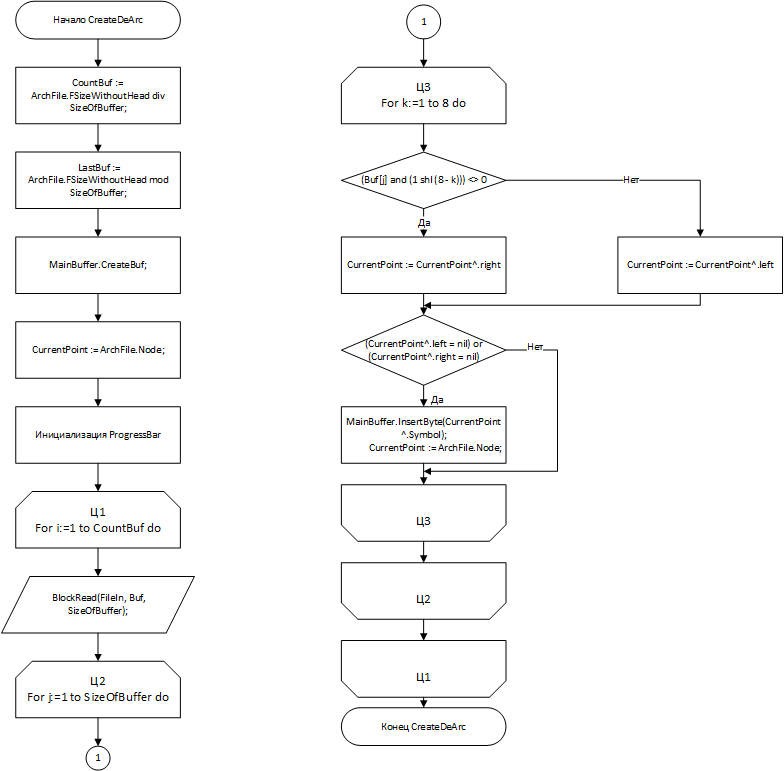
*Приложение 1 (схема алгоритма программы на А1)*

**

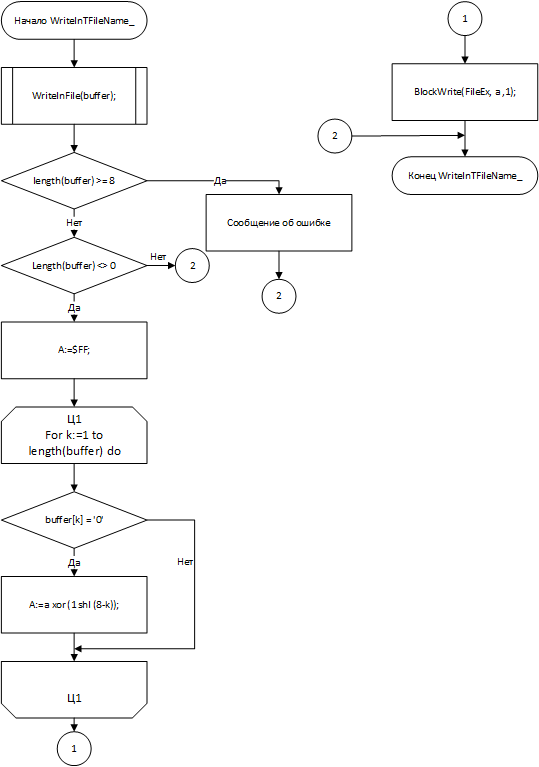
*Приложение 2 (алгоритм создания дерева кодирования Хаффмана)*

**

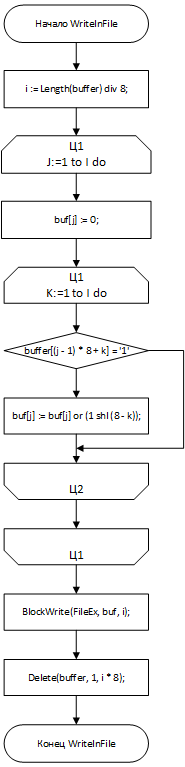
*Приложение 3 (алгоритм разжатия файла)*

**

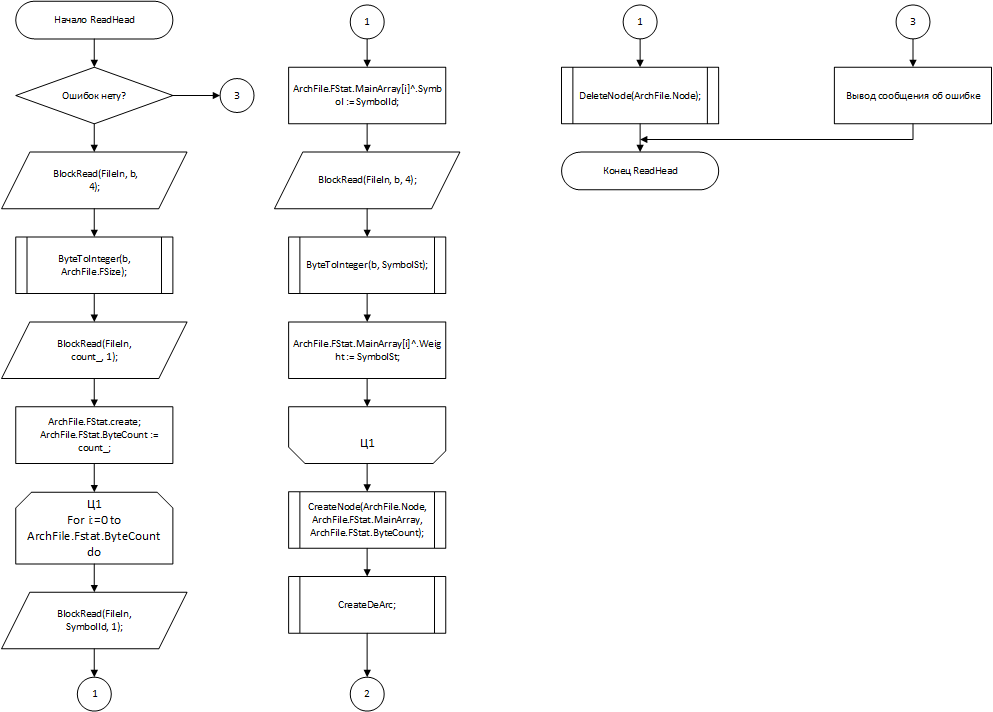
*Приложение 4 (алгоритм записи в файл)*

**

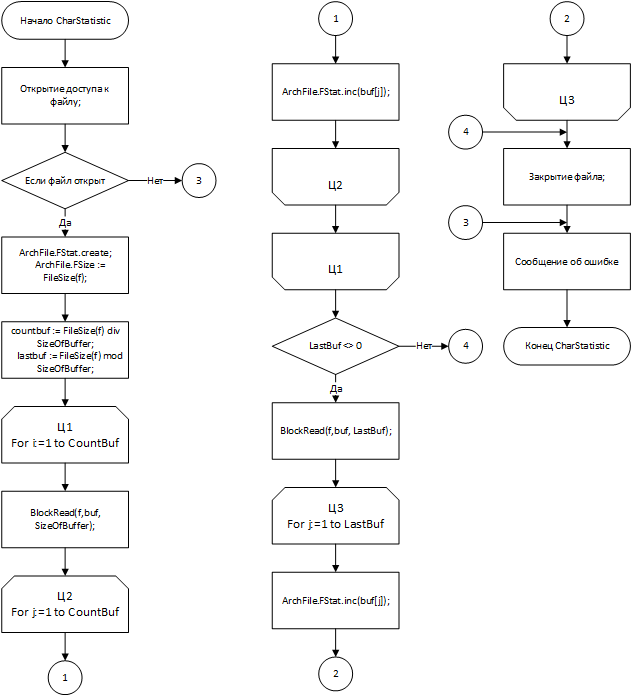
*Дополнение к приложению 4*

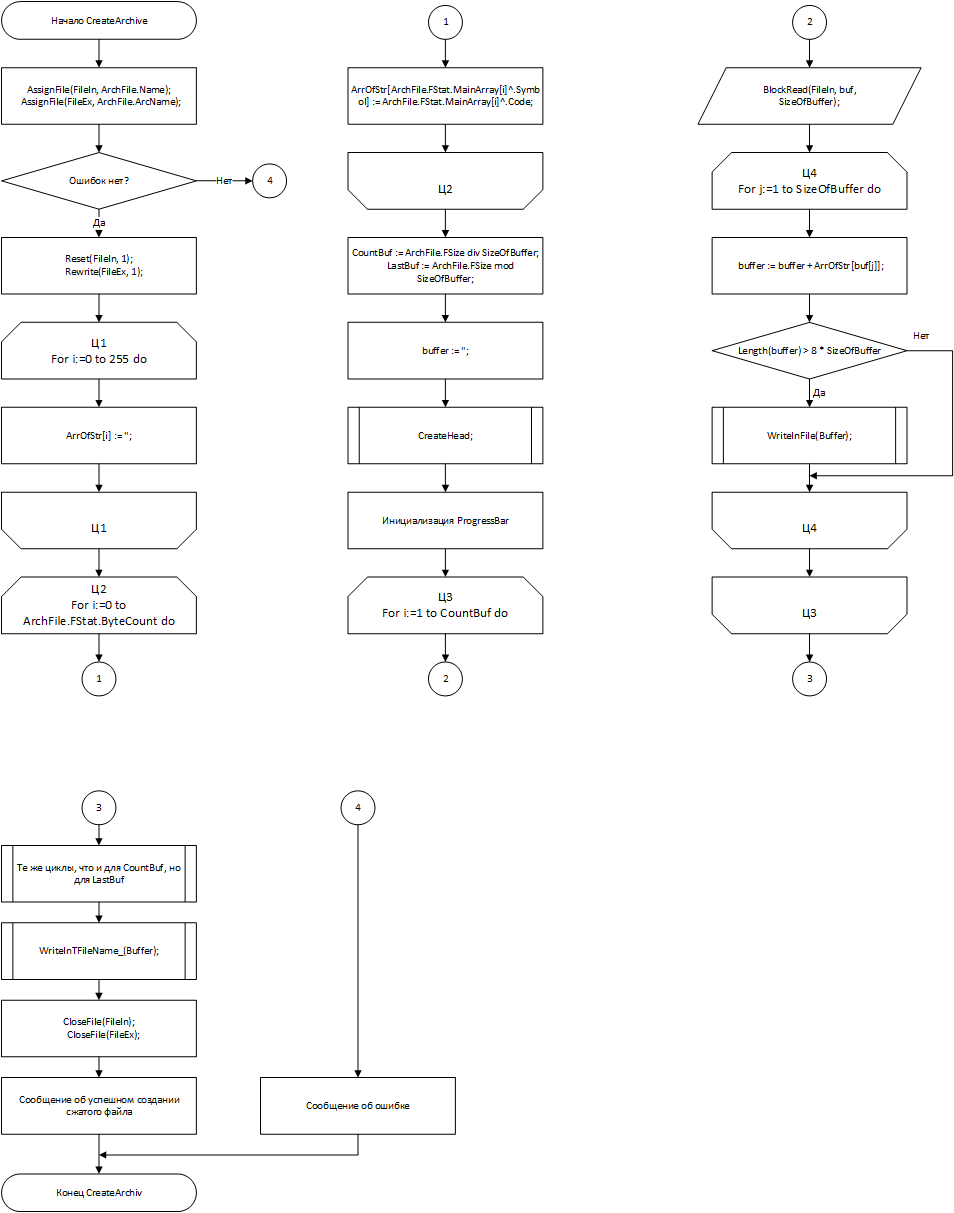
**

*Приложение 5 (алгоритм чтения заголовка файла)*

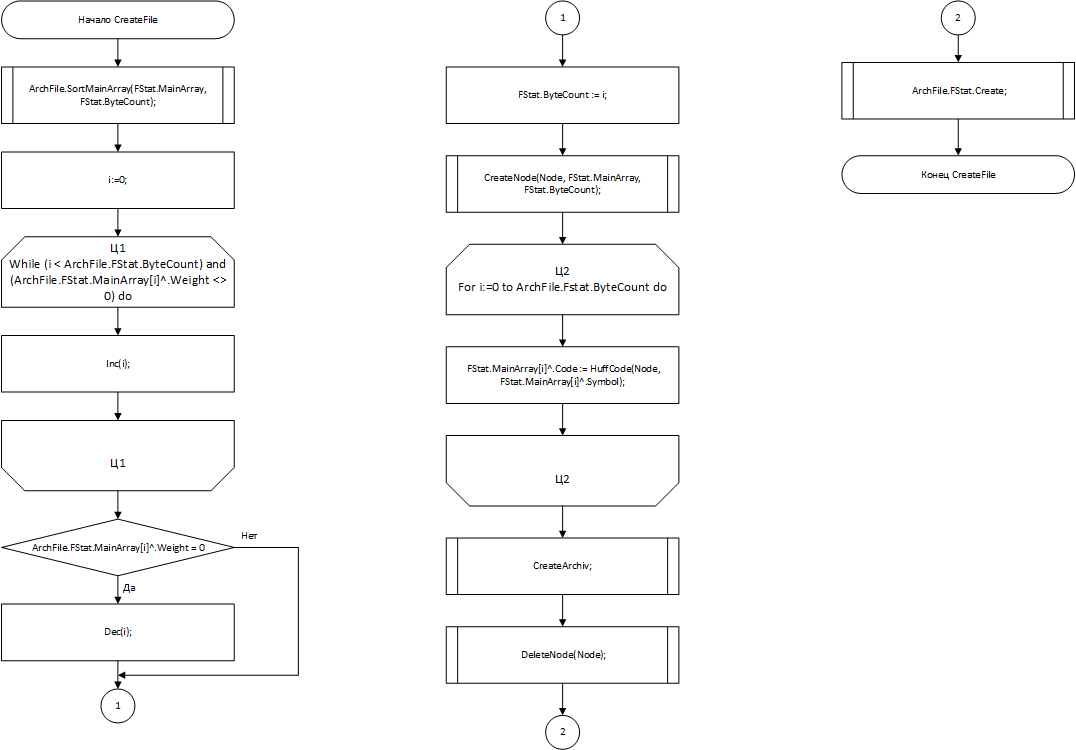
**

*Алгоритм подсчета статистики байт в файле*

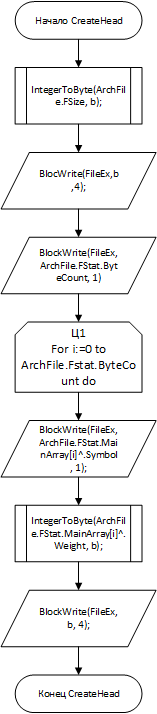
**

*Алгоритм создания сжатого файла*

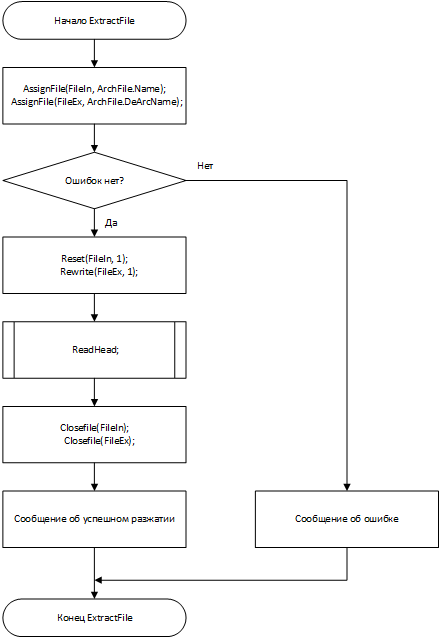
*Алгоритм создания файла*

**

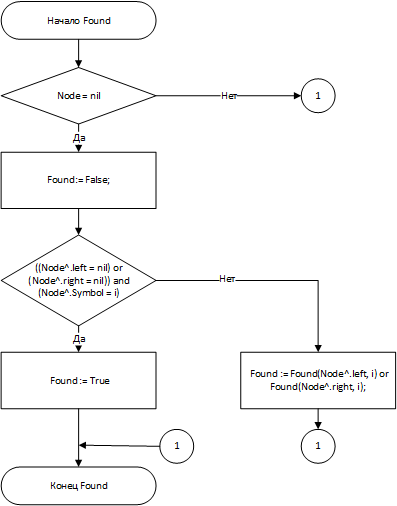
*Алгоритм создания заголовка файла*

**

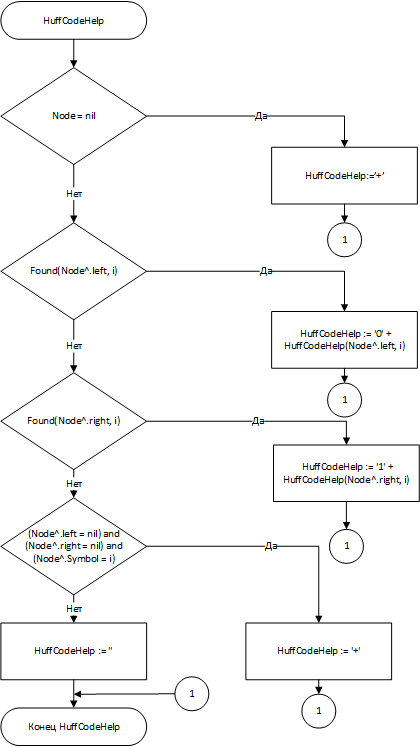
*Алгоритм разжатия файла*

**

*Алгоритм поиска кодов для символа (1)*

**

*Алгоритм для генерации кодов Хаффмана для символов*

**

*Приложение А (Код программы)*

**unit** ParrotMain;

**interface**

**uses**

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, StdCtrls, Menus, ExtCtrls, jpeg;

**const**

SizeOfBuffer = 4096;

**type**

TParrot = **class**(TForm)

OpenDialog1: TOpenDialog;

BackgroundOf: TImage;

MainMenuOf: TMainMenu;

CreateArch: TMenuItem;

DeArch: TMenuItem;

Help: TMenuItem;

Settings: TMenuItem;

OutText: TMemo;

OpenDialog2: TOpenDialog;

**procedure** CreateArchClick(Sender: **TObject**);

**procedure** DeArchClick(Sender: **TObject**);

**procedure** SettingsClick(Sender: **TObject**);

**procedure** HelpClick(Sender: **TObject**);

**procedure** FormCreate(Sender: **TObject**);

**private**

*{ Private declarations }*

**public**

*{ Public declarations }*

**end**;

**var**

Parrot: TParrot;

FileIn, FileEx: **file**;

**implementation**

**uses**

ParrotBar, ParrotSettings;

*{$R \*.dfm}*

**type**

*{Дерево, хранит символ, его "вес" и код. Так же хранит ссылки на левую и правую веточки}*

TPNode = ^PNode;

PNode = **record**

Symbol: **Byte**;

Weight: **Integer**;

Code: **string**;

left, right: TPNode;

**end**;

*{Массив таких деревьев. Нужен для создания заголовка файла}*

BytesWithStat = **array**[0..255] **of** TPNode;

*{Объект, необходимый для создания заголовка и инициализации переменных, также для подсчета статистики}*

TWeightTable = **object**

MainArray: BytesWithStat;

ByteCount: **byte**;

**procedure** Create;

**procedure** Inc(i: **Byte**);

**end**;

*{-------------------------------------------------------------------------------*

*Процедура: TWeightTable.Create*

*Автор: Алексей*

*Дата: 2018.05.13*

*Входные параметры: Нет*

*Результат: Инициализация таблицы встречаемости*

*-------------------------------------------------------------------------------}*

**procedure** **TWeightTable**.Create;

**var**

i: **Byte**;

**begin**

ByteCount := 255;

**for** i := 0 **to** ByteCount **do**

**begin**

**New**(MainArray[i]);

**with** MainArray[i]^ **do**

**begin**

Symbol := i;

Weight := 0;

left := **nil**;

right := **nil**;

**end**;

**end**;

**end**;

*{-------------------------------------------------------------------------------*

*Процедура: TWeightTable.Inc*

*Автор: Алексей*

*Дата: 2018.05.13*

*Входные параметры: i: Byte - Байт, встречаемость которого мы увеличиваем*

*Назначение : увеличить встречаемость байта*

*-------------------------------------------------------------------------------}*

**procedure** **TWeightTable**.Inc(i: **Byte**);

**begin**

MainArray[i]^.Weight := MainArray[i]^.Weight + 1;

**end**;

**type**

TFileName\_ = **object**

Name: **string**;

FSize: **Integer**;

FStat: TWeightTable;

Node: TPNode;

**function** ArcName: **string**;

**function** DeArcName: **string**;

**function** FSizeWithoutHead: **Integer**;

**end**;

*{фнкция создания имени сжатого файла по имени исходного файла}*

**function** **TFileName\_**.ArcName: **string**;

**var**

i: **Integer**;

name\_: **string**;

**const**

Exten = *'Loss'*;

**begin**

name\_ := name;

ArcName := name\_ + *'.'* + Exten;

**end**;

*{функция создания имени исходного файла по имени сжатого файла}*

**function** **TFileName\_**.DeArcName: **string**;

**var**

i: **Integer**;

Name\_: **string**;

**const**

dot = *'.'*;

Exten = *'Loss'*;

**begin**

Name\_ := Name;

**if** pos(dot + Exten, Name\_) = 0 **then**

**begin**

ShowMessage(*'Имя архива должно заканчиваться на '* + dot + Exten);

Application.Terminate;

**end**

**else**

**begin**

i := Length(Name\_);

**while** (i > 0) **and** (Name\_[i] <> *'.'*) **do**

**begin**

Dec(i);

**end**;

**if** i = 0 **then**

**begin**

Name\_ := copy(Name\_, 1, pos(dot + Exten, Name\_) - 1);

**if** Name\_ = *''* **then**

**begin**

ShowMessage(*'Неверное имя архива.'*);

Application.Terminate;

**end**

**else**

DeArcName := Name\_;

**end**

**else**

**begin**

Delete(Name\_, pos(dot + Exten, Name\_), 5);

DeArcName := Name\_;

**end**;

**end**;

**end**;

*{функция вычисления размера файла без заголовка для разжатия}*

**function** **TFileName\_**.FSizeWithoutHead: **Integer**;

**begin**

FSizeWithoutHead := FileSize(FileIn) - 4 - 1 - (FStat.ByteCount + 1) \* 5;

*{FileSize - стандартная функция, возвращающая размера файла в байтах,*

*отнимаем 4 - потому что размер исходного файла записывается в 4 байтах,*

*1 байт на количество уникальных символов, и еще минус длина таблицы}*

**end**;

*{обычная сортировка перестановками; сортирую по убыванию частоты символа}*

**procedure** SortMainArray(**var** a: BytesWithStat; LengthOfMass: **byte**);

**var**

i, j: **Byte**;

b: TPNode;

**begin**

**if** LengthOfMass <> 0 **then**

**for** j := 0 **to** LengthOfMass - 1 **do**

**begin**

**for** i := 0 **to** LengthOfMass - 1 **do**

**begin**

**if** a[i]^.Weight < a[i + 1]^.Weight **then**

**begin**

b := a[i];

a[i] := a[i + 1];

a[i + 1] := b;

**end**;

**end**;

**end**;

**end**;

*{процедура очистки дерева}*

**procedure** DeleteNode(Root: TPNode);

**begin**

**if** Root <> **nil** **then**

**begin**

DeleteNode(Root^.left);

DeleteNode(Root^.right);

**Dispose**(Root);

Root := **nil**;

**end**;

**end**;

*{процедура создания дерева кодирования}*

**procedure** CreateNode(**var** Root: TPNode; MainArray: BytesWithStat; last: **byte**);

**var**

Node: TPNode; *{элемент дерева}*

**begin**

**if** last <> 0 **then** *{пока длина массива не ноль}*

**begin**

SortMainArray(MainArray, last); *{соритруем}*

**new**(Node);

Node^.Weight := MainArray[last - 1]^.Weight + MainArray[last]^.Weight;*{присваиваем весу нового узла вес двух его родителей}*

Node^.left := MainArray[last - 1]; *{оставляю ссылку на левого родителя}*

Node^.right := MainArray[last]; *{оставляю ссылку на правого родителя}*

MainArray[last - 1] := Node; *{помещаю на место предпоследнего элемента*

*новый узел, зачищая последние два}*

**if** last = 1 **then**

**begin**

Root := Node;*{ если остался один узел, то делаю его корнем}*

**end**

**else**

**begin**

CreateNode(Root, MainArray, last - 1); *{если нет, то рекурсивно вызываю*

*создание дерева дальше}*

**end**;

**end**

**else**

Root := MainArray[last]; *{если длина изначально была 0, то есть в файле был один символ (чередовался), то сразу задаю корень из этого элемента }*

**end**;

**var**

ArchFile: TFileName\_; *{сжимаемый файл}*

*{процедура подсчета статистика для байтов, встречающихся в файле хотя бы 1 раз}*

**procedure** CharStatictic(fname: **string**);

**var**

f: **file**;

i, j: **Integer**;

buf: **array**[1..SizeOfBuffer] **of** **Byte**;

countbuf, lastbuf: **Integer**;

**begin**

AssignFile(f, fname);

**try**

Reset(f, 1);

ArchFile.FStat.create;*{инициализирую статистику}*

ArchFile.FSize := FileSize(f);

countbuf := FileSize(f) **div** SizeOfBuffer;*{количество целых буферов}*

lastbuf := FileSize(f) **mod** SizeOfBuffer;*{остаточный буфер}*

**for** i := 1 **to** countbuf **do**

**begin**

BlockRead(f, buf, SizeOfBuffer);*{читаю из файла в буфер 4Кб символов}*

**for** j := 1 **to** SizeOfBuffer **do**

**begin**

ArchFile.FStat.inc(buf[j]);*{подсчет статистики байтов}*

**end**;

**end**;

*{дособираю статистику не целого буфера}*

**if** lastbuf <> 0 **then**

**begin**

BlockRead(f, buf, lastbuf);

**for** j := 1 **to** lastbuf **do**

**begin**

ArchFile.FStat.inc(buf[j]);

**end**;

**end**;

CloseFile(f);

**except**

ShowMessage(*'Файл не доступен (Скорее всего занят каким-то другим приложением).'*)

**end**;

**end**;

*{Функция HuffCodeHelp вызывает данную функцию с параметрами: левый потомок и символ, если символ есть в вызванной ветке, то Found возвращает True; если нет, то False}*

**function** Found(Node: TPNode; i: **byte**): **Boolean**;

**begin**

**if** (Node = **nil**) **then**

Found := **False***{если дерево пустое, то false}*

**else**

**begin**

**if** ((Node^.left = **nil**) **or** (Node^.right = **nil**)) **and** (Node^.Symbol = i) **then**

Found := **True** *{Если в дереве один узел, то он найден}*

**else**

Found := Found(Node^.left, i) **or** Found(Node^.right, i);*{Если не один узел, то ищу рекурсивно}*

**end**;

**end**;

*{функция для создания кодов хаффмана}*

**function** HuffCodeHelp(Node: TPNode; i: **Byte**): **string**;

**begin**

**if** (Node = **nil**) **then**

HuffCodeHelp := *'+'* *{если в дереве пусто, возвращаю символ +}*

**else**

**begin**

**if** (Found(Node^.left, i)) **then**

HuffCodeHelp := *'0'* + HuffCodeHelp(Node^.left, i)*{если найдено в левом, то записываю в код 0, и идем дальше}*

**else**

**begin**

**if** Found(Node^.right, i) **then**

HuffCodeHelp := *'1'* + HuffCodeHelp(Node^.right, i)*{если найдено в правом, то записываю в код 1, и иду дальше}*

**else**

**begin**

**if** (Node^.left = **nil**) **and** (Node^.right = **nil**) **and** (Node^.Symbol = i) **then**

HuffCodeHelp := *'+'* *{если символ найден, то записываю в конец кода символ +}*

**else**

HuffCodeHelp := *''*; *{если нет, то ничего не пишу}*

**end**;

**end**;

**end**;

**end**;

*{эта функция вызывает прошлую, но еще и учитывает тот случай, когда у нас один единственный символ}*

**function** HuffCode(Node: TPNode; i: **Byte**): **string**;

**var**

s: **string**;

**begin**

s := HuffCodeHelp(Node, i);

s := s;

**if** (s = *'+'*) **then**

HuffCode := *'0'*

**else**

HuffCode := Copy(s, 1, length(s) - 1); *{если все хорошо, то удаляю + из конца кода}*

**end**;

*{процедура записи в файл; перевод строки с байтами в биты}*

**procedure** WriteInFile(**var** buffer: **string**);

**var**

i, j: **Integer**;

k: **Byte**;

buf: **array**[1..2 \* SizeOfBuffer] **of** **byte**;

**begin**

i := Length(buffer) **div** 8; *{узнаю количество целых байтов}*

**for** j := 1 **to** i **do**

**begin**

buf[j] := 0; *{зануляю текущий элемент буфера}*

**for** k := 1 **to** 8 **do** *{тут начинается работа с битами}*

**begin**

**if** buffer[(j - 1) \* 8 + k] = *'1'* **then** *{если в входящем буфере первым байтом встречается*

*единичка, то с помощью логического сдвига влево единицы и логической операции "или" я записываю в первый бит байта выходящего буфера единицу}*

buf[j] := buf[j] **or** (1 **shl** (8 - k));

**end**;

**end**;

BlockWrite(FileEx, buf, i);

Delete(buffer, 1, i \* 8);*{удаляю уже записанные байты}*

**end**;

*{процедура записи в файл, но уже с учетом остаточной цепочки байт (<8)}*

**procedure** WriteInTFileName\_(**var** buffer: **string**);

**var**

a, k: **byte**;

**begin**

WriteInFile(buffer);*{записываю целые байты}*

**if** length(buffer) >= 8 **then**

ShowMessage(*'С буффером что-то не так...'*)

**else** **if** Length(buffer) <> 0 **then**

**begin**

a := $FF;*{делаю в текущем выходном буфере все 1 (11111111)}*

**for** k := 1 **to** Length(buffer) **do**

**if** buffer[k] = *'0'* **then**

a := a **xor** (1 **shl** (8 - k)); *{если во входящем буфере с байтами*

*встречаются 0, то я их записываю в советующие биты выходящего*

*буфера с помощью операции xor и логического сдвига влево}*

BlockWrite(FileEx, a, 1);

**end**;

**end**;

**type**

Integer\_ = **array**[1..4] **of** **Byte**;

*{структура данных, необходимы для побайтовой записи в файл}*

**procedure** IntegerToByte(i: **Integer**; **var** mass: Integer\_);

**var**

a: **Integer**;

b: ^Integer\_;

**begin**

b := @a;*{соединяю адрес переменной а с b}*

a := i;*{в а помещаю значение типа integer}*

mass := b^;*{разыменовываю b и соединяю результат с mass}*

**end**;

*{}*

**procedure** ByteToInteger(mass: Integer\_; **var** i: **Integer**);

**var**

a: ^**Integer**;

b: Integer\_;

**begin**

a := @b;*{соединяем адрес переменной b c a}*

b := mass;*{b присваиваю значение массива}*

i := a^;*{разыменовываю а и соединяю результат с i}*

**end**;

*{процедура создания заголовка файла}*

**procedure** CreateHead;

**var**

b: Integer\_;

i: **Byte**;

**begin**

IntegerToByte(ArchFile.FSize, b);

BlockWrite(FileEx, b, 4);

BlockWrite(FileEx, ArchFile.FStat.ByteCount, 1);

**for** i := 0 **to** ArchFile.FStat.ByteCount **do**

**begin**

BlockWrite(FileEx, ArchFile.FStat.MainArray[i]^.Symbol, 1);

IntegerToByte(ArchFile.FStat.MainArray[i]^.Weight, b);

BlockWrite(FileEx, b, 4);

**end**;

**end**;

**const**

MaxCount = 4096;

**type**

buffer\_ = **object**

ArrOfByte: **array**[1..MaxCount] **of** **Byte**;

ByteCount: **Integer**;

GeneralCount: **Integer**;

**procedure** CreateBuf;

**procedure** InsertByte(a: **Byte**);

**procedure** FlushBuf;

**end**;

*{инициализация перенных}*

**procedure** **buffer\_**.CreateBuf;

**begin**

ByteCount := 0;

GeneralCount := 0;

**end**;

*{процедура вставки разжатых файлов в файл}*

**procedure** **buffer\_**.InsertByte(a: **Byte**);

**begin**

**if** GeneralCount < ArchFile.FSize **then**

**begin**

inc(ByteCount);

inc(GeneralCount);

ArrOfByte[ByteCount] := a;

**if** ByteCount = MaxCount **then**

**begin**

BlockWrite(FileEx, ArrOfByte, ByteCount);

ByteCount := 0;

**end**;

**end**;

**end**;

*{запись остаточных байтов в файл}*

**procedure** **Buffer\_**.FlushBuf;

**begin**

**if** ByteCount <> 0 **then**

BlockWrite(FileEx, ArrOfByte, ByteCount);

**end**;

*{процедура создания разжатого файла}*

**procedure** CreateDeArc;

**var**

i, j: **Integer**;

k: **Byte**;

Buf: **array**[1..SizeOfBuffer] **of** **Byte**;

CountBuf, LastBuf: **Integer**;

MainBuffer: buffer\_;

CurrentPoint: TPNode;

**begin**

CountBuf := ArchFile.FSizeWithoutHead **div** SizeOfBuffer;*{считаю количество целых буферов по 4кб}*

LastBuf := ArchFile.FSizeWithoutHead **mod** SizeOfBuffer;*{считаю остаточные байты}*

MainBuffer.CreateBuf;*{инициализирую буфер}*

CurrentPoint := ArchFile.Node;*{устанавливаю указатель на корень дерева разжатия}*

ParrotPB.ProgressBar.Min := 1;

ParrotPB.ProgressBar.Max := CountBuf;

ParrotPB.Visible := **True**;

**for** i := 1 **to** CountBuf **do**

**begin**

ParrotPB.ProgressBar.Position := i;

BlockRead(FileIn, Buf, SizeOfBuffer);

**for** j := 1 **to** SizeOfBuffer **do**

**begin**

**for** k := 1 **to** 8 **do**

**begin**

**if** (Buf[j] **and** (1 **shl** (8 - k))) <> 0 **then**

CurrentPoint := CurrentPoint^.right

**else**

CurrentPoint := CurrentPoint^.left;

*{в каждом байте я просматриваю биты, если встречаю единичку - иду по*

*дереву направо, нет - налево. Когда дохожу до символа, то запускаю вставку*

*разжатых байтов в файл; тоже самое происходит и для остаточных байтов}*

**if** (CurrentPoint^.left = **nil**) **or** (CurrentPoint^.right = **nil**) **then**

**begin**

MainBuffer.InsertByte(CurrentPoint^.Symbol);

CurrentPoint := ArchFile.Node;

**end**;

**end**;

**end**;

**end**;

ParrotPB.Visible := **False**;

**if** LastBuf <> 0 **then**

**begin**

BlockRead(FileIn, Buf, LastBuf);

**for** j := 1 **to** LastBuf **do**

**begin**

**for** k := 1 **to** 8 **do**

**begin**

**if** (Buf[j] **and** (1 **shl** (8 - k))) <> 0 **then**

CurrentPoint := CurrentPoint^.right

**else**

CurrentPoint := CurrentPoint^.left;

**if** (CurrentPoint^.left = **nil**) **or** (CurrentPoint^.right = **nil**) **then**

**begin**

MainBuffer.InsertByte(CurrentPoint^.Symbol);

CurrentPoint := ArchFile.Node;

**end**;

**end**;

**end**;

**end**;

MainBuffer.FlushBuf;

**end**;

*{процедура чтения заголовка сжатого файла}*

**procedure** ReadHead;

**var**

b: Integer\_;

SymbolSt: **Integer**;

count\_, SymbolId, i: **Byte**;

**begin**

**try**

BlockRead(FileIn, b, 4);

ByteToInteger(b, ArchFile.FSize);

*{считываю размер исходного файла}*

BlockRead(FileIn, count\_, 1);*{считываю длину таблицы}*

ArchFile.FStat.create;*{инициализирую статистику}*

ArchFile.FStat.ByteCount := count\_;

**for** i := 0 **to** ArchFile.FStat.ByteCount **do**

**begin**

BlockRead(FileIn, SymbolId, 1);

ArchFile.FStat.MainArray[i]^.Symbol := SymbolId;*{первый байт - символ, следующие за ним 4 байта - встречаемость и так до конца файла}*

BlockRead(FileIn, b, 4);

ByteToInteger(b, SymbolSt);

ArchFile.FStat.MainArray[i]^.Weight := SymbolSt;

**end**;

CreateNode(ArchFile.Node, ArchFile.FStat.MainArray, ArchFile.FStat.ByteCount);*{строю дерево кодирования Хаффмана}*

CreateDeArc;*{разжимаю файл}*

DeleteNode(ArchFile.Node);*{зачищаю дерево}*

**except**

ShowMessage(*'С архивом что-то не так...'*);

**end**;

**end**;

*{внешняя процедура разжатия файла}*

**procedure** ExtractFile;

**begin**

AssignFile(FileIn, ArchFile.Name);

AssignFile(FileEx, ArchFile.DeArcName);

**try**

Reset(FileIn, 1);

Rewrite(FileEx, 1);

ReadHead;

Closefile(FileIn);

Closefile(FileEx);

ShowMessage(*'Файл успешно разархивирован.'*);

**except**

ShowMessage(*'Ошибка в извлечении файла.'*);

**end**;

**end**;

*{процедура создания сжатого файла}*

**procedure** CreateArchiv;

**var**

buffer: **string**;

ArrOfStr: **array**[0..255] **of** **string**;

i, j: **Integer**;

buf: **array**[1..SizeOfBuffer] **of** **Byte**;

CountBuf, LastBuf: **Integer**;

**begin**

AssignFile(FileIn, ArchFile.Name);

AssignFile(FileEx, ArchFile.ArcName);

**try**

Reset(FileIn, 1);

Rewrite(FileEx, 1);

**for** i := 0 **to** 255 **do***{инициализирую массив кодов}*

ArrOfStr[i] := *''*;

**for** i := 0 **to** ArchFile.FStat.ByteCount **do**

**begin**

ArrOfStr[ArchFile.FStat.MainArray[i]^.Symbol] := ArchFile.FStat.MainArray[i]^.Code;

*{помещаю в массив строк коды Хаффмана, соответствующие символам}*

**end**;

CountBuf := ArchFile.FSize **div** SizeOfBuffer;*{количество целых буферов, которые*

*будут в сжимаемом файле}*

LastBuf := ArchFile.FSize **mod** SizeOfBuffer;*{остаток этих байт}*

buffer := *''*;

CreateHead;*{создаю заголовок}*

ParrotPB.ProgressBar.Min := 1;

ParrotPB.ProgressBar.Max := CountBuf;

ParrotPB.Visible := **True**;

**for** i := 1 **to** CountBuf **do**

**begin**

ParrotPB.ProgressBar.Position := i;

BlockRead(FileIn, buf, SizeOfBuffer);

**for** j := 1 **to** SizeOfBuffer **do**

**begin**

buffer := buffer + ArrOfStr[buf[j]];

**if** Length(buffer) > 8 \* SizeOfBuffer **then** *{8 \* 4096 - это количество в битах;*

*если будет превышена данная длина, то нужно записывать буфер в файл}*

WriteInFile(buffer);

**end**;

**end**;

ParrotPB.Visible := **False**;

**if** LastBuf <> 0 **then**

**begin**

BlockRead(FileIn, buf, LastBuf);

**for** j := 1 **to** LastBuf **do**

**begin**

buffer := buffer + ArrOfStr[buf[j]];

**if** Length(buffer) > 8 \* SizeOfBuffer **then**

WriteInFile(buffer);

**end**;

**end**;

WriteInTFileName\_(buffer ); *{остаточная цепочка}*

CloseFile(FileIn);

CloseFile(FileEx);

ShowMessage(*'Файл успешно заархивирован.'*)

**except**

ShowMessage(*'Ошибка создания архива.'*);

**end**;

**end**;

*{внешняя процедура создания сжатого файла}*

**procedure** CreateFile;

**var**

i: **Byte**;

**begin**

**with** ArchFile **do**

**begin**

SortMainArray(FStat.MainArray, FStat.ByteCount);*{соритрую масив}*

i := 0;

**while** (i < FStat.ByteCount) **and** (FStat.MainArray[i]^.Weight <> 0) **do**

**begin**

Inc(i);*{тут я ищу количество задействованных байтов}*

**end**;

**if** FStat.MainArray[i]^.Weight = 0 **then**

Dec(i);*{уменьшаю на один, чтобы попасть на последний элемент}*

FStat.ByteCount := i;

CreateNode(Node, FStat.MainArray, FStat.ByteCount);*{строю дерево кодирования Хаффмана}*

**for** i := 0 **to** FStat.ByteCount **do**

FStat.MainArray[i]^.Code := HuffCode(Node, FStat.MainArray[i]^.Symbol);

*{загоняю в таблицу коды Хаффмана}*

CreateArchiv;*{создаю заголовок и записываю поток сжатых байтов в файл}*

DeleteNode(Node);*{зачищаю дерево}*

ArchFile.FStat.Create;*{зачищаю таблицу}*

**end**;

**end**;

*{конечная процедура по созаднию сжимаемого файла, с учетом имени выбранного файла}*

**procedure** RunEncodeHaff(FileName\_: **string**);

**begin**

ArchFile.Name := FileName\_;

CharStatictic(ArchFile.Name);

CreateFile;

**end**;

*{конечная процедура по созданию разжатого файла, с учетом имени}*

**procedure** RunDecodeHaff(FileName\_: **string**);

**begin**

ArchFile.name := FileName\_;

ExtractFile;

**end**;

*{реакция программы на нажатие кнопки меню "Сжать файл"}*

**procedure** **TParrot**.CreateArchClick(Sender: **TObject**);

**begin**

OutText.Visible := **false**;

OpenDialog1.Filter := *'Любые файлы|'*;

**if** OpenDialog1.Execute **then**

**begin**

RunEncodeHaff(OpenDialog1.FileName);

**end**;

**if** ParrotSet.CheckBox2.checked **then**

DeleteFile(OpenDialog1.Filename);

**end**;

*{реакция программы на нажатие кнопки меню "Разжать файл"}*

**procedure** **TParrot**.DeArchClick(Sender: **TObject**);

**begin**

OutText.Visible := **false**;

OpenDialog1.Filter := *'Архив(.Loss)|\*.Loss'*;

**if** OpenDialog1.Execute **then**

**begin**

RunDecodeHaff(OpenDialog1.FileName);

**end**;

**if** ParrotSet.CheckBox2.checked **then**

DeleteFile(OpenDialog1.Filename);

**end**;

*{реакция программы на нажатие кнопки меню "Настройки"}*

**procedure** **TParrot**.SettingsClick(Sender: **TObject**);

**begin**

OutText.Visible := **false**;

ParrotSet.Visible := **True**;

**end**;

*{реакция программы на нажатие кнопки меню "Помощь"}*

**procedure** **TParrot**.HelpClick(Sender: **TObject**);

**begin**

OutText.Visible := **True**;

**end**;

*{создание интерфейса помощи}*

**procedure** **TParrot**.FormCreate(Sender: **TObject**);

**var**

f: **TextFile**;

line: **string**;

**begin**

OutText.Clear;

OutText.Visible := **false**;

AssignFile(f, *'Help.LossHelp'*);

Reset(f);

**while** **not** (Eof(f)) **do**

**begin**

readln(f, line);

OutText.Lines.Add(line);

**end**;

CloseFile(f);

**end**;

**end**.

*Приложение В (Код модуля ParrotSet)*

**unit** ParrotSettings;

**interface**

**uses**

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, StdCtrls;

**type**

TParrotSet = **class**(TForm)

CheckBox2: TCheckBox;

**private**

*{ Private declarations }*

**public**

*{ Public declarations }*

**end**;

**var**

ParrotSet: TParrotSet;

**implementation**

*{$R \*.dfm}*

**end**.

*Приложение C (Код модуля ParrotBar)*

**unit** ParrotBar;

**interface**

**uses**

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, ComCtrls, StdCtrls;

**type**

TParrotPB = **class**(TForm)

ProgressBar: TProgressBar;

**private**

*{ Private declarations }*

**public**

*{ Public declarations }*

**end**;

**var**

ParrotPB: TParrotPB;

**implementation**

*{$R \*.dfm}*

**end**.