МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА №43

КУРСОВАЯ РАБОТА (ПРОЕКТ)   
ЗАЩИЩЕНА С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| старший преподаватель |  |  |  | А.А.Фоменкова |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ (ПРОЕКТУ) |
| ЭФФЕКТИВНОЕ КОДИРОВАНИЕ СООБЩЕНИЯ |
| по дисциплине: ОСНОВЫ ПРОГРАММИРОВАНИЯ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | М112 |  |  |  | A.А.Поспелова |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2022

**ОГЛАВЛЕНИЕ**

[**ЭФФЕКТИВНОЕ КОДИРОВАНИЕ СООБЩЕНИЯ 1**](#_Toc120686764)

[**1. ПОСТАНОВКА ЗАДАЧИ 3**](#_Toc120686765)

[1.1. Первичная постановка задачи 3](#_Toc120686766)

[1.2. Цель и задачи приложения 3](#_Toc120686767)

[1.3. Сценарии использования 3](#_Toc120686768)

[**2.** **ОПИСАНИЕ ДАННЫХ И СТРУКТУР ДАННЫХ** **5**](#_Toc120686769)

[2.1. Описание данных, запрашиваемых у пользователя 5](#_Toc120686770)

[2.2. Структура кодовой таблицы: 5](#_Toc120686771)

[**3.** **ОПИСАНИЕ ПРОГРАММЫ И ФУНКЦИЙ** **7**](#_Toc120686772)

[**4.** **РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ** **17**](#_Toc120686773)

[**ЗАКЛЮЧЕНИЕ 25**](#_Toc120686774)

[**СПИСОК ЛИТЕРАТУРЫ 26**](#_Toc120686775)

[**ПРИЛОЖЕНИЕ А** **27**](#_Toc120686776)

# 1. ПОСТАНОВКА ЗАДАЧИ

# 

## Первичная постановка задачи

Вариант 19

Задание: разработать приложение на языке программирования С/С++, выполняющее эффективное кодирование сообщения.

# 

## Цель и задачи приложения

На вход программы подается текстовый файл с исходным сообщением, на выходе имеется текстовый файл с закодированным сообщением и соответствующая кодовая таблица.

## Сценарии использования

Предусловие: создаётся файл формата *.txt* с исходным текстовым сообщением.

Сценарий: запуск приложения.

1. Пользователь запускает приложение.
2. На экране отображается текст с предложением ввести путь к исходному файлу на компьютере.
3. Пользователь вводит название файла.
4. Приложение выводит сообщение о том, что выполняет анализ текста.
5. Приложение выводит исходный текст, размер алфавита, количество вхождений каждой буквы и кодовую таблицу.
6. На экране отображается текст с предложением ввести путь к файлу для записи закодированного текста.
7. Пользователь вводит название файла.
8. Приложение выводит сообщение о том, что кодовая таблица была

записана в файл.

1. На экране отображается текст с предложением ввести путь к файлу для записи кодовой таблицы.
2. Пользователь вводит название файла.
3. Приложение выводит сообщение о том, что кодовая таблица была

записана в файл.

1. **ОПИСАНИЕ ДАННЫХ И СТРУКТУР ДАННЫХ**
   1. **Описание данных, запрашиваемых у пользователя**

Разрабатываемая программа предполагает текстовый интерфейс взаимодействия с пользователем. Введенные пользователем данные проходят проверки в соответствии с таблицей 1.

Таблица 1 – Описание данных, вводимых пользователем

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование переменной | Тип данных | Семантика | Описание проводимых проверок корректности ввода |
| *filename* | *string* | Путь и имя файла, в котором имеется исходный текст | - |
| *filename\_code* | *string* | Путь и имя файла, в который будет записан результат | - |
| *filename\_table* | *string* | Путь и имя файла, в который будет записана таблица кодировки | - |

* На вход поступает файл, содержащий исходный текст;
* На выходе программы: - файл с зашифрованным сообщением;

- файл с таблицей кодировки.

* 1. **Структура кодовой таблицы:**

Левый столбец – символы (алфавит), правый столбец – код буквы. Между столбцами разделительный символ “|”. В конец строки добавляется символ *“\n”*

Пример:

*A|00\n*

*B|01\n*

*C|100\n*

*D|101\n*

*E|110\n*

*F|111\n*

1. **ОПИСАНИЕ ПРОГРАММЫ И ФУНКЦИЙ**

Точкой входа в программу является функция *main()*, организующая вывод меню пользователя и вызов остальных функций. Блок-схема функции main() представлена на рисунке 1, листинги функции *main()* и всех реализованных функций приведены в разделе Приложение А.

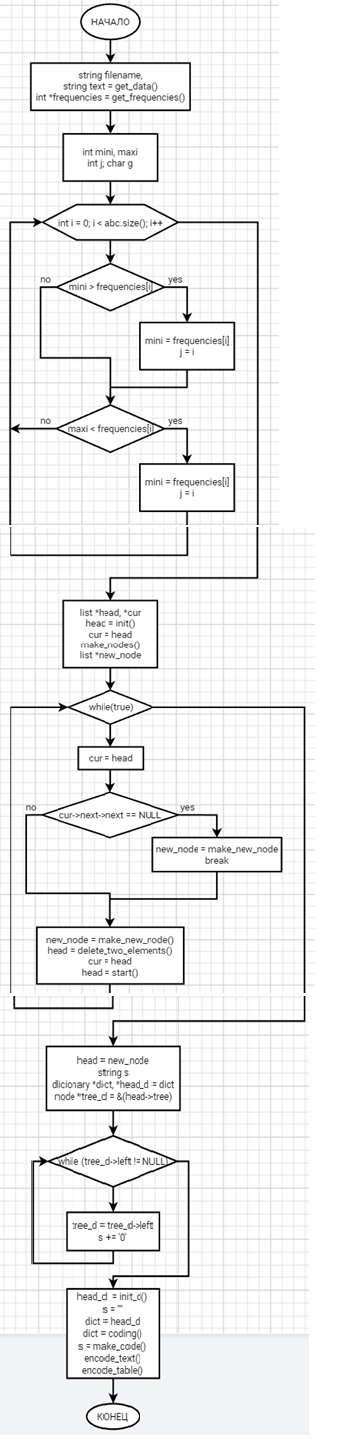


Рисунок 1 – Блок-схема алгоритма работы функции *main()*

В программе выполняется заполнение линейного списка, элементы которого в последствии становятся узлами дерева. Для реализации этого метода в курсовом проекте разработана функция *make\_nodes(),* блок-схема алгоритма которой представлена на рис. 2.

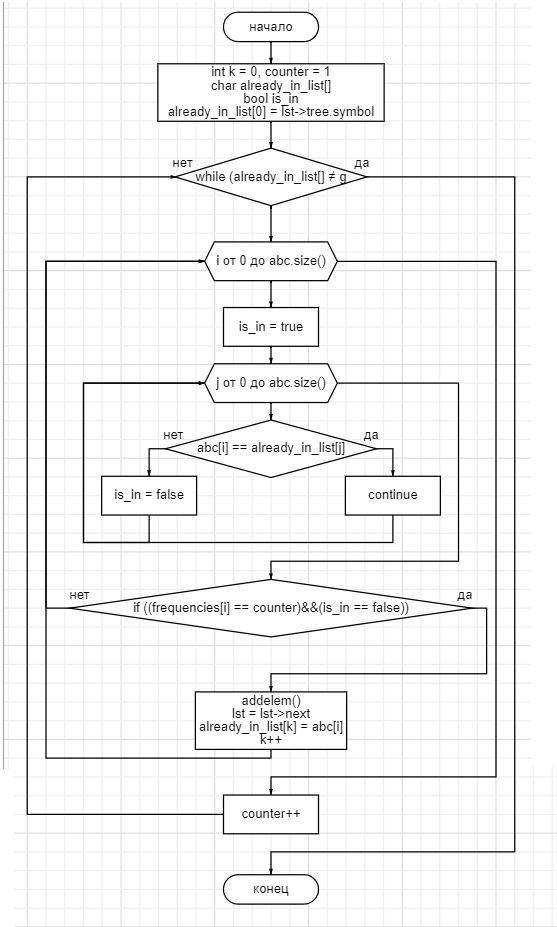


Рисунок 2 – Блок-схема алгоритма работы функции *make\_nodes ()*

Для реализации заполнения словаря букв и их кодовых слов разработана функция *coding()*, которая считывает «имя» буквы из двоичного дерева и сохраняет в словарь. Блок-схема функции *coding()* представлена на рис. 3.

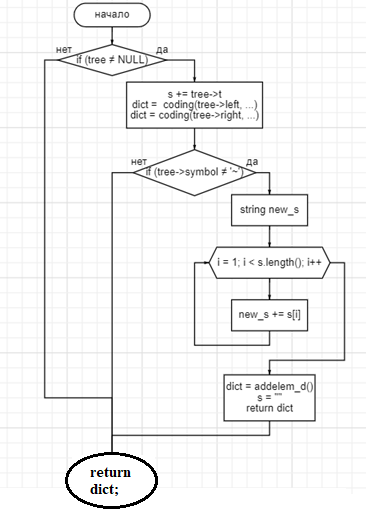


Рисунок 3 – Блок-схема алгоритма работы функции *coding ()*

В таблице 2 приведен список всех разработанных для реализации

программы функций и процедур.

Таблица 2 – функции программы

|  |  |  |  |
| --- | --- | --- | --- |
| **Прототип** | **Входные данные** | **Выходные параметры** | **Описание функционала** |
| *int check\_numbers()* | - | Корректный ввод числа | Функция запрашивает у пользователя на ввод с клавиатуры число и проверяет, является ли введённое данное натуральным целым числом. При некорректном вводе пользователю выводится сообщение об ошибке и предлагается повторить ввод. |
| *string get\_data (string filename)* | Имя файла | Текст для обработки | Функция по указанному имени файла (*filename*) считывает его содержимое и записывает в строку *text*, затем возвращает её значение. |
| *string alphabet (string text)* | Искомый текст | Алфавит текса | Функция принимает на вход тест и получает его алфавит (используемые в тексте буквы), записывая их в строку *abc*, которую затем возвращает из функции. |
| *void get\_frequency (string text, int \*frequencies, string abc)* | Искомый текст,  Указатель на массив, содержащий число вхождений каждого символа,  Алфавит | - | Функция, анализируя текст, записывает в массив *frequencies* соответствующее каждой букве алфавита число её вхождений в текст. |
| *void print\_abc(string text, string abc, int \*frequencies)* | Искомый текст,  Алфавит, Указатель на массив, содержащий число вхождений каждого символа | - | Функция вывода алфавита, размера алфавита и число вхождений каждого символа в текст. |
| *void make\_nodes(int \*frequencies, string abc, list \*lst, char g)* | Указатель на массив, содержащий число вхождений каждого символа,  Алфавит,  *list \*lst* – структура линейного двусвязного списка, которая содержит указатели на предыдущий и последующий элементы, а также структуру корня дерева,  Указатель на начало списка,  Буква, встречающаяся в тексте максимальное количество раз | - | Функция заполняет список корнями дерева по возрастанию числа вхождения символов в текст |
| *struct list \* delete\_two\_elements(list \*lst)* | Указатель на начало списка | Указатель на новое начало списка | Функция удаляет два первых элемента списка.  Если элементов осталось два, то удаляет только один. Если остался только один элемент, то ничего не удаляет. |
| *struct list \* make\_new\_node(list \*lst)* | Указатель на начало списка | Указатель на новый элемент списка | Функция создаёт новый элемент списка, содержащий два поддерева, и возвращает его. |
| *struct list \* paste\_node (list \*lst, list \*new\_node)* | Указатель на начало списка,  Указатель на новый элемент списка | Указатель на новый элемент списка | Функция вставляет новый элемент в зависимости от числа вхождений буквы в текст, чтобы не выполнять сортировку заново. |
| *struct list \* start (list \*lst)* | Указатель на элемент списка | Указатель на первый элемент списка | Функция ищет первый элемент списка и возвращает его. |
| *struct list \* init (int data, char symbol)* | Число вхождений буквы в текст,  Буква | Указатель на новый элемент списка | Функция инициализации списка. Создаётся первый элемент типа *list*, имеющий во вложенной структуре дерева букву и число вхождений буквы в текст |
| *struct list \* addelem(list \*lst, int data, char symbol)* | Указатель на последний элемент списка,  Число вхождений буквы в текст,  Буква | Указатель на новый элемент списка | Функция добавляет элемент в список. Создаётся элемент типа *list*, имеющий указатель на предыдущий элемент, а также во вложенной структуре дерева букву и число вхождений буквы в текст |
| *void listprint(list \*lst)* | Указатель на начало списка | - | Функция вывода элементов списка. |
| *struct dictionary \* init\_d(char letter, string code)* | Буква,  Код буквы | dictionary – структура-словарь, содержащая букву и код буквы, а также указатели на предыдущий и последующий элементы | Функция инициализации словаря.  Создаётся первый элемент типа *dictionary.* |
| *struct dictionary \* addelem\_d(dictionary \*dict, char letter, string code)* | Указатель на предыдущий элемент словаря,  Буква,  Код буквы | Указатель на новый элемент словаря | Функция создаёт новый элемент словаря с переданными параметрами. |
| *dictionary\* coding(node \*tree, string s, dictionary \*dict)* | node – структура элемента дерева, содержащая букву, число вхождений буквы в текст, указатели на левое и правое поддерево (лист), а также символ “0”(элемент слева от родителя) или “1”(элемент справа от родителя) | Указатель на элемент словаря | Функция заполняет словарь элементами дерева и их кодовыми “словами”. |
| *int find\_size(dictionary \*dict)* | Указатель на первый элемент словаря | Размер словаря | Функция поиска размера словаря. |
| *struct dictionary \* start\_d(dictionary \*dict)* | Указатель на элемент словаря. | Указатель на первый элемент словаря | Функция поиска указателя на первый элемент словаря. |
| *string make\_code(dictionary \*dict, string text)* | Указатель на первый элемент словаря,  Искомый текст | Закодирован-  ное сообщение | Функция кодирует каждую букву текста соответствующим ему “словом” и сохраняет в общую строку *code*, которую в последствии возвращает. |
| *void dict\_print(dictionary \*dict)* | Указатель на первый элемент словаря | - | Функция вывода элементов словаря. |
| *void encode\_text(string code)* | Закодирован-  ное сообщение | - | Функция запрашивает у пользователя название файла, куда впоследствии записывается закодированное сообщение. |
| *void encode\_table(dictionary \*dict)* | Указатель на первый элемент словаря | - | Функция запрашивает у пользователя название файла, куда впоследствии записывается кодовая таблица. |

1. **РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ**

Для проверки корректности работы приложения были использованы

тестовые данные, приведенные в таблице 3.

Таблица 3 – Тестовые данные

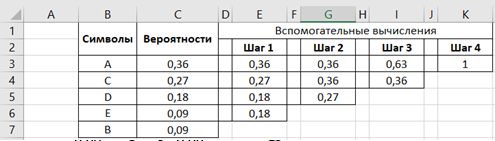
|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Название этапа тестирования** | **Тестовые данные** | **Ожидаемый результат** |
| 1 | Тестирование на маленьком тексте | Тестовые данные А | Вывод таблицы кодировки и закодированного сообщения в соответствующие файлы, результат которых будет соответствовать «Тестовым данным А» |
| 2 | Тестирование на большом тексте | Текст из файла “*hello.txt*” | Вывод таблицы кодировки и закодированного сообщения в соответствующие файлы |

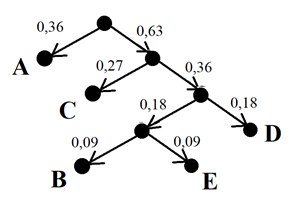
Тестовые данные А:

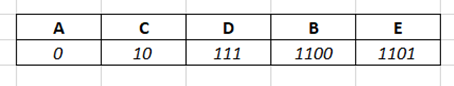
Искомый файл содержит пробную строку: «*AAAABCCCDDE*»

1. Отсортировать символы в порядке убывания их вероятности появления в тексте:

*AAAA, CCC, DD, B, E*;

1. Построить таблицу по правилу Д. Хаффмана:
2. Построить кодовое дерево:



1. Сделать таблицу кодов:
2. Ожидаемый результат:

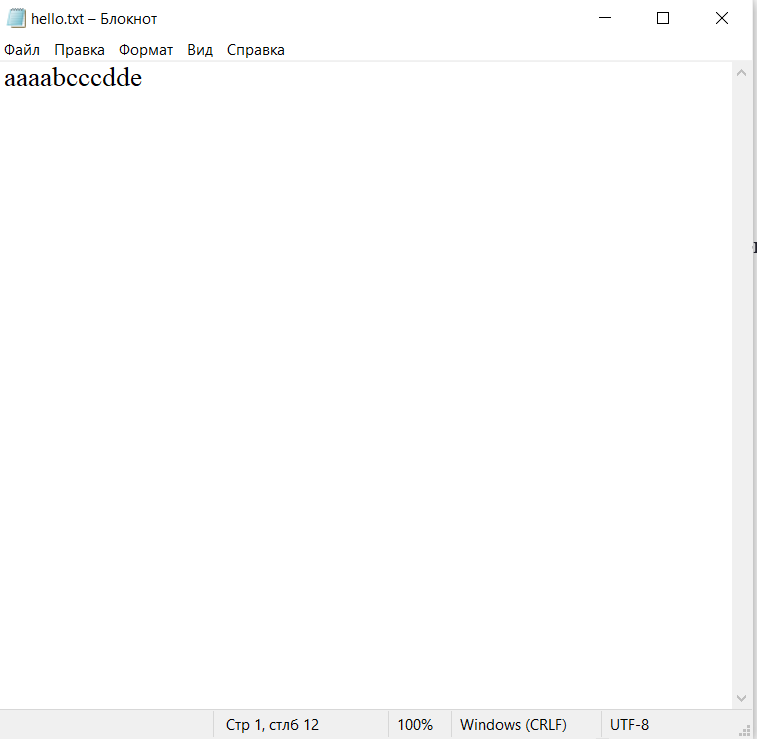
000011001010101111111101

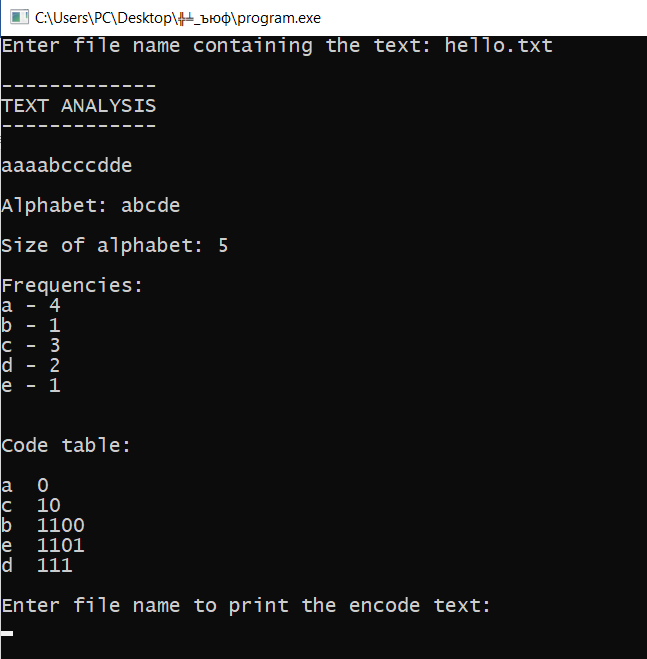
Результаты тестирования приложения по данным таблицы 1 приведены

ниже.

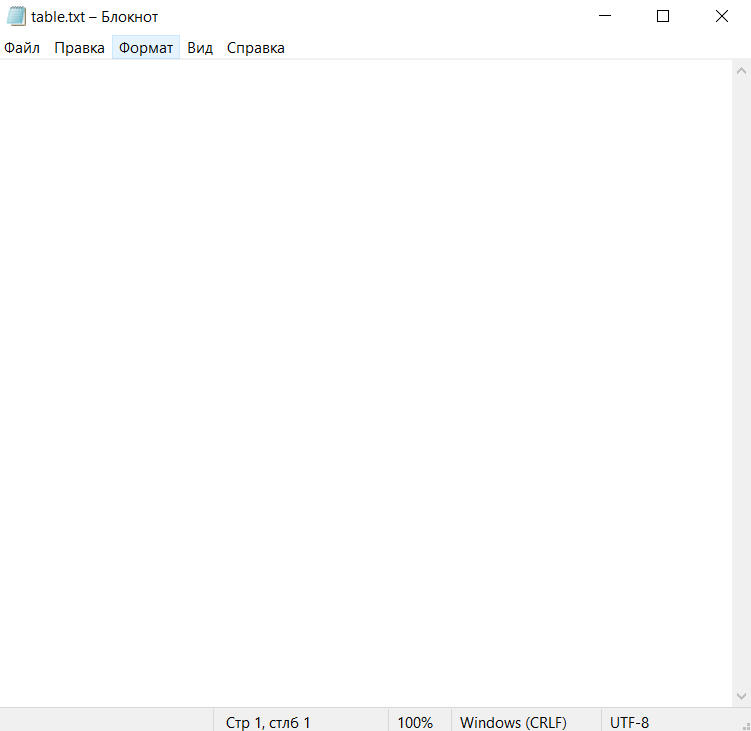
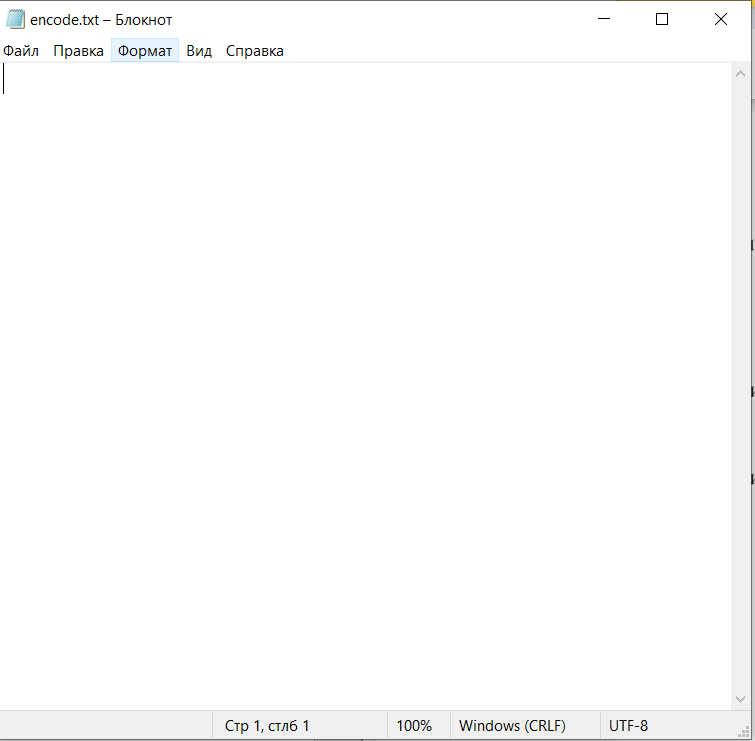
Результаты тестирования п. 1:

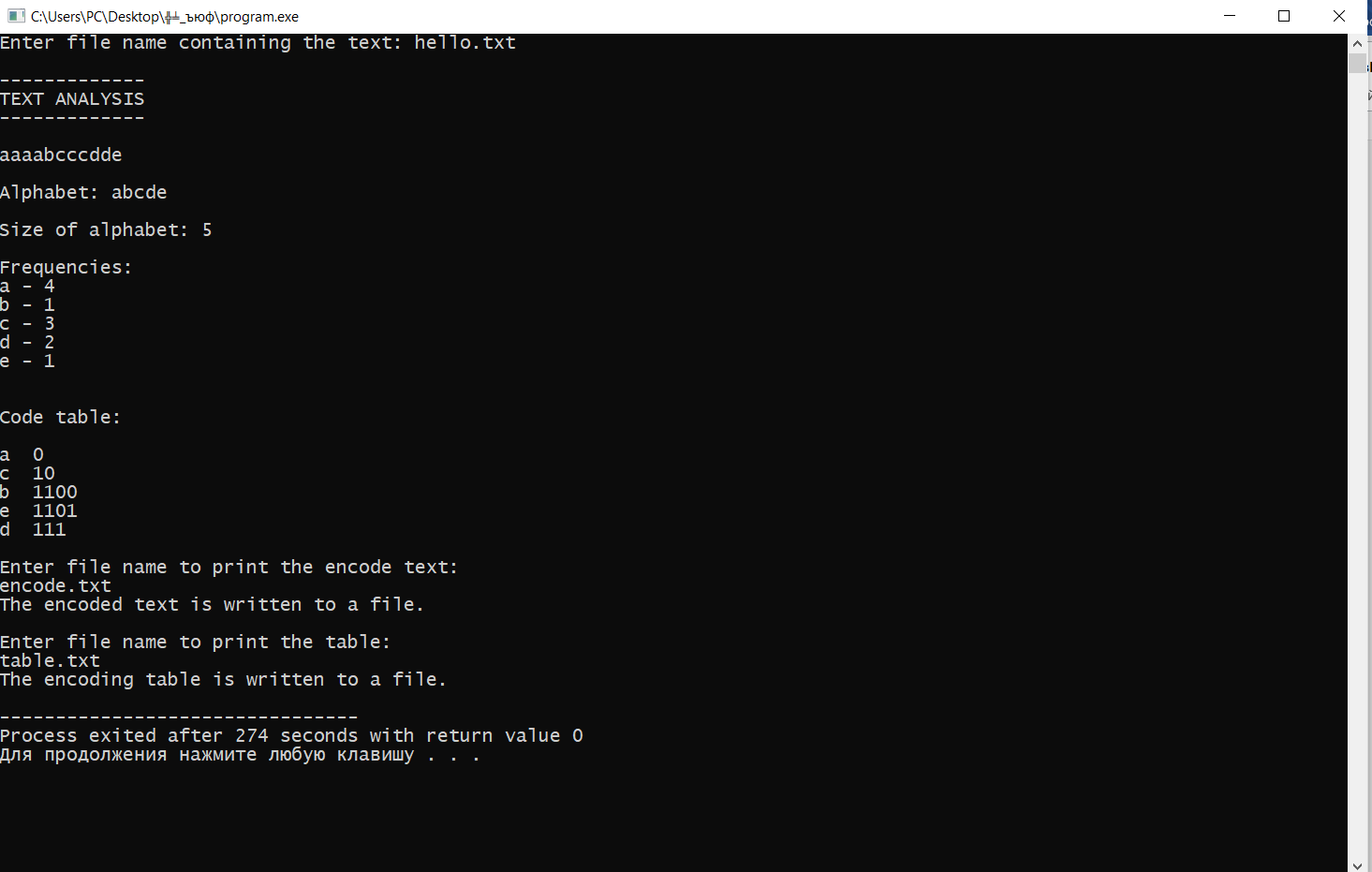
Искомый текст в файле “hello.txt”



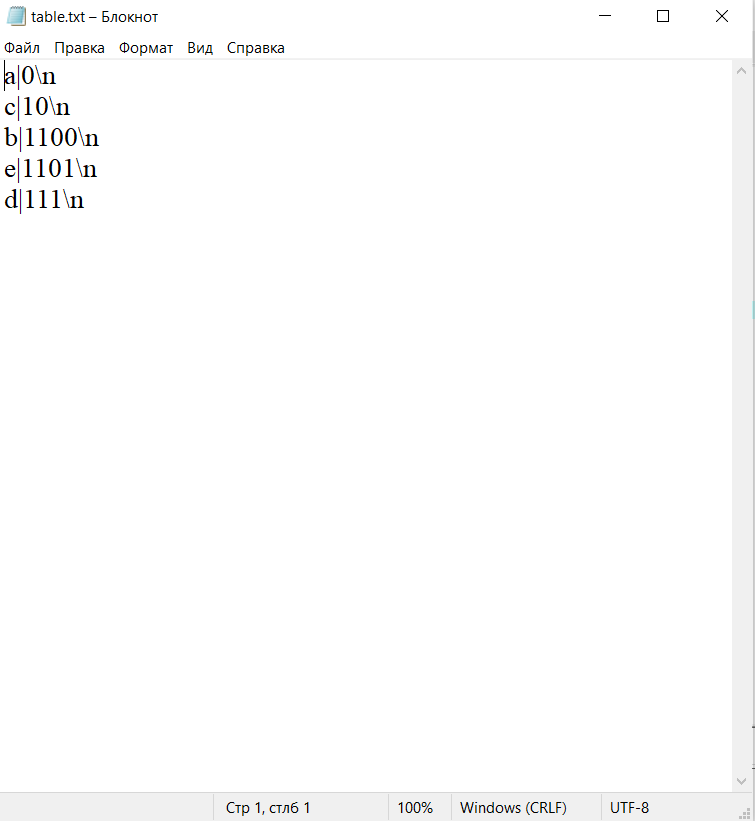
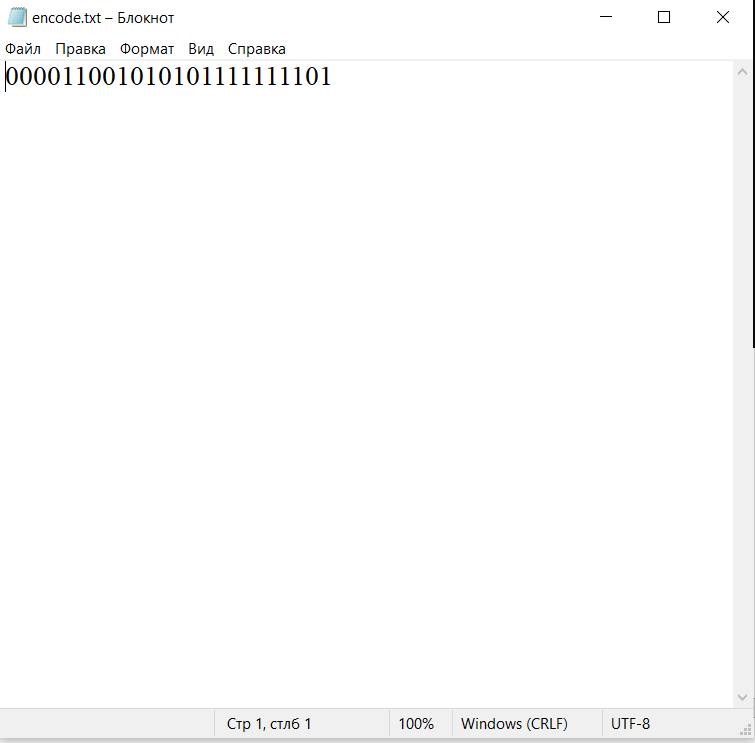


Файл для записи закодированного текста (*encode.txt*) и файл для записи таблицы кодировки (*table.txt*):



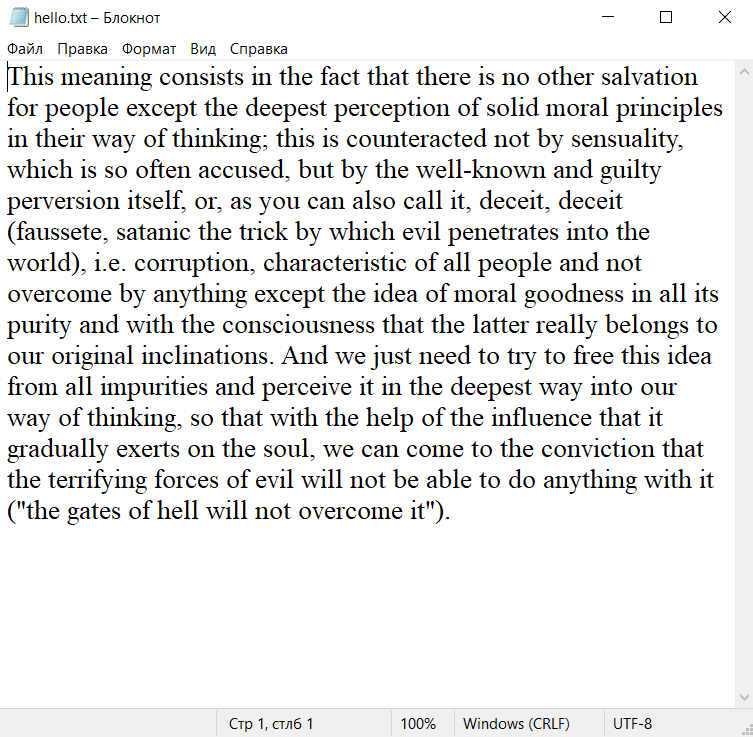


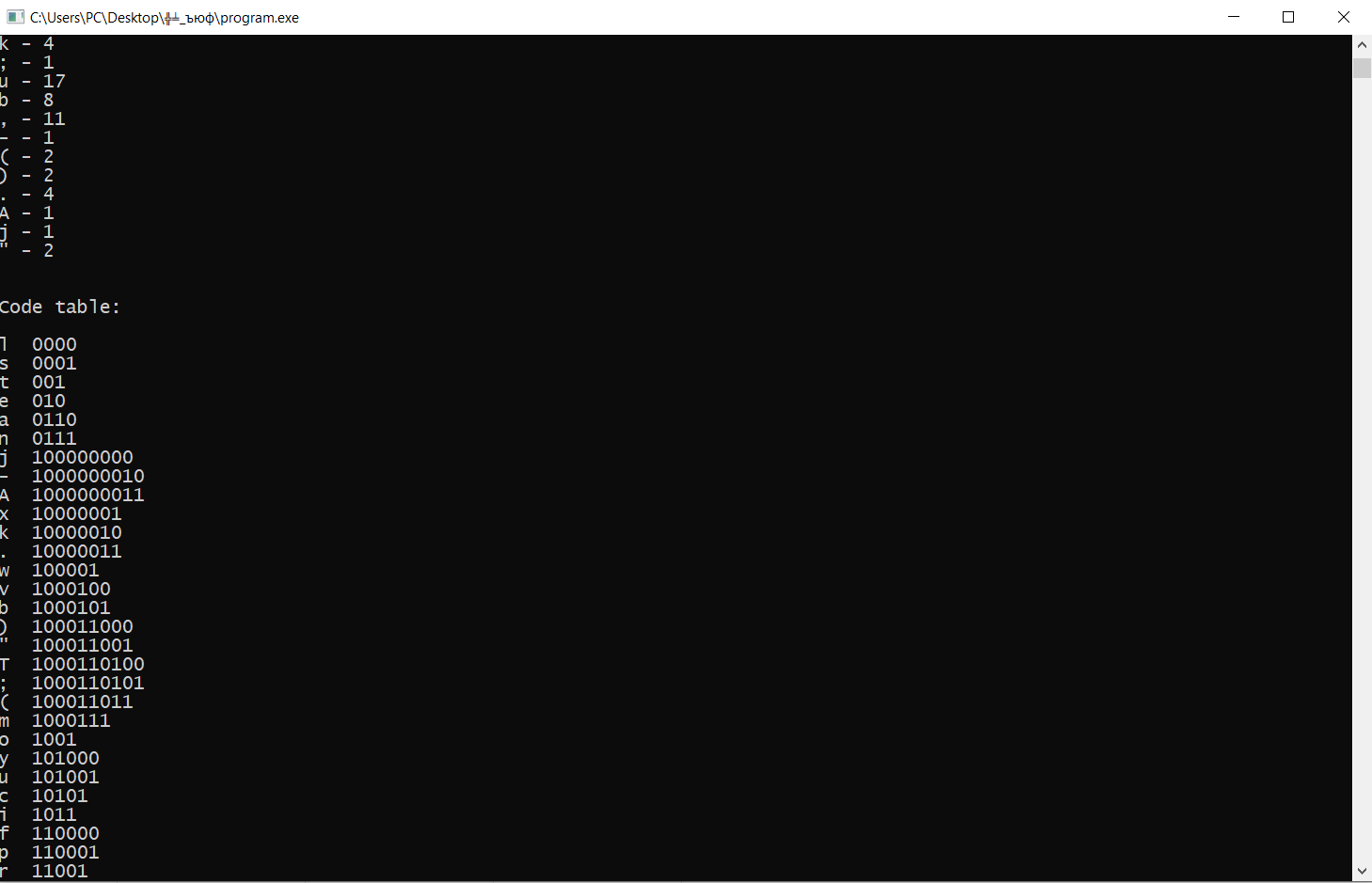
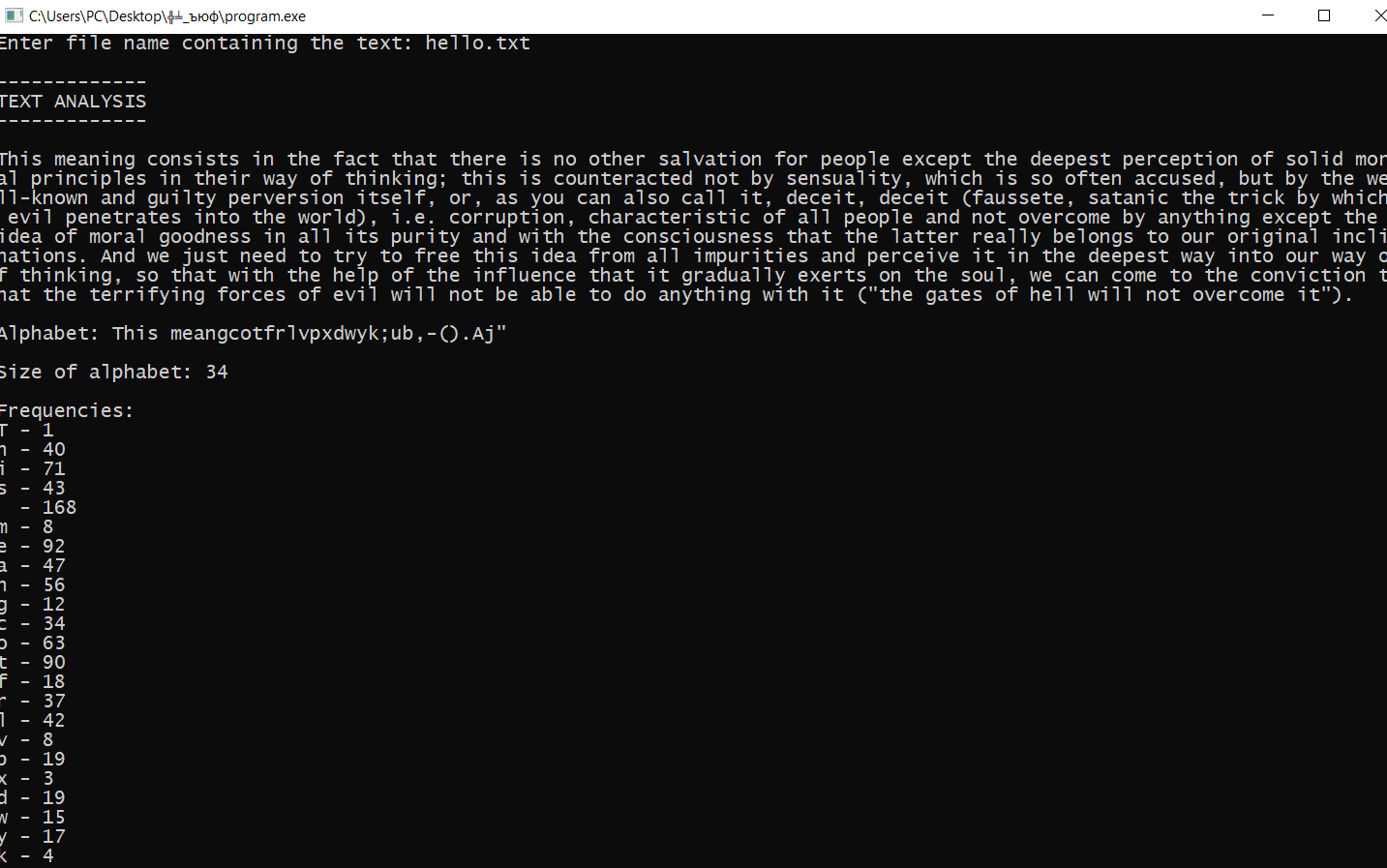
Результат:



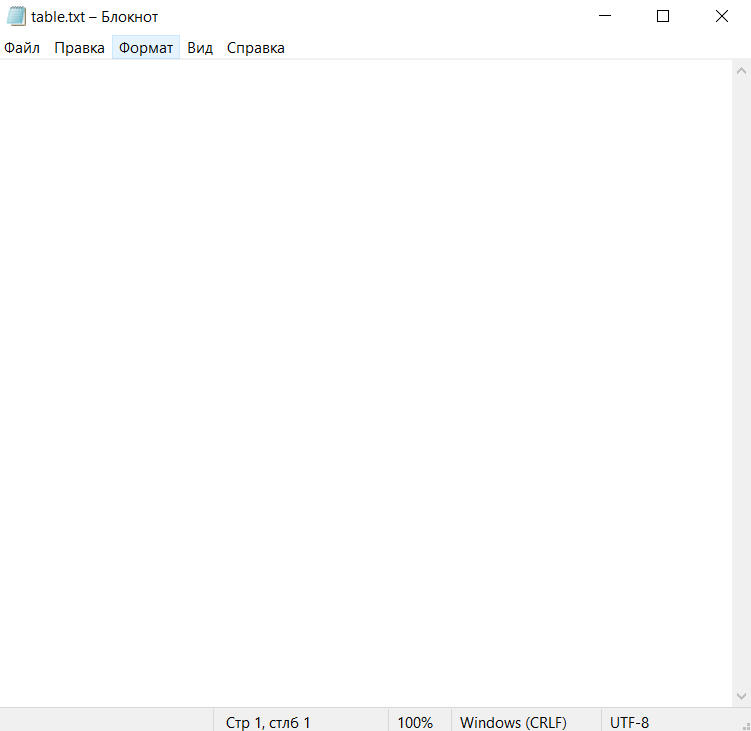
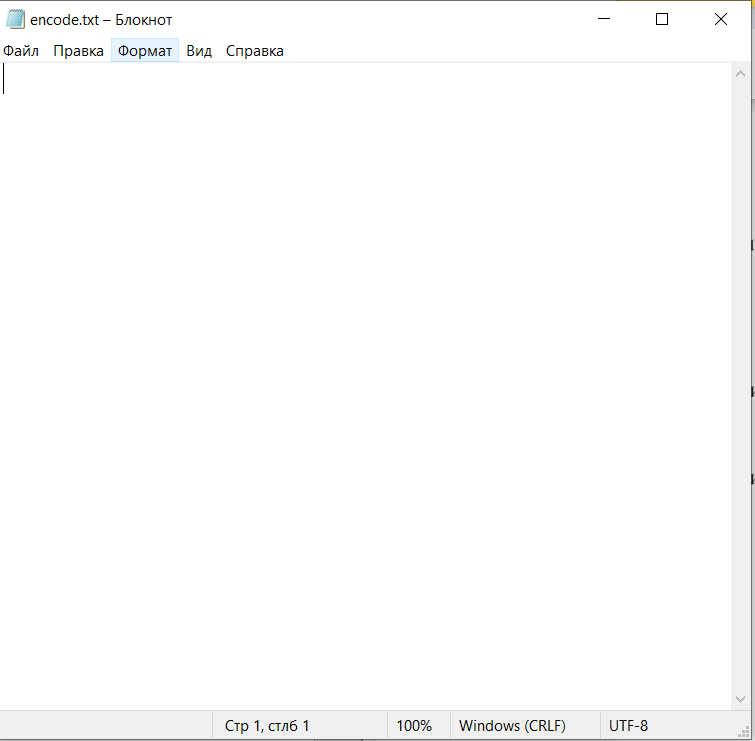
Результат вычислений вручную совпадает с результатом работы программы.

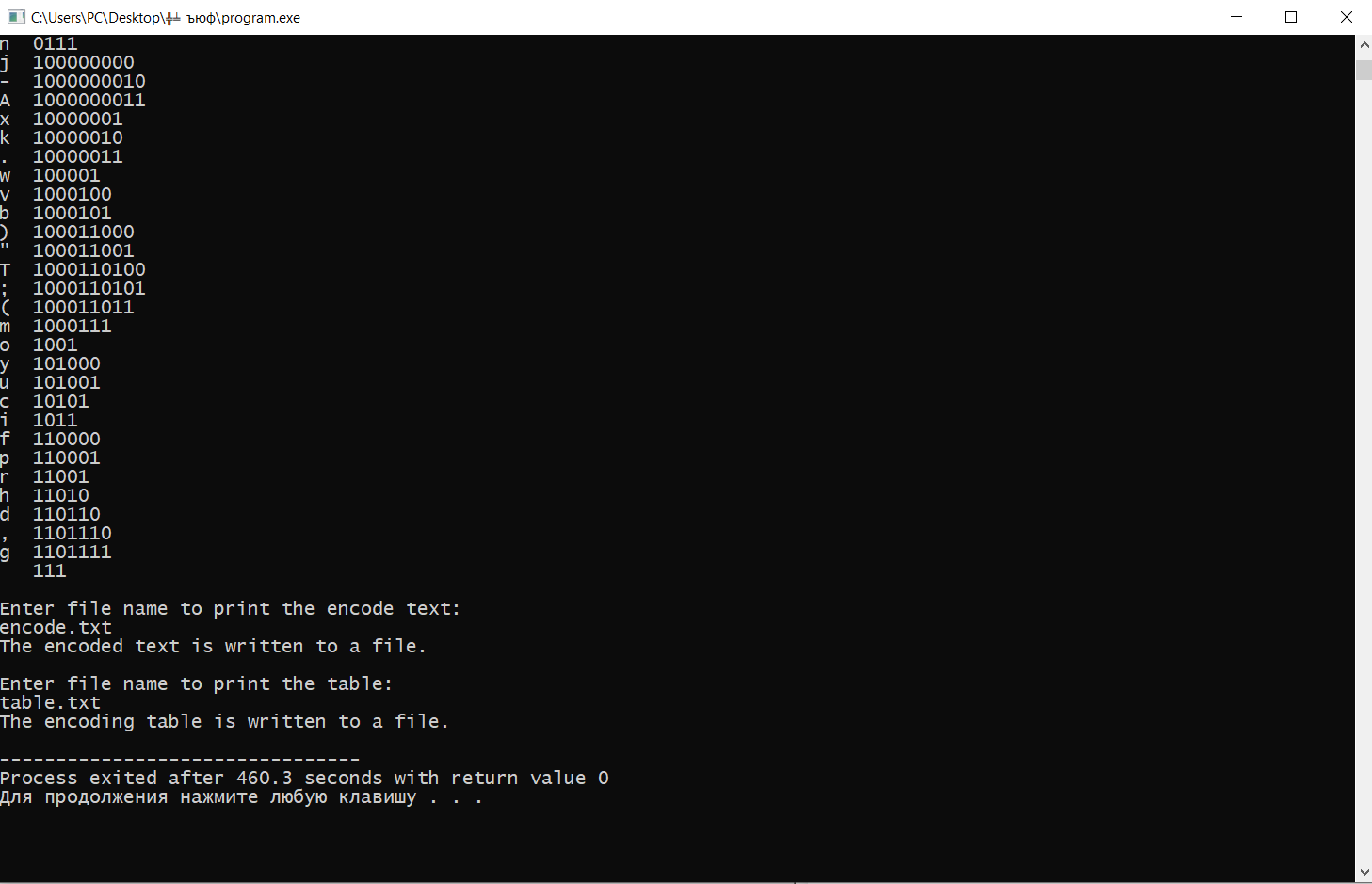
Результаты тестирования п. 2:

Искомый текст в файле “*hello.txt*”

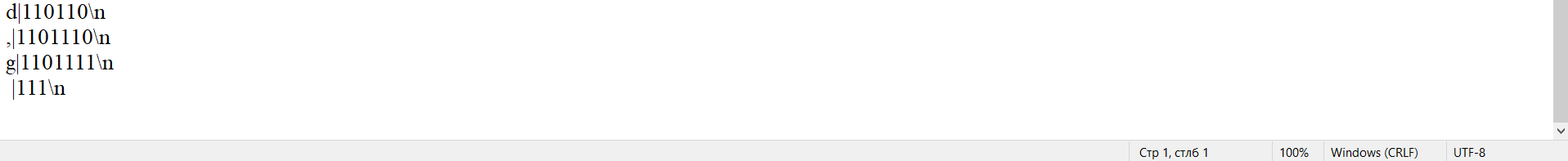
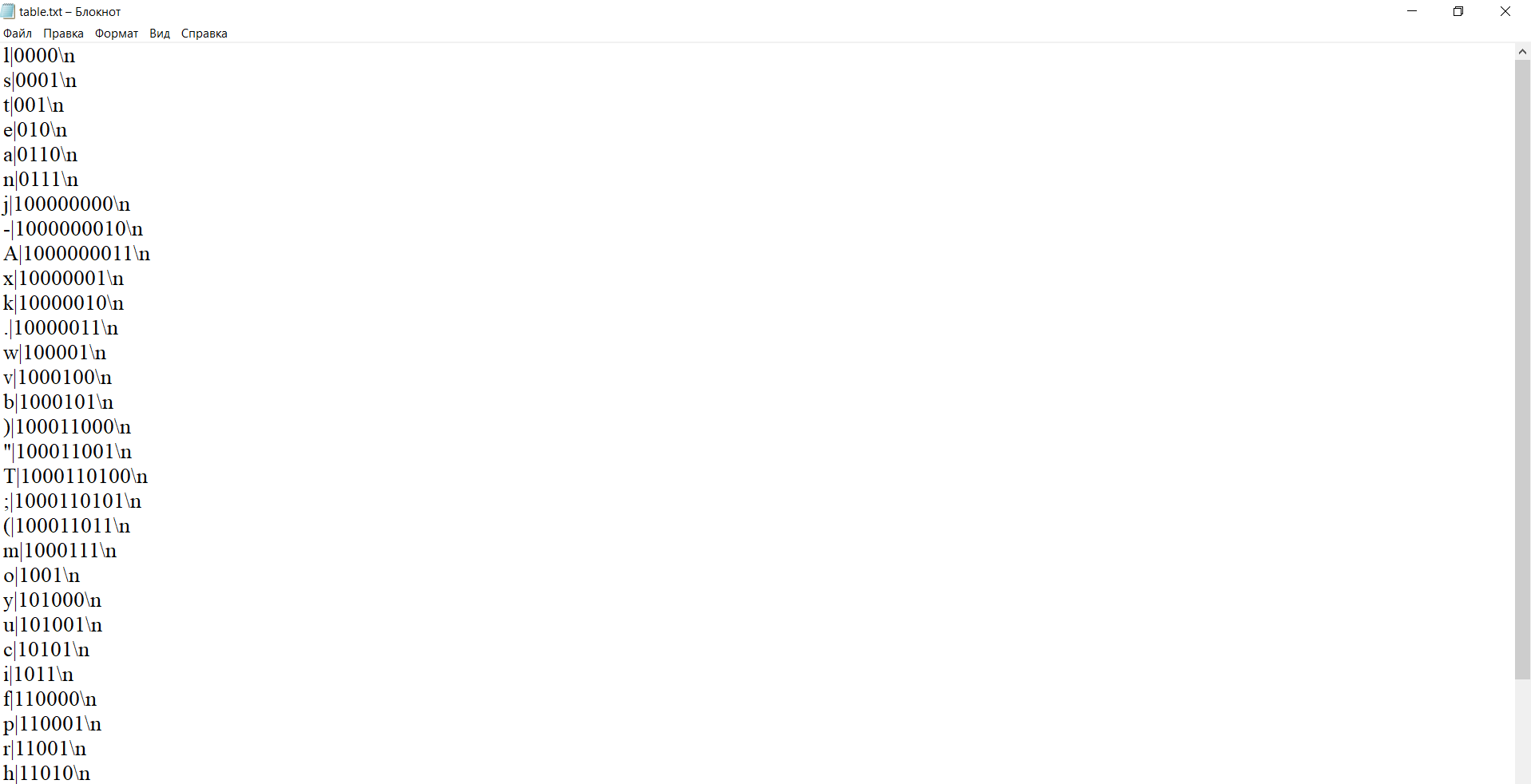


Файл для записи закодированного текста (*encode.txt*) и файл для записи таблицы кодировки (*table.txt*):





Результат:



# ЗАКЛЮЧЕНИЕ

Приложение выполняет эффективное кодирование сообщения на основании алгоритма Хаффмана. Были описаны некоторые сложные алгоритмы, а также указаны все функции программы. Приложение работает с любым текстовым сообщением, как было указано в разделе тестирования.

# СПИСОК ЛИТЕРАТУРЫ

1. Алгоритм Хаффмана на пальцах. URL: <https://habr.com/ru/post/144200/> (дата обращения: 02.11.2022).
2. Алгоритм Хаффмана. URL: <https://neerc.ifmo.ru/wiki/index.php?title=Алгоритм_Хаффмана> (дата обращения: 02.11.2022).
3. Двоичные деревья. URL: <https://cpp.com.ru/shildt_spr_po_c/22/2208.html> (дата обращения: 15.11.2022).
4. Обход бинарных деревьев: рекурсия, итерации и указатель на родителя. URL: <https://habr.com/ru/post/144850/> (дата обращения: 15.11.2022).
5. Чтение и запись текстовых файлов. URL: <https://metanit.com/cpp/tutorial/8.3.php> (дата обращения: 16.11.2022).

**ПРИЛОЖЕНИЕ А**

Листинг программы.

*#include <iostream>*

*#include <fstream>*

*#include<limits>*

*#include "list.cpp"*

*using namespace std;*

*int check\_numbers() {*

*int number;*

*while (true) {*

*cin >> number;*

*if (!cin.good()) {*

*cin.clear();*

*cin.ignore(numeric\_limits<streamsize>::max(),'\n');*

*cout << "The number is incorrect. Try again." << endl;*

*}*

*else break;*

*}*

*return number;*

*}*

*string get\_data(string filename) {*

*string text;*

*fstream input;*

*input.open(filename, ios::in);*

*while (!input.eof()) {*

*string temp;*

*getline(input, temp);*

*if (!input.eof()) {*

*temp += '\n';*

*}*

*text += temp;*

*}*

*input.close();*

*return text;*

*}*

*string alphabet(string text) {*

*string abc;*

*bool exit\_flag = 0;*

*abc.push\_back(text[0]);*

*for (int i = 1; i < text.size(); i++) {*

*exit\_flag = 0;*

*for (int j = 0; j < abc.size(); j++) {*

*if (abc[j] == text[i]) {*

*exit\_flag = 1;*

*break;*

*}*

*}*

*if (exit\_flag == 1) {*

*continue;*

*}*

*else {*

*abc.push\_back(text[i]);*

*continue;*

*}*

*}*

*return abc;*

*}*

*void get\_frequency(string text, int \*frequencies, string abc) {*

*int len = abc.size();*

*for (int i = 0; i < len; i++) {*

*frequencies[i] = 0;*

*}*

*for (int j = 0; j < len; j++) {*

*for (int i = 0; i < text.size(); i++) {*

*if (abc[j] == text[i]) {*

*frequencies[j]++;*

*}*

*}*

*}*

*}*

*void print\_abc(string text, string abc, int \*frequencies) {*

*cout << endl;*

*cout << text << endl;*

*cout << endl;*

*cout << "Alphabet: " << abc << endl;*

*cout << endl;*

*cout << "Size of alphabet: " << abc.size() << endl;*

*cout << endl;*

*cout << "Frequencies: " << endl;*

*for (int i = 0; i < abc.size(); i++) {*

*cout << abc[i] << " - " << frequencies[i] << endl;*

*}*

*cout << endl;*

*}*

*void make\_nodes(int \*frequencies, string abc, list \*lst, char g) {*

*int k = 0, counter = 1;*

*char already\_in\_list[abc.size()];*

*bool is\_in;*

*already\_in\_list[0] = lst->tree.symbol;*

*while (already\_in\_list[abc.size()-1] != g) {*

*for (int i = 0; i < abc.size(); i++) {*

*is\_in = true;*

*for (int j = 0; j < abc.size(); j++) {*

*if (abc[i] == already\_in\_list[j]) continue;*

*else is\_in = false;*

*}*

*if ((frequencies[i] == counter) && (is\_in == false)) {*

*addelem(lst, frequencies[i], abc[i]);*

*lst = lst->next;*

*already\_in\_list[k] = abc[i];*

*k++;*

*}*

*}*

*counter++;*

*}*

*}*

*struct list \* delete\_two\_elements(list \*lst) {*

*if (lst->next != NULL) {*

*lst = lst->next;*

*lst->prev = NULL;*

*if(lst->next != NULL) {*

*lst = lst->next;*

*lst->prev = NULL;*

*}*

*}*

*return lst;*

*}*

*struct list \* make\_new\_node(list \*lst) {*

*list \*new\_node;*

*new\_node = (struct list\*)malloc(sizeof(struct list));*

*node \*i;*

*i = (struct node\*)malloc(sizeof(struct node));*

*i->data = lst->tree.data;*

*i->symbol = lst->tree.symbol;*

*i->left = lst->tree.left;*

*i->right = lst->tree.right;*

*i->t = '0';*

*new\_node->tree.left = i;*

*lst = lst->next;*

*node \*j;*

*j = (struct node\*)malloc(sizeof(struct node));*

*j->data = lst->tree.data;*

*j->symbol = lst->tree.symbol;*

*j->left = lst->tree.left;*

*j->right = lst->tree.right;*

*j->t = '1';*

*new\_node->tree.right = j;*

*new\_node->tree.data = i->data + j->data;*

*new\_node->tree.symbol = '~';*

*return new\_node;*

*}*

*struct list \* paste\_node (list \*lst, list \*new\_node) {*

*bool r = true;*

*while(r) {*

*if ((new\_node->tree).data <= (lst->tree).data) {*

*if (lst->prev == NULL) {*

*lst->prev = new\_node;*

*new\_node->next = lst;*

*new\_node->prev = NULL;*

*r = false;*

*}*

*else {*

*list \*p;*

*p = lst->prev;*

*new\_node->next = lst;*

*lst->prev = new\_node;*

*p->next = new\_node;*

*new\_node->prev = p;*

*r = false;*

*}*

*}*

*else if (lst->next != NULL){*

*lst = lst->next;*

*}*

*else {*

*lst->next = new\_node;*

*new\_node->prev = lst;*

*new\_node->next = NULL;*

*r = false;*

*}*

*}*

*return new\_node;*

*}*

*struct list \* start(list \*lst) {*

*while (lst->prev != NULL) {*

*lst = lst->prev;*

*}*

*return lst;*

*}*

*struct node {*

*int data;*

*char symbol;*

*struct node \*left;*

*struct node \*right;*

*char t;*

*};*

*struct list {*

*struct list \*next;*

*struct list \*prev;*

*node tree;*

*};*

*struct list \* init(int data, char symbol){*

*struct list \*lst;*

*lst = (struct list\*)malloc(sizeof(struct list));*

*lst->next = NULL;*

*lst->prev = NULL;*

*lst->tree.data = data;*

*lst->tree.symbol = symbol;*

*lst->tree.left = NULL;*

*lst->tree.right = NULL;*

*return(lst);*

*};*

*struct list \* addelem(list \*lst, int data, char symbol){*

*struct list \*tem, \*p;*

*tem = (struct list\*)malloc(sizeof(struct list));*

*p = lst->next;*

*lst->next = tem;*

*tem->next = p;*

*tem->prev = lst;*

*if (p != NULL) p->prev = tem;*

*tem->tree.data = data;*

*tem->tree.symbol = symbol;*

*tem->tree.left = NULL;*

*tem->tree.right = NULL;*

*return(tem);*

*};*

*void listprint(list \*lst){*

*do {*

*cout << lst->tree.data << " " << lst->tree.symbol << endl;*

*lst = lst->next;*

*} while (lst != NULL);*

*}*

*struct dictionary {*

*char letter;*

*string code;*

*struct dictionary \*next;*

*struct dictionary \*prev;*

*};*

*struct dictionary \* init\_d(char letter, string code) {*

*struct dictionary \*dict;*

*dict = (struct dictionary\*)malloc(sizeof(struct dictionary));*

*dict->next = NULL;*

*dict->prev = NULL;*

*dict->code = code;*

*dict->letter = letter;*

*return(dict);*

*}*

*struct dictionary \* addelem\_d(dictionary \*dict, char letter, string code){*

*struct dictionary \*tem;*

*tem = (struct dictionary\*)malloc(sizeof(struct dictionary));*

*dict->next = tem;*

*tem->prev = dict;*

*tem->next = NULL;*

*tem->code = code;*

*tem->letter = letter;*

*return(tem);*

*};*

*dictionary\* coding(node \*tree, string s, dictionary \*dict) {*

*if (tree != NULL) {*

*s += tree->t;*

*dict = coding(tree->left, s, dict);*

*dict = coding(tree->right, s, dict);*

*if (tree->symbol != '~') {*

*string new\_s;*

*for (int i = 1; i < s.length(); i++){*

*new\_s += s[i];*

*}*

*dict = addelem\_d(dict, tree->symbol, new\_s);*

*s = "";*

*// cout << dict->letter << " " << dict->code << endl;*

*return dict;*

*}*

*}*

*return dict;*

*}*

*int find\_size(dictionary \*dict) {*

*int counter = 1;*

*while (dict->next != NULL) {*

*counter++;*

*dict = dict->next;*

*}*

*return counter;*

*}*

*struct dictionary \* start\_d(dictionary \*dict) {*

*while (dict->prev != NULL) dict = dict->prev;*

*return dict;*

*}*

*string make\_code(dictionary \*dict, string text) {*

*string code;*

*int i = 0;*

*int size\_dict = find\_size(dict);*

*while (text[i] != '\0') {*

*for (int j = 0; j < size\_dict; j++) {*

*if (text[i] == dict->letter) {*

*code += dict->code;*

*break;*

*}*

*dict = dict->next;*

*}*

*dict = start\_d(dict);*

*i++;*

*}*

*return code;*

*}*

*void dict\_print(dictionary \*dict) {*

*do {*

*cout << dict->letter << " " << dict->code << endl;*

*dict = dict->next;*

*} while (dict != NULL);*

*}*

*void encode\_text(string code) {*

*string filename\_code;*

*cout << "Enter file name to print the encode text: " << endl;*

*cin >> filename\_code;*

*std::ofstream out; // поток для записи*

*out.open(filename\_code, ios::out); // окрываем файл для записи*

*if (out.is\_open()) {*

*out << code << std::endl;*

*}*

*cout << "The encoded text is written to a file." << endl;*

*}*

*void encode\_table(dictionary \*dict) {*

*string filename\_code;*

*cout << "Enter file name to print the table: " << endl;*

*cin >> filename\_code;*

*std::ofstream out; // поток для записи*

*out.open(filename\_code, ios::out); // окрываем файл для записи*

*if (out.is\_open()) {*

*while (dict != NULL) {*

*out << dict->letter << "|" << dict->code << "\\n" << std::endl;*

*dict = dict->next;*

*}*

*}*

*cout << "The encoding table is written to a file." << endl;*

*}*

*int main() {*

*cout << "Enter file name containing the text: ";*

*string filename, text;*

*cin >> filename;*

*text = get\_data(filename); // Считываение текста из файла*

*string abc = alphabet(text);*

*int \*frequencies = new int[abc.size()];*

*get\_frequency(text, frequencies, abc); // Число вхождений букв в текст*

*cout << endl;*

*cout << "-------------" << endl;*

*cout << "TEXT ANALYSIS" << endl;*

*cout << "-------------" << endl;*

*print\_abc(text, abc, frequencies);*

*int mini = 9999;*

*int maxi = 0;*

*int j;*

*char g;*

*for (size\_t i = 0; i < abc.size(); i++) { // Поиск самой редкой и самой частой бувкы*

*if (mini > frequencies[i]) {*

*mini = frequencies[i];*

*j = i;*

*}*

*if (maxi < frequencies[i]) {*

*maxi = frequencies[i];*

*g = abc[i];*

*}*

*}*

*list \*head, \*cur;*

*head = init(mini, abc[j]);*

*cur = head;*

*make\_nodes(frequencies, abc, cur, g); // Создание списка корней*

*head = head->next;*

*head->prev = NULL;*

*cur = head;*

*list \*new\_node;*

*while(true) { // Создание дерева*

*cur = head;*

*if (cur->next->next == NULL) {*

*new\_node = make\_new\_node(cur);*

*break;*

*}*

*new\_node = make\_new\_node(cur);*

*head = delete\_two\_elements(head);*

*cur = head;*

*new\_node = paste\_node(cur, new\_node);*

*head = start(head);*

*}*

*cur = head;*

*head = new\_node;*

*head->prev = NULL;*

*string s;*

*dictionary \*dict, \*head\_d = dict;*

*node \*tree\_d = &(head->tree);*

*while (tree\_d->left != NULL) { // Поиск самого левого эл-та дерева для инициализации*

*tree\_d = tree\_d->left;*

*s += '0';*

*}*

*head\_d = init\_d(tree\_d->symbol, s);*

*s = "";*

*dict = head\_d;*

*dict = coding(&(head->tree), s, dict); // Функция заполнения словаря, содержащего буквы и соответствующие им кодовые слова*

*dict = start\_d(dict);*

*dict = dict->next;*

*dict->prev = NULL;*

*cout << endl;*

*cout << "Code table:" << endl;*

*cout << endl;*

*dict\_print(dict);*

*s = make\_code(dict, text); // Кодировка текста*

*cout << endl;*

*encode\_text(s); // Запись закодированного текста в файл*

*cout << endl;*

*encode\_table(dict); // Запись таблицы кодировки в файл*

*return 0;*

*}*