МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Белгородский государственный технологический университет
имени В.Г. Шухова

Институт ИЭИТУС
Кафедра Информационных технологий

Курсовой проект
по дисциплине информатика на тему:

«Разработка программы для сжатия и распаковки файлов методом равномерного кодирования «VDPROTO»

Выполнил студент группы ИТ- 22
Давыдов В.О.
Проверил
ст.преп.Рога С.Н.
ст.преп. Четвериков А.В.

Белгород 2018

Оглавление

Введение....3

1.Постановка задачи и основных требований к разрабатываемому программному обеспечению «VDPROTO»....4

1.1. Постановка задачи....4

1.2. Основания для разработки....4

1.3. Назначение программного средства «VDPROTO»....4

1.4. Требования к программному средству «VDPROTO»....4

1.4.1. Требования к функциональным характеристикам....4

1.4.2. Требования к надежности....5

1.4.3. Требования к составу и параметрам технических средств....5

1.5. Требования к условиям эксплуатации....5

1.6. Требования к программной документации....5

2. Проектирование программного средства и реализация программы «VDPROTO» 5

2.1. Разработка алгоритмов программы....6

2.2. Описание структур и функций....22

2.4. Описание функций....22

Заключение....32

Список литературы....33

Приложение 1 Листинг программы....34

Приложение 2 Руководство пользователя....43

Введение

В настоящее время происходит бурное развитие информационных технологий, которые с каждым годом всё больше и больше входят в жизнь людей. Информация становится важнейшим ресурсом в данной области. В связи с этим проблема защиты информации становится всё более актуальной. Один из надёжных способов защиты — кодирование, которое поможет скрыть важные данные, даже при попадании их в руки злоумышленников.

1.Постановка задачи и основных требований к разрабатываемому программному обеспечению «VDPROTO»

1.1. Постановка задачи

Основание для проведения разработки является задание по курсовому проектированию, которое предполагает разработку программного средства, согласно своему варианту:

Программа должна запускать с командной строки с тремя параметрами: имя входного файла, имя выходного файла и режим работы (кодирование или декодирование). При кодировании входной файл — текстовый или двоичный, выходной — двоичный. Структура выходного файла: тип входного файла, количество различных символов в тексте; количество бит, задействованных в последнем байте; массив структур, содержащих сведения о символах, их кодах и т.д. (продумать самостоятельно) и собственно байты с закодированными символами. При декодировании входной файл — двоичный, выходной — текстовый или двоичный. Расширение декодированного файла должно соответствовать исходному файлу.

1.2. Основания для разработки

Программное средство разрабатывается на основе учебного плана кафедры «Информационные технологии» для специальности 09.03.02 «Информационные системы и технологии» по дисциплине «Информатика».

1.3. Назначение программного средства «VDPROTO»

Программа представляет собой средство кодирования данных и предназначена для защиты информации пользователя.

1.4. Требования к программному средству «VDPROTO»

1.4.1. Требования к функциональным характеристикам

1.4.1.1. Программа должна обеспечивать возможность выполнения следующих функций

- выбор режима работы — кодирование или декодирование;

- запуск с командной строки с тремя параметрами: имя входного файла, имя выходного файла и режим работы (кодирование или декодирование).
- проведение процесса кодирования или декодирования;
- вывод сообщения при возникновении ошибок

1.4.1.2. Исходные данные

- Режим работы;
- Имя входного файла;
- Имя выходного файла;

1.4.2. Требования к надежности

Предусмотреть контроль вводимой пользователем информации. Программа должна функционировать корректно, в соответствии с разработанным алгоритмом.

1.4.3. Требования к составу и параметрам технических средств

Необходимо наличие ПК с ОС Windows. Дисковое пространство должно быть не менее 200 Кб, объем свободной оперативной памяти - не менее 700 Кбайт. Наличие манипулятора типа “клавиатура”.

1.5. Требования к условиям эксплуатации

В ОС Windows должен быть выставлен русский язык для программ, не поддерживающих Unicode. Исправно функционирующие комплектующие компьютера.

1.6. Требования к программной документации

Необходимы спецификации программ, содержащие описания целей создания переменных и блок-схемы.

Разрабатываемые программные модули должны быть самодокументированы, т.е. тексты программ должны содержать все необходимые комментарии.

2. Проектирование программного средства и реализация программы «VDPROTO»

2.1. Разработка алгоритмов программы

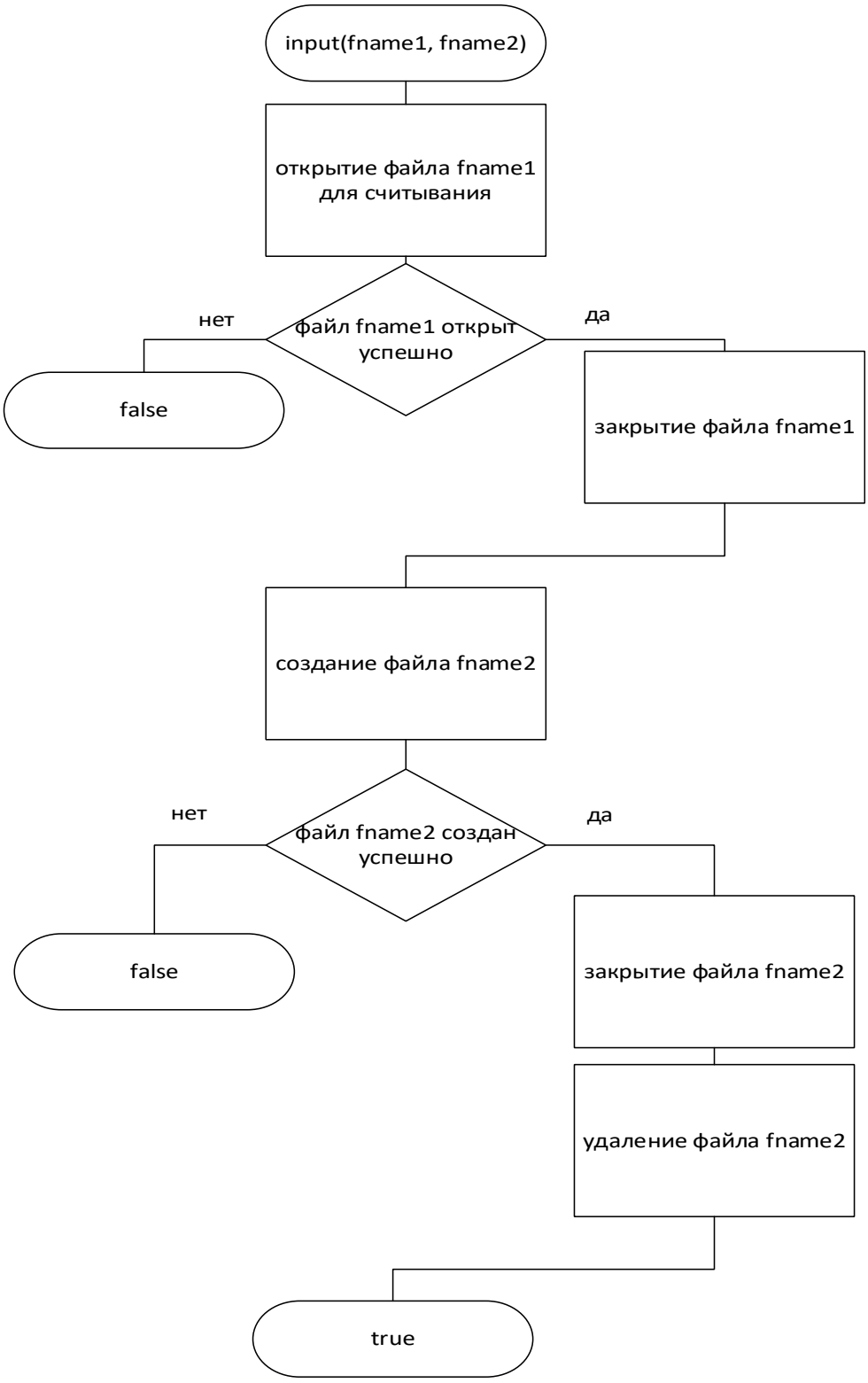

Рисунок 1 Функция, проверяющая правильность ввода имён файлов

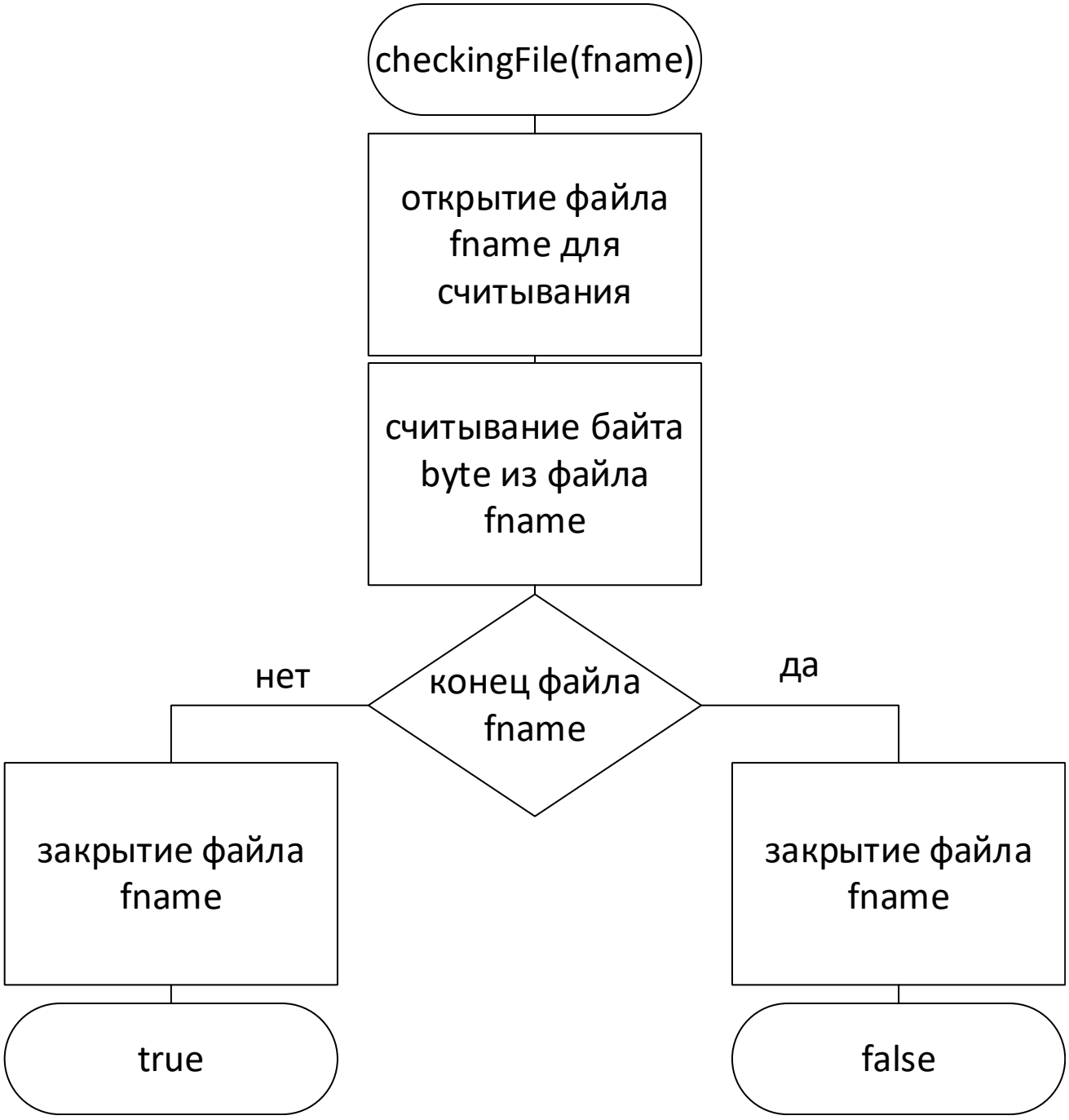

Рисунок 2 Функция, проверяющая наличие данных во входном файле

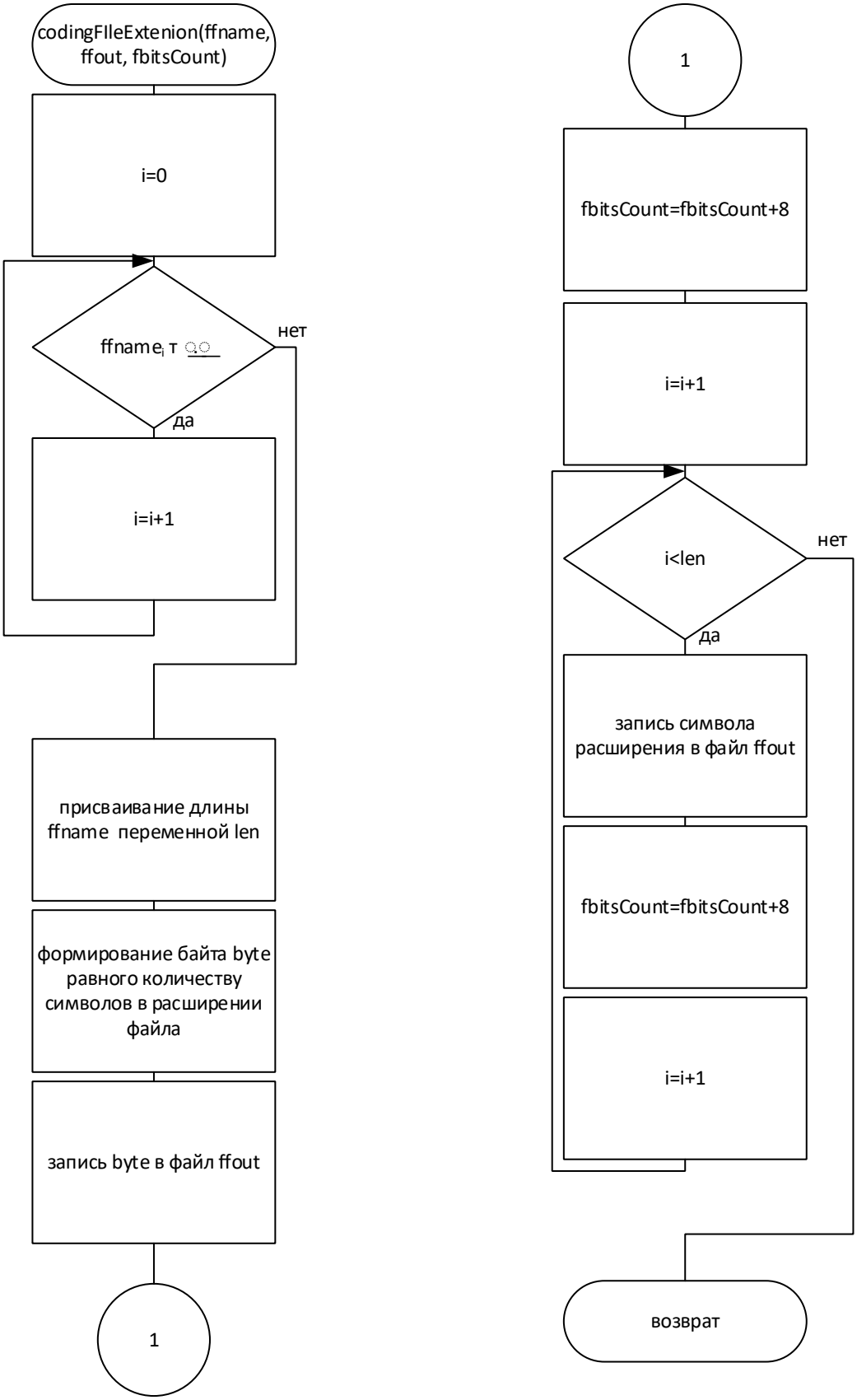

Рисунок 3 Функция кодирования расширения файла

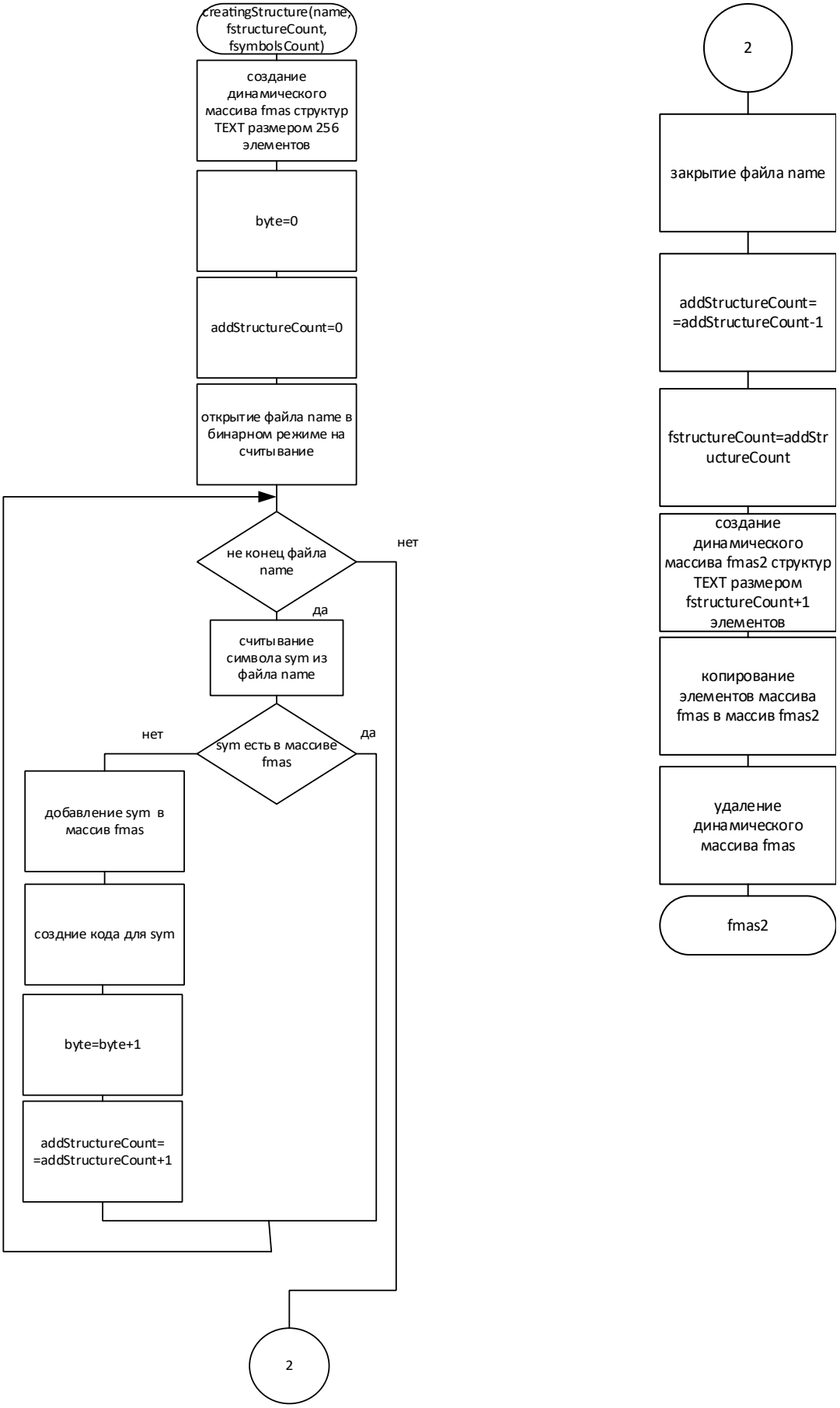

Рисунок 4 Функция создания и заполнения массива структур символов

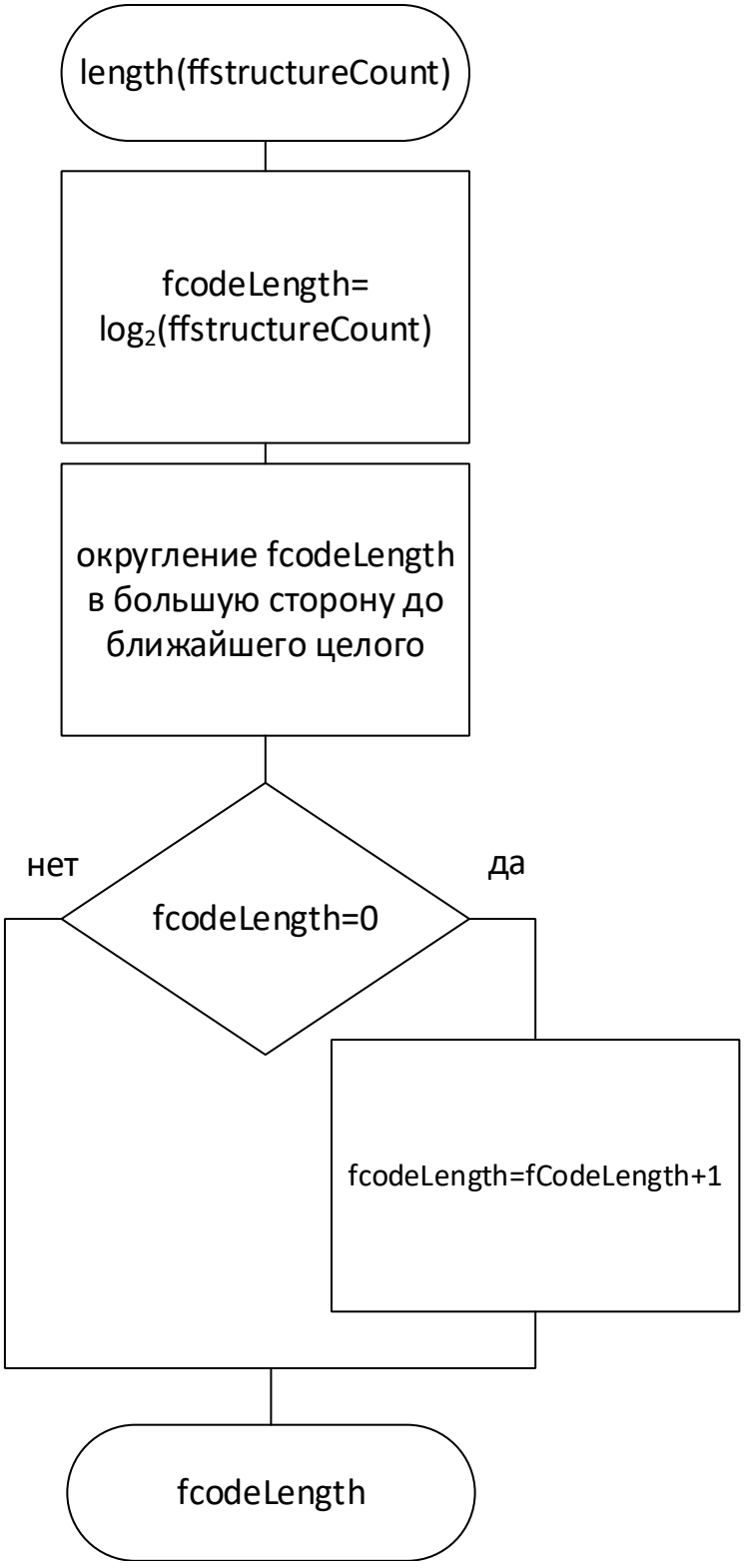

Рисунок 5 Функция вычисления длины кода

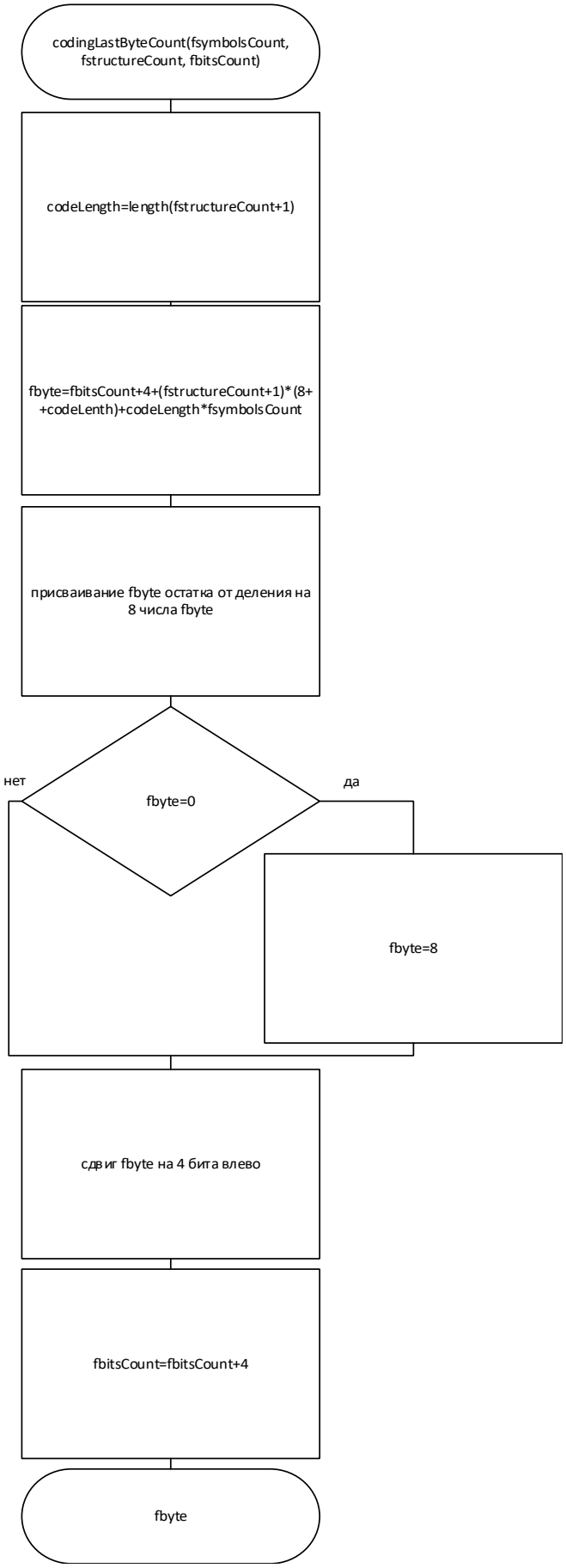

Рисунок 6 Функция кодирования количества бит, задействованных в последнем байте

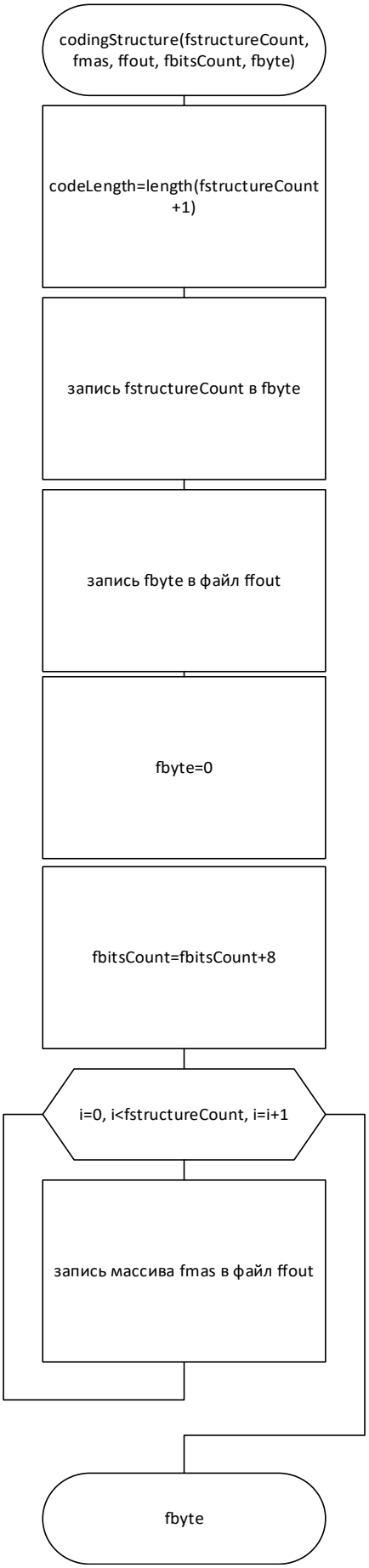

Рисунок 7 Функция кодирования массива структур символов

Рисунок 8 Функция кодирования входного файла

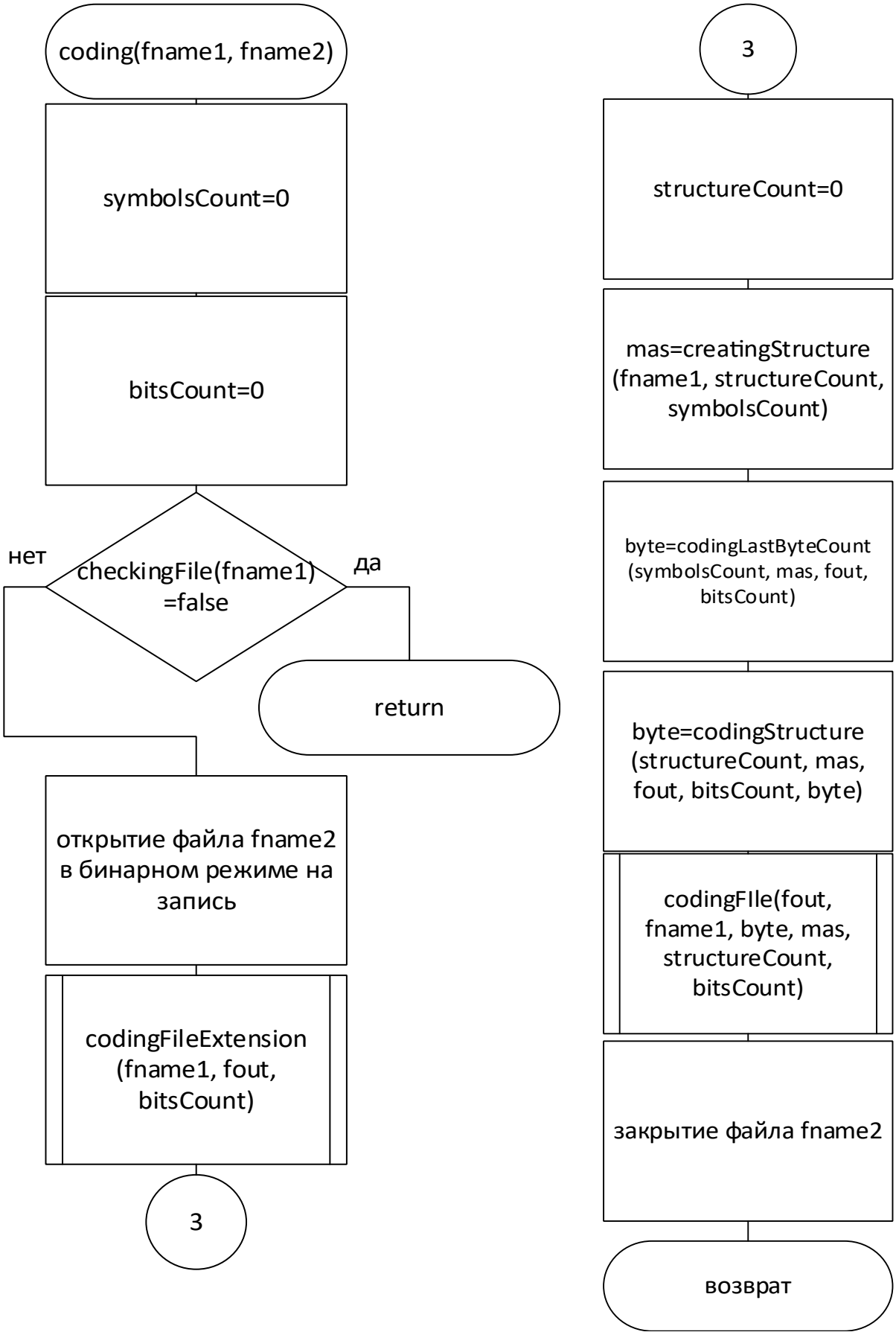

Рисунок 9 Функция кодирования

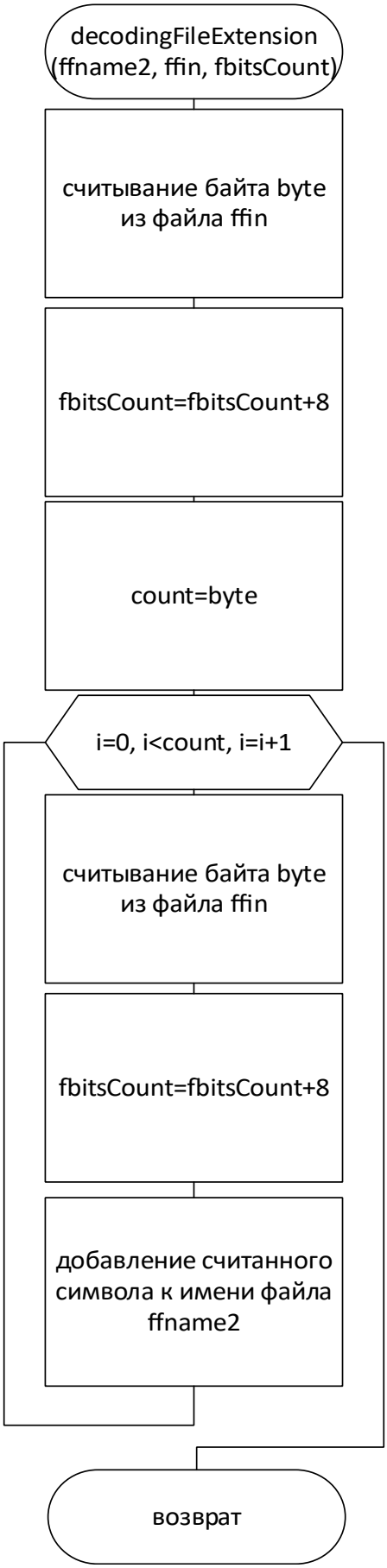

Рисунок 10 Функция расшифровки расширения файла

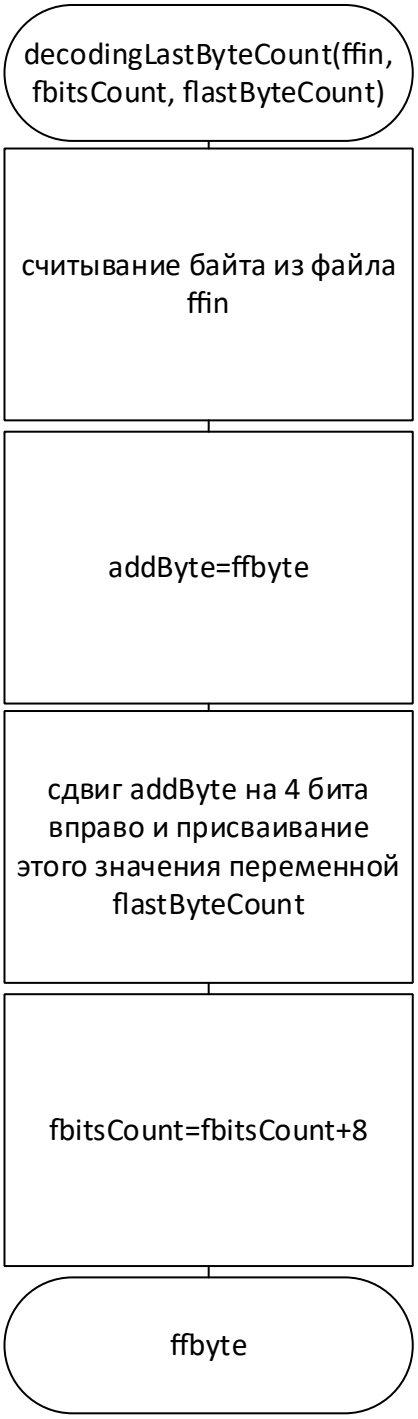

Рисунок 11 Функция расшифровки количества бит, задействованных в последнем байте

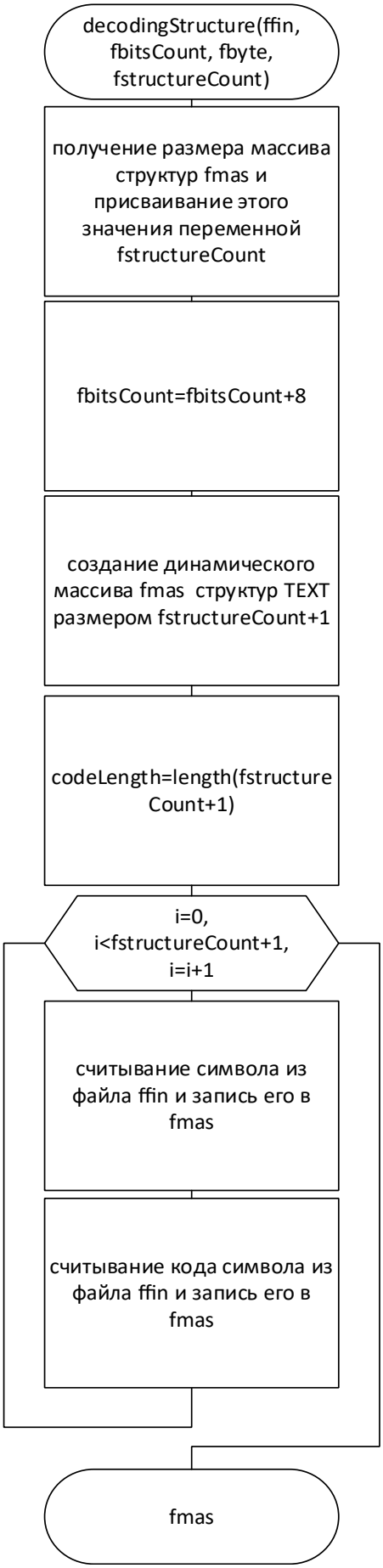

Рисунок 12 Функция создания и расшифровки массива структур символов

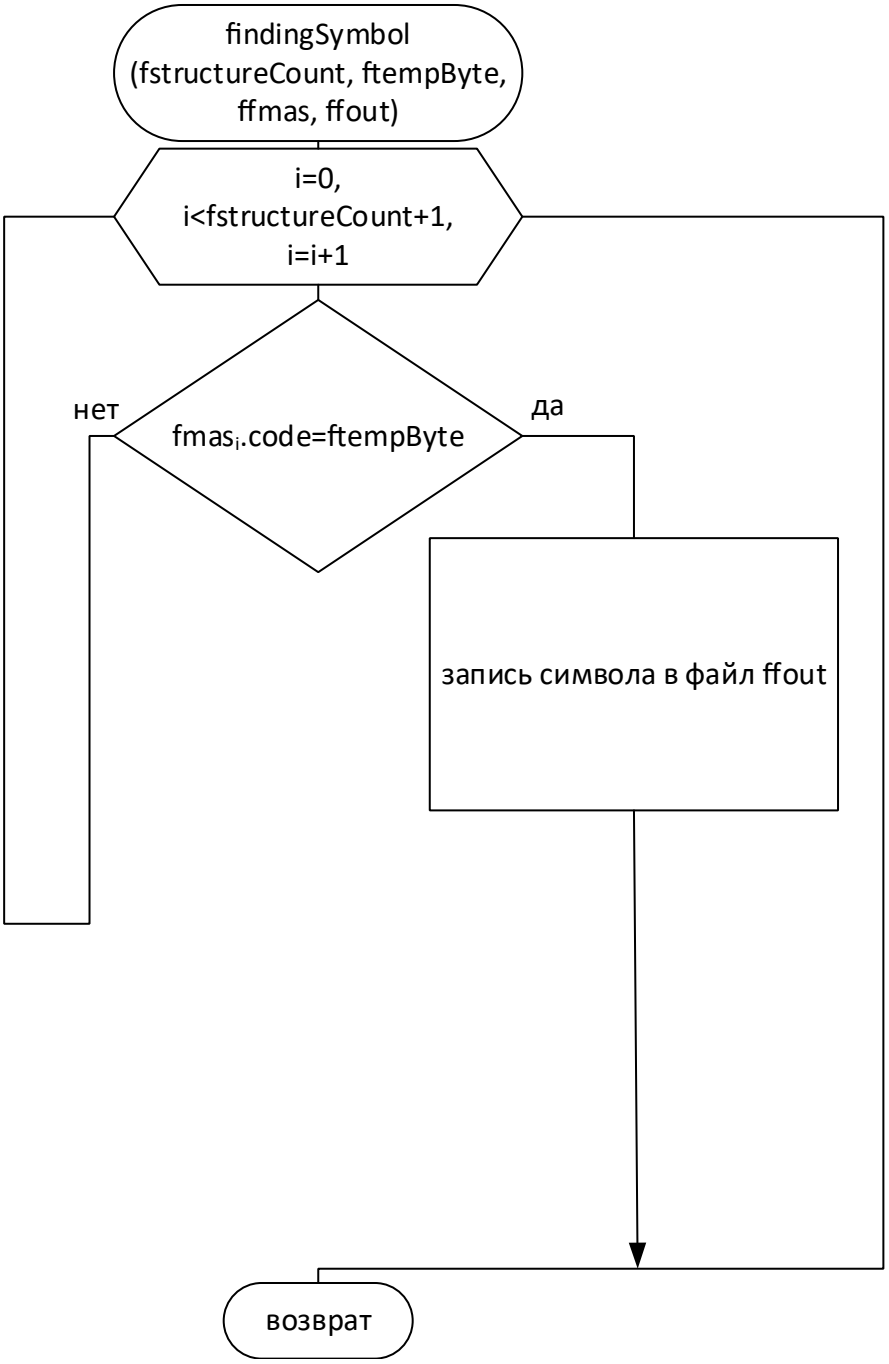

Рисунок 13 Функция поиска символа в массиве структур

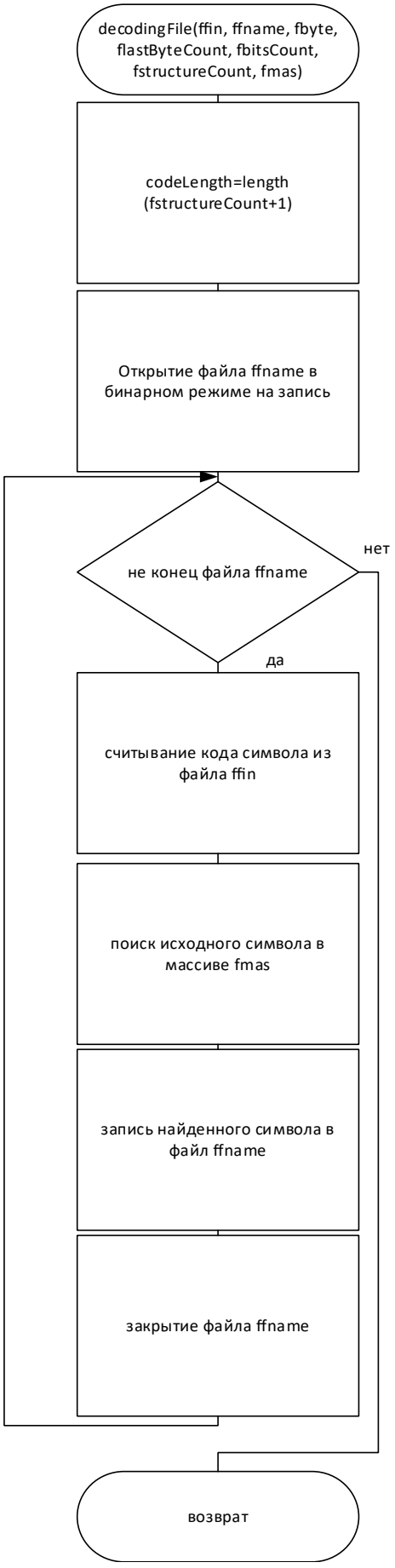

Рисунок 14 Функция расшифровки исходного файла

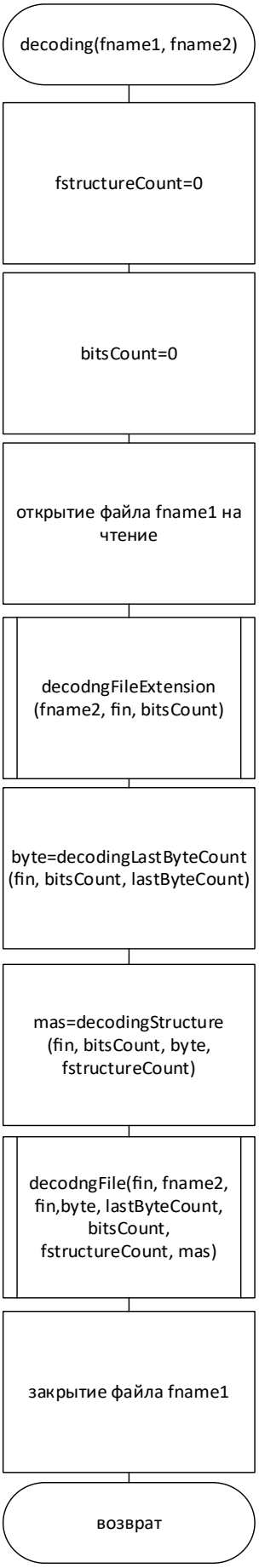

Рисунок 15 Функция декодирования

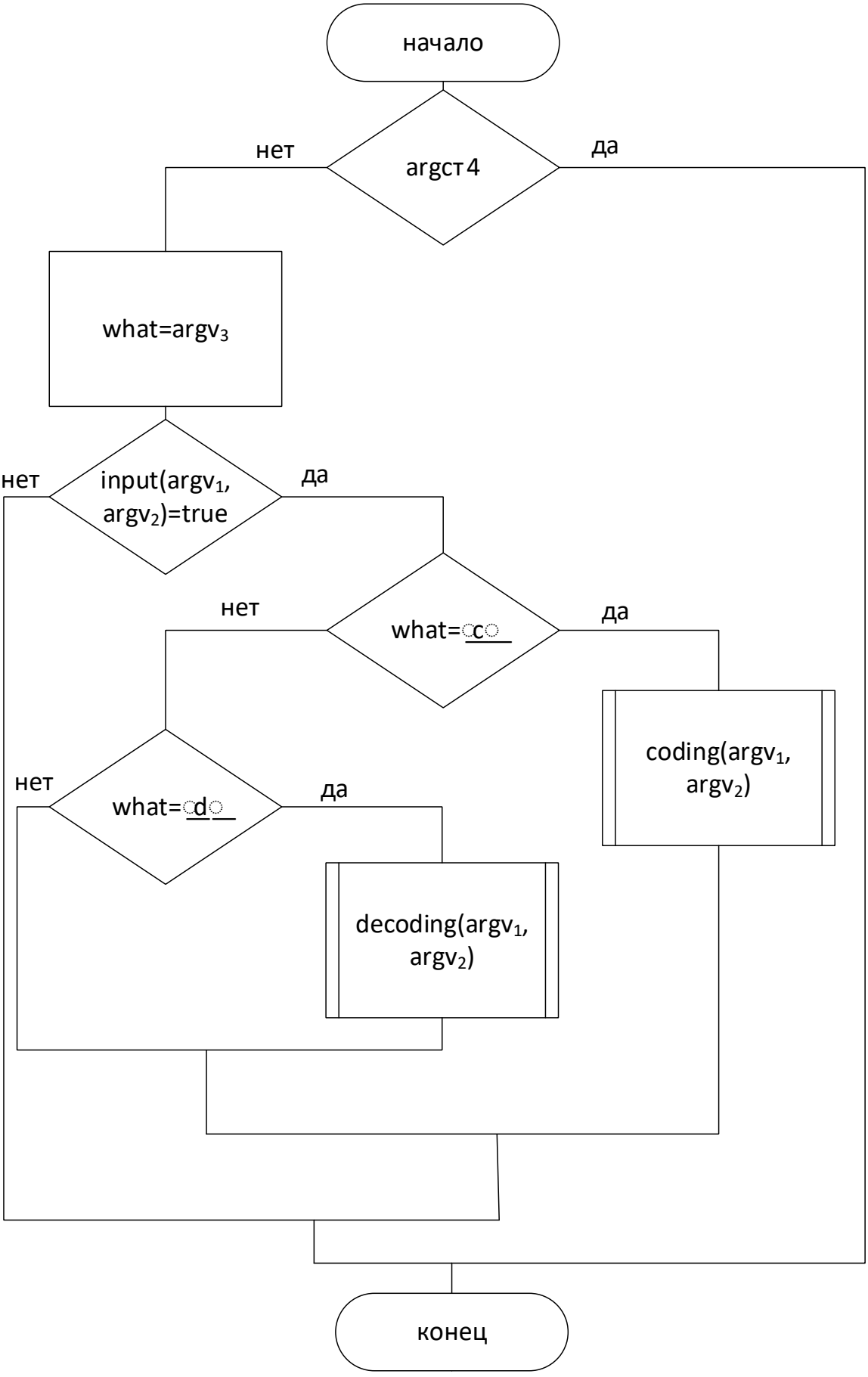

Рисунок 16 Основная программа

2.2. Описание структур и функций

- Структуры
    - TEXT — состоит из 2 элементов:
        1. unsigned char symbol — хранит непосредственно символ;
        2. unsigned code — хранит сформированный код символа;

2.4. Описание функций

1. input (const string fname1, const string fname2); (Рисунок 1, стр.6)

- Входные данные:
    - fname1 — имя входного файла;
    - fname2 — имя выходного файла;
- Выходные данные:
    - «true» если оба файла успешно открылись, иначе «false»

Происходит открытие входного файла. Если успешно — попытка создания выходного файла. Если и это успешно, то выходной файл удаляется, и функция возвращает «true». Если хоть один из файлов не открылся, функция возвращает «false» и выводит сообщение об ошибке.

2. checkingFile(const string fname); (Рисунок 2, стр.7)

- Входные данные:
    - fname — имя входного файла;
- Локальные переменные:
    - unsigned char sym — используется для считывания байта из файла;
- Выходные данные:
    - «false», если файл пуст, иначе — «true»;

Входной файл открывается на чтение, затем производится считывание байта из файла. Если достигнут конец файла, то функция закрывает файл и возвращает «false», т.к. файл пуст, иначе происходит закрытие файла и возврат «true».

3. codingFileExtension(const string ffname, ofstream &ffout, unsigned int &fbitsCount); (Рисунок 3, стр.8)

- Входные данные:

- ffname — имя входного файла;
- ffout — файловый поток вывода;
- fbitsCount — количество задействованных в кодировании бит;

- Локальные переменные:
  - int i — переменная счётчик;
  - unsigned char byte — используется для считывания байта из входного файла;
  - unsigned short int len — используется для вычисления количества символов в расширении входного файла;
- Выходные данные:
  - ffout — файловый поток вывода;
  - fbitsCount — количество задействованных в кодировании бит;

Высчитывается количество символов, находящихся в расширении входного файла, затем кодируется это значение и непосредственно сами символы расширения. Параллельно с этим происходит подсчёт количества бит, участвующих в кодировании. После успешного завершения операции функция возвращает поток вывода и количество задействованных в кодировании бит.

4. creatingStructure(const string name, unsigned char &fstructureCount, unsigned long long int &fsymbolsCount); (Рисунок 4, стр.9)

- Входные данные:
  - name — имя входного файла;
  - fstructureCount — количество различных байт во входном файле;
  - fsymbolsCount — общее количество символов во входном файле;
- Локальные переменные
  - unsigned char sym — используется для считывания байта из входного файла;
  - unsigned char sym2, bool checking— вспомогательные переменные;
  - unsigned char byte — используется для генерации кодов символов;
  - unsigned short int addStructureCount — используется для подсчёта количества различных байтов во входном файле;
  - TEXT fmas, fmas2 — массивы структур символов;

- Выходные данные
    - fmas2 — сформированный массив структур символов;
    - fstructureCount — количество различных символов во входном файле;
    - fsymbolsCount — количество всех символов во входном файле;

Создаётся массив структур символов на 256 элементов. Затем открывается входной файл на чтение и происходит считывание байтов. Если данный байт ещё не присутствует в массиве структур, то он туда добавляется и для него генерируется новый код. Если данный байт уже присутствует — происходит считывание следующего байта. Параллельно подсчитывается количество всех символов и количество различных символов. После того, как весь входной файл считан, происходит закрытие файла и создание нового массива структур размером, равным количеству различных символов. Далее происходит копирование старого массива в новый, удаление старого и возврат указателя на новый массив.

5. unsigned char length(const unsigned short int ffstructureCount); (Рисунок 5, стр.10)

- Входные данные:
    - const unsigned short int ffstructureCount — количество различных символов во входном файле;
- Локальные переменные:
    - unsigned char fcodeLength — используется для вычисления длины кода символов;
- Выходные данные:
    - fcodeLength — длина кода символов;

Происходит вычисление длины кода символов. Если она равна 0, то увеличиваем на 1. Затем происходит возврат этого значения.

6. codingLastByteCount(const unsigned long long int fsymbolsCount, const unsigned char fstructureCount, unsigned int &fbitsCount); (Рисунок 6, стр.11)

- Входные данные:
    - fsymbolsCount — количество всех символов во входном файле;
    - fstructureCount — количество различных символов во входном файле;
    - fbitsCount — количество задействованных в кодировании бит;

- Локальные переменные:
    - unsigned char fcodeLength — используется для вычисления длины кода символов;
    - unsigned char fbyte — байт с закодированным количеством символов в последнем байте;
- Выходные данные
    - fbyte — байт с закодированным количеством символов в последнем байте;

Происходит вычисление длины кода символов, затем вычисляется количество символов в последнем байте. Далее это значение кодируется и возвращается закодированный байт.

7. codingStructure(const unsigned char fstructureCount, TEXT *fmas, ofstream &ffout, unsigned int &fbitsCount, unsigned char fbyte); (Рисунок 7, стр.12)

- Входные данные:
    - fstructureCount — количество различных символов во входном файле;
    - fmas — массив структур символов;
    - ffout — файловый поток вывода;
    - fbitsCount — количество задействованных в кодировании бит;
    - unsigned char fbyte — байт с закодированными данными;
- Локальные переменные;
    - unsigned char fcodeLength — используется для вычисления длины кода символов;
- Выходные данные
    - fbyte — байт с закодированными данными;

Происходит кодирование массива структур символов: сначала кодируется сам символ, затем его код. По мере заполнения байта, он записывается в поток вывода. Параллельно происходит процесс подсчёта количества бит, задействованных в кодировании байтов. Процесс продолжается, пока не будут закодированы все символы. После этого функция возвращает текущий байт с закодированными данными, который не был записан в поток вывода.

8. void codingFile(ofstream &ffout, const string ffname, unsigned char &fbyte, TEXT *fmas, const unsigned char fstructureCount, unsigned int &fbitsCount, const unsigned long long int fsymbolsCount); (Рисунок 8,стр.13)

- Входные данные:
  - ffout — файловый поток вывода;
  - ffname — имя входного файла;
  - fbyte — байт с закодированными данными, который не был записан в поток вывода;
  - fmas — массив структур символов;
  - fstructureCount — количество различных символов во входном файле;
  - fbitsCount — количество задействованных бит в кодировании текущего байта;
  - fsymbolsCount — количество всех символов во входном файле;
- Локальные переменные:
  - unsigned char addByte — вспомогательная переменная;
  - unsigned char fcodeLength — используется для вычисления длины кода символов;

Происходит открытие входного файла на чтение, затем — процесс кодирования: из входного файла считывается байт, происходит его поиск в массиве структур символов и далее происходит формирование байта с кодом этого символа. По мере формирования байта происходит его запись в поток ввода. Операция продолжается, пока не будут обработаны все символы во входном файле. После этого входной файл закрывается.

9. coding(const string fname1, const string fname2); (Рисунок 9, стр.14)

- Входные данные:
  - fname1 — имя входного файла;
  - fname2 — имя выходного файла;
- Локальные переменные:
  - unsigned char byte — рабочий байт для кодирования;
  - unsigned long long int symbolsCount — количество всех символов во входном файле;

- unsigned int bitsCount — количество задействованных бит в кодировании текущего байта;
- ofstream fout — файловый поток вывода;
- unsigned char structureCount — количество различных символов во входном файле;

- Выходные данные:
  - Сообщение об успешно завершённой операции

Происходит открытие на запись выходного файла и поочерёдный вызов функций для кодирования входного файла. После этого выходной файл закрывается и выводится сообщение об успешно завершённой операции.

10. decodingFileExtension(string &ffname2, ifstream &ffin, unsigned int &fbitsCount); (Рисунок 10, стр.15)

- Входные данные:
  - ffname2 — имя выходного файла;
  - ffin — файловый поток ввода;
  - fbitsCount — количество задействованных бит в кодировании текущего байта;
- Локальные переменные:
  - char temp — вспомогательная переменная;
  - unsigned char byte — используется для считывания байтов из закодированного файла;
  - unsigned int count — используется для подсчёта количества символов в расширении закодированного файла;
- Выходные данные
  - ffname2 — имя выходного файла;
  - ffin — файловый поток ввода;
  - fbitsCount — количество задействованных бит в кодировании текущего байта;

Происходит вычисление количества символов, входящих в расширение файла. После этого считываются непосредственно сами символы расширения и добавляются к имени выходного файла. Параллельно с этим подсчитывается количество бит,

обработанных в текущем байте. По завершении работы функция возвращает новое имя выходного файла, а также количество бит, обработанных в текущем байте.

11. decodingLastByteCount(ifstream &ffin, unsigned int &fbitsCount, unsigned char &fLastByteCount); (Рисунок 11, стр.16)

- Входные данные:
  - ffin — файловый поток ввода;
  - fbitsCount — количество задействованных бит в кодировании текущего байта;
  - fLastByteCount — количество символов, задействованных в последнем байте;
- Локальные переменные
  - unsigned char ffbyte — рабочий байт для считывания данных;
  - unsigned char addByte — вспомогательная переменная;
- Выходные данные
  - unsigned char ffbyte — рабочий байт для считывания данных;

Из входного файла считывается байт, в котором хранится значение количества бит, используемых в последнем байте. Функция возвращает это значение, количество обработанных бит в текущем байте и непосредственно сам рабочий байт.

12. decodingStructure(ifstream &ffin, unsigned int &fbitsCount, unsigned char &fbyte, unsigned char &fstructureCount); (Рисунок 12, стр.17)

- Входные данные:
  - ffin — файловый поток ввода;
  - fbitsCount — количество задействованных бит в кодировании текущего байта;
  - ffbyte — рабочий байт для считывания данных;
  - fstructureCount — количество различных символов в закодированном файле;
- Локальные переменные:
  - unsigned char codeLength — используется для вычисления длины кода символов;
- Выходные данные:

- fmas — массив структур символов;
- ffin — файловый поток ввода;
- fbitsCount — количество задействованных бит в кодировании текущего байта;
- ffbyte — рабочий байт для считывания данных;
- fstructureCount — количество различных символов в закодированном файле;

Первым делом расшифровывается количество различных символов в закодированном файле, затем создаётся массив размером, равным данному количеству. После этого происходит расшифровка символа и его кода. После расшифровки всех символов функция возвращает указатель на сформированный массив, а также поток ввода, количество задействованных бит в кодировании текущего байта, текущий байт и размер массива структур.

13. findingSymbol(const unsigned char ffstructureCount, const unsigned char ftempByte, TEXT *ffmas, ofstream &ffout); (Рисунок 13, стр.18)

- Входные данные:
  - fstructureCount — количество различных символов в закодированном файле;
  - ftempByte — код символа, который необходимо раскодировать;
  - ffmas — массив структур символов;
  - ffout — файловый поток вывода;
- Выходные данные:
  - Раскодированный символ;

Функция осуществляет поиск закодированного символа в массиве структур. После его обнаружения происходит запись раскодированного символа в поток вывода.

14. decodingFile(ifstream &ffin, const string ffname, unsigned char &fbyte, unsigned char flastByteCount, unsigned int &fbitsCount, const unsigned char fstructureCount, TEXT *fmas); (Рисунок 14, стр.19)

- Входные данные:
  - ffname — имя выходного файла;
  - ffbyte — рабочий байт для считывания данных;

- ffin — файловый поток ввода;
- fLastByteCount — количество символов, задействованных в последнем байте;
- fbitsCount — количество задействованных бит в кодировании текущего байта;
- fstructureCount — количество различных символов в закодированном файле;
- ffmas — массив структур символов;

- Локальные переменные:
  - unsigned char codeLength — используется для вычисления длины кода символов;
  - unsigned char lastByte, tempByte — вспомогательные переменные;
- Выходные данные:
  - Раскодированный файл;

Первым делом происходит вычисление длины кода закодированных символов. Далее открывается выходной файл на запись и происходит следующее: из входного файла вычленяется код символа, затем происходит поиск самого символа, который после обнаружения записывается в выходной файл. Процесс продолжается пока не будут раскодированы все символы. После выполнения данной операции выходной файл закрывается.

15. decoding(const string fname1, const string fname2); (Рисунок 15, стр.20)

- Входные данные:
  - fname1 — имя входного файла;
  - fname2 — имя выходного файла;
- Локальные переменные:
  - unsigned char byte — рабочий байт для декодирования;
  - unsigned char structureCount — количество различных символов во входном файле;
  - unsigned int bitsCount — количество задействованных бит в декодировании текущего байта;

- unsigned char lastByteCount — количество задействованных бит в последней байте;
- unsigned int bitsCount — количество задействованных бит в кодировании текущего байта;
- ifstream fin — файловый поток ввода;

- Выходные данные:
  - Сообщение об успешно завершённой операции;

Происходит открытие на чтение входного файла и поочерёдный вызов функций для декодирования входного файла. После этого входной файл закрывается и выводится сообщение об успешной завершении операции.

1. Основная программа; (Рисунок 16, стр.21)

- Локальные переменные:
  - what – хранит необходимое действие: кодирование или декодирование;

Программа запускается с командной строки с 4-мя параметрами: имя программы, имя входного файла, имя выходного файла, необходимое действие. Если количество аргументов не равно 4, неправильно введены имена файлов или необходимое действие, то выводится сообщение об ошибке. Иначе происходит вызов соответствующих функций для кодирования или декодирования файла

Заключение

В ходе выполнения данной курсовой работы была написана программа для сжатия/распаковки файлов методом равномерного кодирования. Разработанное средство получилось нетребовательным к системным ресурсам, обработка файлов происходит максимально быстро. Также в программе производится контроль вводимой пользователем информации, исключены тупиковые ситуации. Можно отметить, что в ходе выполнения курсовой работы мною более глубоко был изучен язык программирования «C++».

Список литературы

1. Информатика: методические указания к выполнению лабораторных работ для студентов очной формы обучения направления бакалавриата 230400 – Информационные системы и технологии / сост: С.Н. Рога, А.Г. Смышляев, Ю.И. Солопов. – Белгород: Изд-во БГТУ, 2013.
2. Макконнел С. Совершенный код. Мастер-класс / Пер.с англ. — М.:Издательство «Русская редакция», 2010. — 896 стр.: ил.
3. Шилдт, Герберт. Полный справочник по C++, 4-е издание. Пер. С англ. — М.: Издательский дом «Вильямс», 2006. — 800 с.: ил. — Парал.тит.англ.

Приложение 1 Листинг программы

```
#include<iostream>
#include<fstream>
#include<locale>
#include<windows.h>
#define errorMessage "Ошибка"
#define complete "Операция завершена"
using namespace std;


struct TEXT
{
	unsigned char symbol, code;
};


bool input(const string fname1,const string fname2)
{
	ifstream fin(fname1);
	if (!fin.is_open())
		return false;
	else
		fin.close();
	ofstream fout(fname2);
	if (!fout.is_open())
		return false;
	else
	{
		fout.close();
		remove(fname2.c_str());
	}
	return true;
}


void codingFileExtension(const string ffname, ofstream &ffout, unsigned int &fbitsCount)
{
	int i = 0;
	unsigned char byte;
	unsigned short int len;
	while (ffname[i] != '.')
		i++;
	len = ffname.length();
	byte = len - i - 1;
	ffout << byte;
	fbitsCount += 8;
	i++;
	while (i < len)
	{
```

```
                ffout << ffname[i];
                fbitsCount += 8;
                i++;
        }
}


bool checkingFile(const string fname)
{
        ifstream fin(fname, ios::binary);
        unsigned char sym = fin.get();
        if (fin.eof())
        {
                cout << "Файл пуст" << endl;
                fin.close();
                return false;
        }
        else
        {
                fin.close();
                return true;
        }
}


TEXT *creatingStructure(const string name, unsigned char &fstructureCount, unsigned long long int &fsymbolsCount)
{
        TEXT *fmas = new TEXT[256];
        ifstream fin(name, ios::binary);
        unsigned char sym, byte = 0;
        unsigned short int addStructureCount = 0;
        unsigned char sym2 = fin.get();
        bool checking;
        while (!fin.eof())
        {
                sym = sym2;
                fsymbolsCount++;
                checking = false;
                for (int i = 0; i < addStructureCount; i++)
                        if (sym == fmas[i].symbol)
                        {
                                checking = true;
                                break;
                        }
                if (!checking)
                {
                        fmas[addStructureCount].symbol = sym;
                        fmas[addStructureCount].code = byte;
                        byte++;
                        addStructureCount++;
```

```
                }
                sym2 = fin.get();
        }
        fin.close();
        addStructureCount--;
        fstructureCount = addStructureCount;
        TEXT *fmas2 = new TEXT[fstructureCount + 1];
        for (int i = 0; i < fstructureCount + 1; i++)
        {
                fmas2[i].symbol = fmas[i].symbol;
                fmas2[i].code = fmas[i].code;
        }
        delete[]fmas;
        return fmas2;
}


unsigned char length(const unsigned short int ffstructureCount)
{
        unsigned char fcodeLength = ceil(log(ffstructureCount) / log(2.0));
        if (fcodeLength == 0)
                fcodeLength++;
        return fcodeLength;
}


unsigned char codingLastByteCount(const unsigned long long int fsymbolsCount, const unsigned char
fstructureCount, unsigned int &fbitsCount)
{
        unsigned char codeLength = length(fstructureCount + 1);
        unsigned char fbyte = fbitsCount + 4 + (fstructureCount + 1) * (8 + codeLength) + codeLength *
fsymbolsCount;
        fbyte %= 8;
        if (fbyte == 0)
                fbyte = 8;
        fbyte = fbyte << 4;
        fbitsCount += 4;
        return fbyte;
}


unsigned char codingStructure(const unsigned char fstructureCount, TEXT *fmas, ofstream &ffout, unsigned int
&fbitsCount, unsigned char fbyte)
{
        unsigned char codeLength = length(fstructureCount + 1);
        fbyte ^= ((fstructureCount) >> 4);
        ffout << fbyte;
        fbyte = 0;
        fbyte ^= ((fstructureCount) << 4);
        fbitsCount += 8;
        for (int i = 0; i < fstructureCount + 1; i++)
```

```
	{
		fbitsCount %= 8;
		if (fbitsCount % 8 == 0)
		{
			fbyte ^= fmas[i].symbol;
			ffout << fbyte;
			fbyte = 0;
			fbitsCount += 8;
		}
		else
		{
			fbyte ^= (fmas[i].symbol >> fbitsCount % 8);
			ffout << fbyte;
			fbyte = 0;
			fbyte ^= (fmas[i].symbol << (8 - fbitsCount % 8));
			fbitsCount += 8;
		}
		if ((fbitsCount % 8 + codeLength) < 8)
		{
			fbyte ^= (fmas[i].code << (8 - fbitsCount % 8 - codeLength));
			fbitsCount += codeLength;
		}
		else
			if ((fbitsCount % 8 + codeLength) == 8)
			{
				fbyte ^= fmas[i].code;
				fbitsCount += codeLength;
				ffout << fbyte;
				fbyte = 0;
			}
			else
			{
				fbyte ^= (fmas[i].code >> (fbitsCount % 8 + codeLength - 8));
				ffout << fbyte;
				fbyte = 0;
				fbitsCount += codeLength;
				fbyte ^= (fmas[i].code << (8 - fbitsCount % 8));
			}
	}
	return fbyte;
}


void codingFile(ofstream &ffout, const string ffname, unsigned char &fbyte, TEXT *fmas, const unsigned char fstructureCount, unsigned int &fbitsCount, const unsigned long long int fsymbolsCount)
{
	ifstream fin(ffname, ios::binary);
	unsigned char addByte;
	unsigned char codeLength = length(fstructureCount + 1);
	for (int i = 0; i < fsymbolsCount; i++)
	{
```

```
                fbitsCount %= 8;
                addByte = fin.get();
                for (int j = 0; j < fstructureCount + 1; j++)
                        if (addByte == fmas[j].symbol)
                        {
                                addByte = fmas[j].code;
                                break;
                        }
                if ((fbitsCount % 8 + codeLength) == 8)
                {
                        fbyte ^= addByte;
                        ffout << fbyte;
                        fbitsCount += codeLength;
                        fbyte = 0;
                }
                else
                        if ((fbitsCount % 8 + codeLength) < 8)
                        {
                                fbyte ^= (addByte << (8 - fbitsCount % 8 - codeLength));
                                fbitsCount += codeLength;
                        }
                        else
                        {
                                fbyte ^= (addByte >> (codeLength + fbitsCount % 8 - 8));
                                ffout << fbyte;
                                fbyte = 0;
                                fbyte ^= (addByte << (16 - fbitsCount % 8 - codeLength));
                                fbitsCount += codeLength;
                        }
        }
        if (fbitsCount % 8 != 0)
                ffout << fbyte;
        fin.close();
}


void coding(const string fname1, const string fname2)
{
        unsigned char byte;
        unsigned long long int symbolsCount = 0;
        unsigned int bitsCount = 0;
        if (!checkingFile(fname1))
                return;
        ofstream fout(fname2, ios::binary);
        codingFileExtension(fname1, fout, bitsCount);
        unsigned char structureCount = 0;
        TEXT *mas = creatingStructure(fname1, structureCount, symbolsCount);
        byte = codingLastByteCount(symbolsCount, structureCount, bitsCount);
        byte = codingStructure(structureCount, mas, fout, bitsCount, byte);
        codingFile(fout, fname1, byte, mas, structureCount, bitsCount, symbolsCount);
        fout.close();
```

```
}


void decodingFileExtension(string &ffname2, ifstream &ffin, unsigned int &fbitsCount)
{
	unsigned int count;
	char temp;
	unsigned char byte;
	ffname2 = ffname2 + '.';
	byte = ffin.get();
	fbitsCount += 8;
	count = byte;
	for (int i = 0; i < count; i++)
	{
		byte = ffin.get();
		fbitsCount += 8;
		temp = byte;
		ffname2 = ffname2 + temp;
	}
}

unsigned char decodingLastByteCount(ifstream &ffin, unsigned int &fbitsCount, unsigned char &fLastByteCount)
{
	unsigned char ffbyte = ffin.get(), addByte = ffbyte;
	fLastByteCount = (addByte >> 4);
	fbitsCount += 4;
	return ffbyte;
}


TEXT *decodingStructure(ifstream &ffin, unsigned int &fbitsCount, unsigned char &fbyte, unsigned char &fstructureCount)
{
	fstructureCount = (fbyte << 4);
	fbyte = ffin.get();
	fstructureCount ^= (fbyte >> 4);
	fbitsCount += 8;
	TEXT *fmas = new TEXT[fstructureCount + 1];
	unsigned short int codeLength = length(fstructureCount + 1);
	for (int i = 0; i < fstructureCount + 1; i++)
	{
		fbitsCount %= 8;
		if (fbitsCount % 8 == 0)
		{
			fmas[i].symbol = fbyte;
			fbyte = ffin.get();
			fbitsCount += 8;
		}
		else
		{
			fmas[i].symbol = (fbyte << fbitsCount % 8);
```

```
                        fbyte = ffin.get();
                        fmas[i].symbol ^= (fbyte >> (8 - fbitsCount % 8));
                        fbitsCount += 8;
                }
                if ((fbitsCount % 8 + codeLength) == 8)
                {
                        fbyte <<= fbitsCount % 8;
                        fbyte >>= fbitsCount % 8;
                        fmas[i].code = fbyte;
                        fbyte = ffin.get();
                        fbitsCount += codeLength;
                }
                else
                        if ((fbitsCount % 8 + codeLength) < 8)
                        {
                                fmas[i].code = (fbyte >> (8 - codeLength - fbitsCount % 8));
                                fmas[i].code <<= (8 - codeLength);
                                fmas[i].code >>= (8 - codeLength);
                                fbitsCount += codeLength;
                        }
                        else
                        {
                                fmas[i].code = fbyte;
                                fmas[i].code <<= fbitsCount % 8;
                                fmas[i].code >>= (8 - codeLength);
                                fbyte = ffin.get();
                                fbitsCount += codeLength;
                                fmas[i].code ^= (fbyte >> (8 - fbitsCount % 8));
                        }
                if (fbitsCount % 8 == 0)
                        fbitsCount = 0;
        }
        return fmas;
}

void findingSymbol(const unsigned char ffstructureCount, const unsigned char ftempByte, TEXT *ffmas, ofstream
&ffout)
{
        for (int i = 0; i < ffstructureCount + 1; i++)
                if (ffmas[i].code == ftempByte)
                {
                        ffout << ffmas[i].symbol;
                        break;
                }
}

void decodingFile(ifstream &ffin, const string ffname, unsigned char &fbyte, unsigned char flastByteCount,
unsigned int &fbitsCount, const unsigned char fstructureCount, TEXT *fmas)
{
        unsigned char codeLength = length(fstructureCount + 1);
        unsigned char lastByte, tempByte;
```

```
	ofstream fout(ffname, ios::binary);
	lastByte = ffin.get();
	while (!ffin.eof())
	{
		fbitsCount %= 8;
		if ((fbitsCount % 8 + codeLength) == 8)
		{
			fbyte <<= fbitsCount % 8;
			fbyte >>= fbitsCount % 8;
			findingSymbol(fstructureCount, fbyte, fmas, fout);
			fbitsCount += codeLength;
			fbyte = lastByte;
			lastByte = ffin.get();
		}
		else
			if ((fbitsCount % 8 + codeLength) < 8)
			{
				tempByte = fbyte << fbitsCount % 8;
				tempByte >>= (8 - codeLength);
				findingSymbol(fstructureCount, tempByte, fmas,  fout);
				fbitsCount += codeLength;
			}
			else
			{
				tempByte = fbyte << fbitsCount % 8;
				fbitsCount += codeLength;
				tempByte >>= (8 - codeLength);
				fbyte = lastByte;
				lastByte = ffin.get();
				tempByte ^= fbyte >> (8 - fbitsCount % 8);
				findingSymbol(fstructureCount, tempByte, fmas, fout);
			}
	}
	fbitsCount = 8 + fbitsCount % 8;
	flastByteCount += 8;
	while (fbitsCount < flastByteCount)
	{
		tempByte = fbyte << fbitsCount % 8;
		fbitsCount += codeLength;
		tempByte >>= (8 - codeLength);
		findingSymbol(fstructureCount, tempByte, fmas, fout);
		if (fbitsCount % 8 == 0)
			break;
	}
	fout.close();
}

void decoding(const string fname1, string fname2)
{
	unsigned char fstructureCount = 0;
	unsigned char byte, lastByteCount;
```

```
	unsigned int bitsCount = 0;
	ifstream fin(fname1, ios::binary);
	decodingFileExtension(fname2, fin, bitsCount);
	byte = decodingLastByteCount(fin, bitsCount, lastByteCount);
	TEXT *mas = decodingStructure(fin, bitsCount, byte, fstructureCount);
	decodingFile(fin, fname2, byte, lastByteCount, bitsCount, fstructureCount, mas);
	fin.close();
}

void main(int argc, char* argv[])
{
	string what;
	setlocale(LC_ALL, "russian");
	if (argc != 4)
		cout << errorMessage<<endl;
	else
	{
		what = argv[3];
		if ((input(argv[1], argv[2])) == true)
		{
			if (what == "c")
			{
				coding(argv[1], argv[2]);
				cout << complete << endl;
			}
			else
				if (what == "d")
				{
					decoding(argv[1], argv[2]);
					cout << complete << endl;
				}
				else
					cout << errorMessage << endl;
		}
		else
			cout << errorMessage << endl;
	}
}
```

Приложение 2 Руководство пользователя

Программа запускается с командной строки с 4-мя параметрами (Рисунок 17):

1. Непосредственно само имя программы
2. имя входного файла
3. имя выходного файла
4. режим работы: «c» — кодирование, «d» - декодирование

При ошибке в каком-либо аргументе выведется сообщение об ошибке, иначе программа выведет сообщение об успешном окончании процесса.

Рисунок 17 Руководство пользователя — неправильный запуск программы

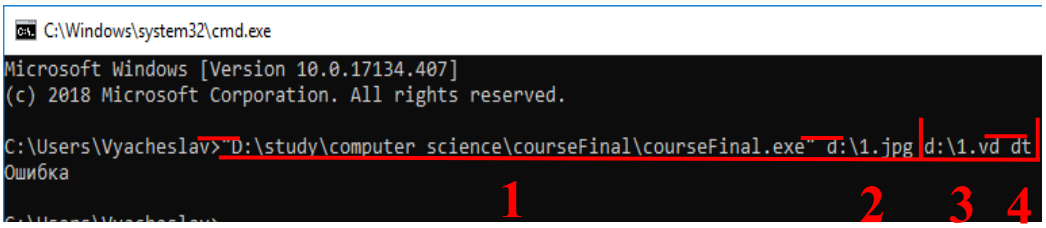

Рисунок 18 Правильный запуск программы (кодирование)

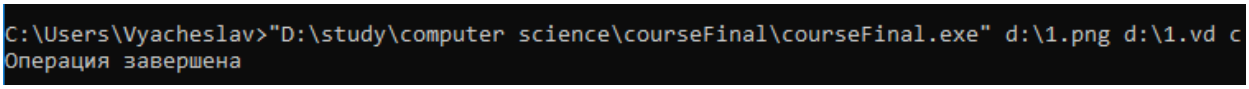

Рисунок 19 Правильный запуск программы (декодирование)