**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра Вычислительной техники**

**Курсовая работа**

**по дисциплине «Программирование»**

**Тема: Обработка текстовой информации**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3312 |  | Шарапов И. Д. |
| Преподаватель |  | Аббас С. А. |

Санкт-Петербург

2023

**Содержание**

[Цель работы 3](#_Toc153717035)

[Задание (Вариант 51) 3](#_Toc153717036)

[Постановка задачи и описание решения 3](#_Toc153717037)

[Структура вызова функций 5](#_Toc153717038)

[Описание переменных 5](#_Toc153717039)

[Схема алгоритма 6](#_Toc153717040)

[Текст программы 13](#_Toc153717041)

[Контрольные примеры 16](#_Toc153717042)

[Содержимое файлов 17](#_Toc153717043)

[Примеры выполнения программы 19](#_Toc153717044)

[Выводы 20](#_Toc153717045)

# Цель работы

Законченное поэтапное решение содержательной задачи (постановка задачи, спецификация, выбор структур данных и разработка алгоритма, программная реализация, тестирование).

# Задание (Вариант 51)

Ввести заданное количество ключевых слов и строку символов-разделителей. Затем вводится текст с неизвестным количеством строк. Ввод текста заканчивается, если после ввода строки в тексте окажется в любой последовательности все ключевые слова. Из строк введённого текста, в которых встречается хотя бы одно ключевое слово, удалить слово, имеющее минимальную длину. Вывести преобразованный текст.

# Постановка задачи и описание решения

Решение предполагает считывание из файла и консоли: для этого у пользователя спрашивается режим и имя файла, в случае ввода из файла. Далее программа считывает количество ключевых слов и сами ключевые слова. В процессе считывания каждого слова, программа считает и запоминает хэш этого слова по следующему алгоритму.

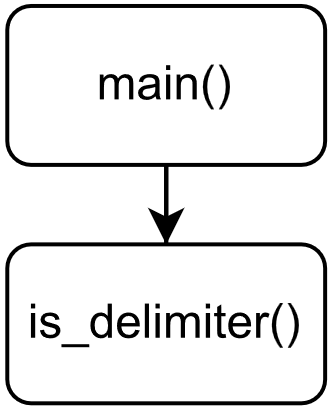
Хэш – это число, полученное из слова, как результат хэш-функции. Каждому числу соответствует своё слово: у разных упорядоченных наборов символов разный хэш. Для вычисления хеша слова переберём по порядку все его символы. Каждый символ имеет числовой код, пусть это будет x. Возьмём два простых числа r = 109+9 и t = 41. Здесь r – модуль хэша, t – основание хэша. Заведём переменную hash = 0 и на каждом шаге перебора будем вычислять её значение по следующей формуле:. При таком алгоритме могут возникать так называемые коллизии – это когда два разных слова соответствуют одному числу. Это возникает из-за того, что мы берём значение по модулю r. Усовершенствуем наш алгоритм и заведём второй хэш hash2 с основанием b = 47. Теперь на каждом шаге мы будем также считать и . Таким образом мы сводим вероятность коллизий к нулю.

После считывания и обработки ключевых слов считаем строку разделителей. Далее начнём считывать текст построчно. Считав новую строчку, мы начнём перебирать её посимвольно и считать хэш между разделителями (слов). Когда мы находим разделитель (проверку на то, что символ разделитель выполним отдельной функцией), мы смотрим значение хэша «слова» и перебираем все ранее посчитанные хэши ключевых слов. Если хэши по обоим основаниям совпадают, то мы прибавляем 1 к счётчику этого ключевого слова. Также мы проверяем является ли это «слово» минимальным в строке. Если является, то мы запоминаем указатель на начало и конец этого слова. И обнуляем значение хэшей. Если символ не разделитель, то мы дальше продолжаем считать хэш по описанному ранее алгоритму. Обработав строчку нам нужно понять было ли в ней ключевое слово: для этого мы сложим все счётчики ключевых слов и сравним с тем значением, что у нас было на прошлом ходу. Если ничего не поменялось, то мы в массиве, в котором запоминали указатели на минимальное слово, присваиваем этим указателем значение на начало строки. И в самом конце работы над строчкой нам надо понять считывать ли дальше. Для этого переберём все счётчики ключевых слов, и если хотя бы какой-то из них равен нулю, то мы продолжим считывание.

После считывания и обработки строк осталось их только вывести. Спросим у пользователя тип вывода (в файл или консоль). Начнём перебирать все строки и выводить их элементы (символы). Если указатель на текущий элемент совпадёт с указателем на начало минимального слова, то мы перенесём указатель на конец слова и продолжим вывод символов. Если вывод был в файл, то выведем в консоль сообщение об успешном завершении работы программы.

Проверка слов на равенство с использованием хэшей, по сравнению с посимвольным сравнением, имеет лучшую асимптотику. O(K\*T + N\*K) против O(N2\*K\*T), где K, T – количество и длина ключевых слов, N – длина текста в символах.

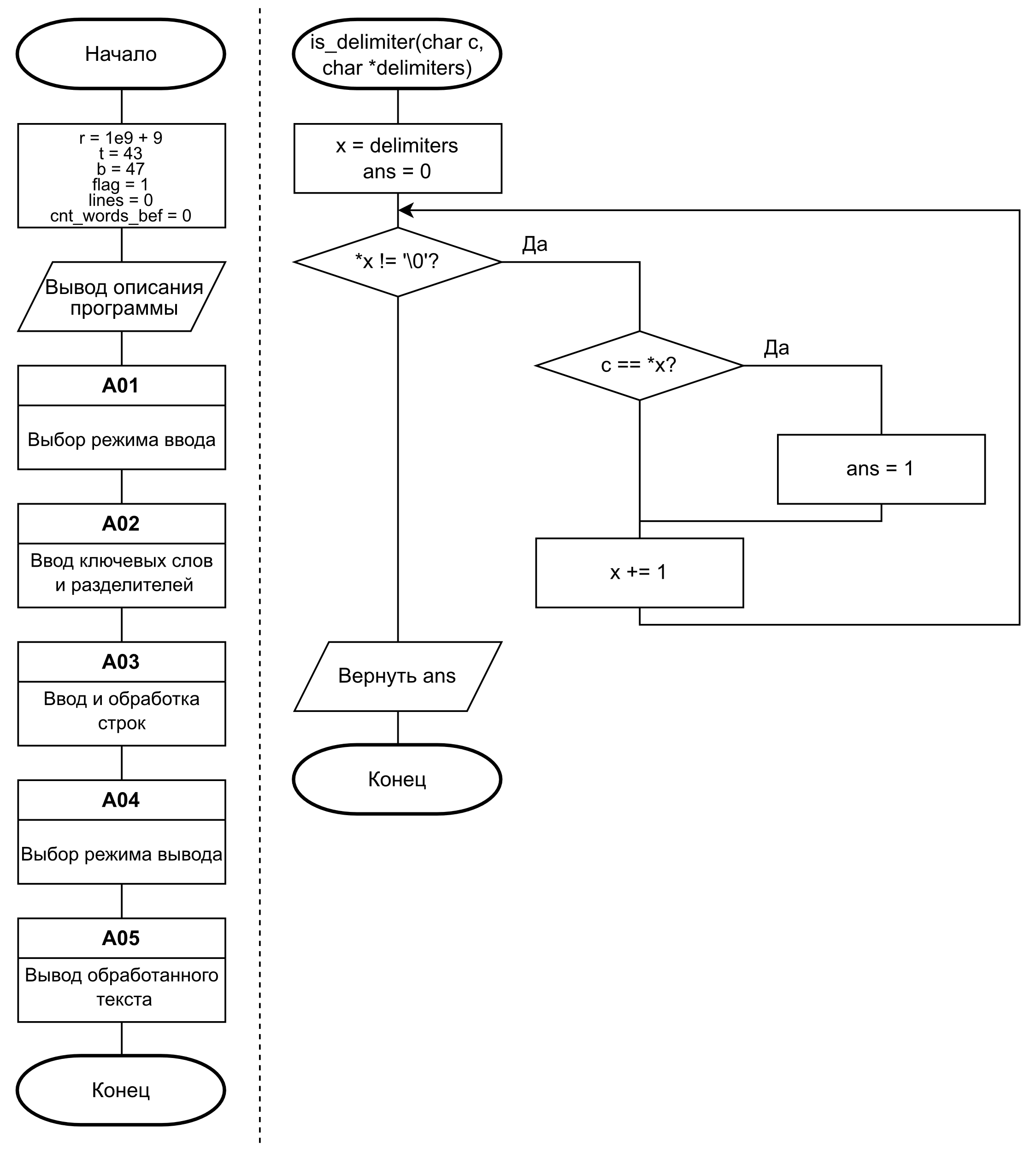
# Структура вызова функций

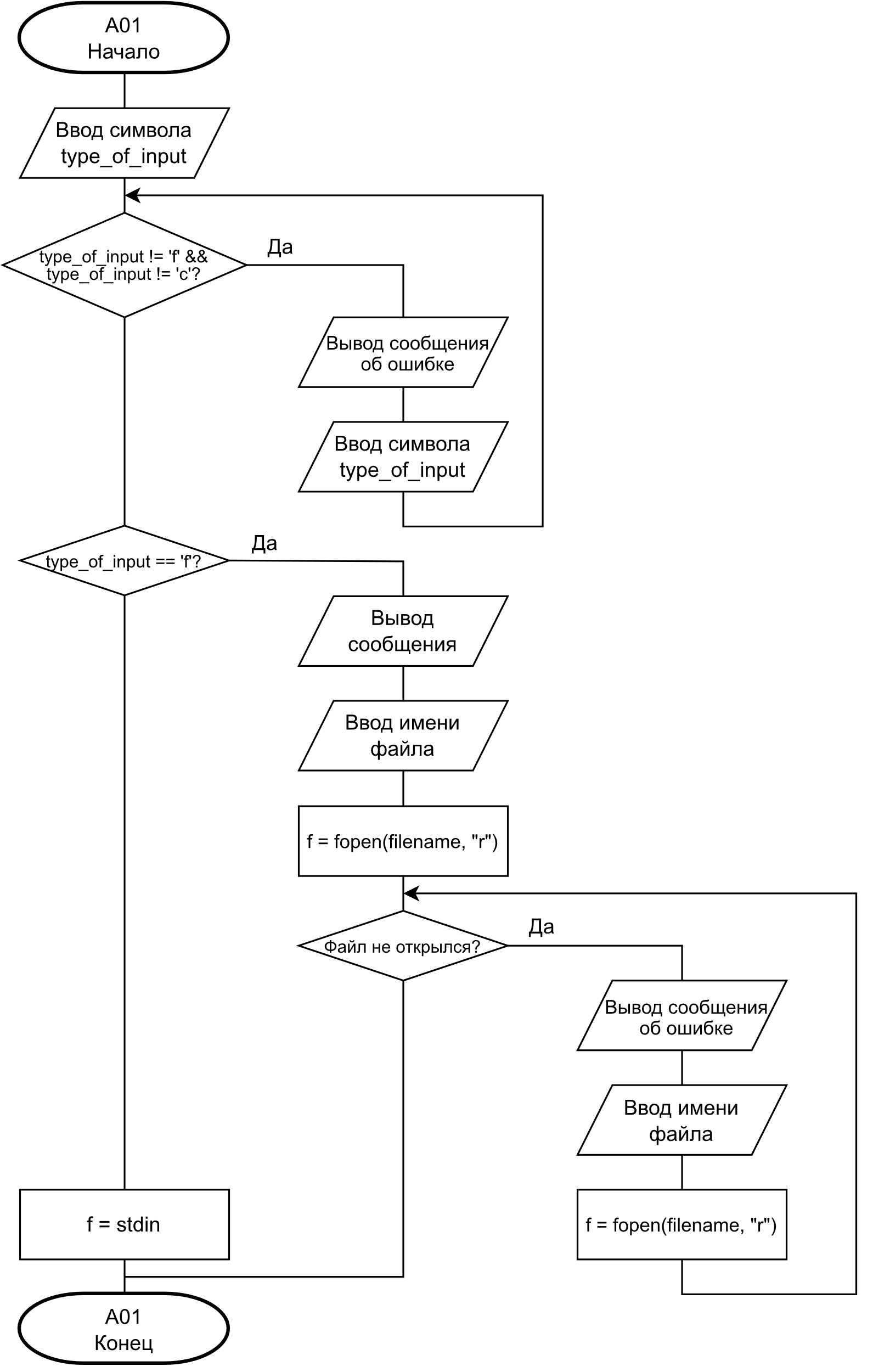


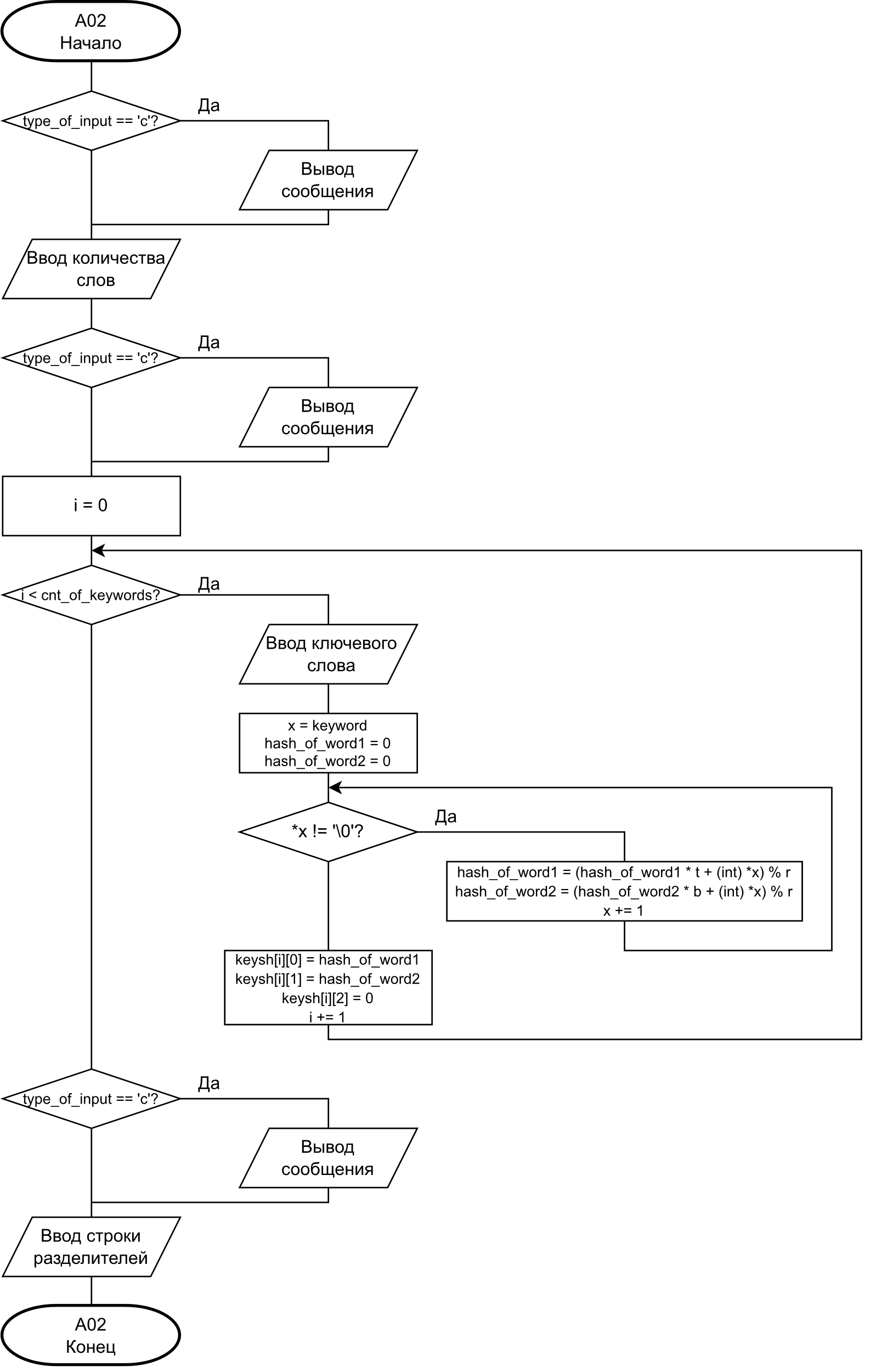
# Описание переменных

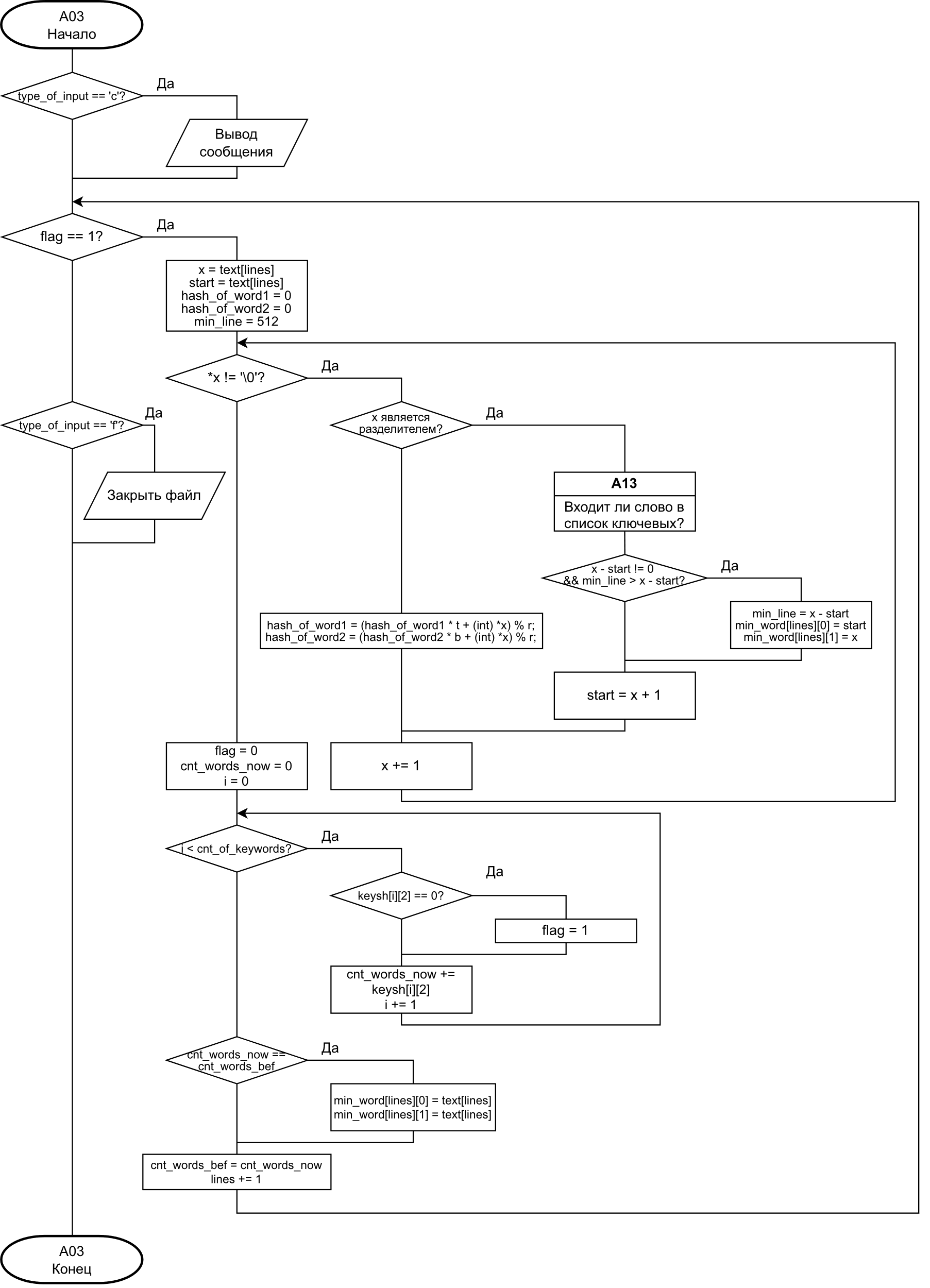
|  |  |  |  |
| --- | --- | --- | --- |
| Функция *int is\_delimiter(char c, char \*delimiters)* | | | |
| № | Имя переменной | Тип | Назначение |
| 1 | c | char | Проверяем: есть ли этот символ в массиве |
| 2 | delimiters | char\* | Массив разделителей |
| 3 | x | char\* | Указатель для перебора всех элементов |
| 4 | ans | int | Логическая переменная |
| Функция *int main()* | | | |
| № | Имя переменной | Тип | Назначение |
| 1 | r | int | Модуль хеширования |
| 2 | t | int | Первое основание хеша |
| 3 | b | int | Второе основание хеша |
| 4 | type\_of\_input | char | Символьная переменная для типа ввода |
| 5 | filename | char[] | Имя файла |
| 6 | f | file | Указатель на файл |
| 7 | cnt\_of\_keywords | int | Количество ключевых слов |
| 8 | keyword | char[] | Ключевое слово |
| 9 | x | char\* | Указатель на символ |
| 10 | hash\_of\_word1 | long long | Первое значения хеша |
| 11 | hash\_of\_word2 | long long | Второе значения хеша |
| 12 | keysh | int[][] | Двумерный массив для хранения значений хеша и счетчиков |
| 13 | delimiters | char[] | Массив для хранения разделителей |
| 14 | flag | int | Флаг для ограничения ввода |
| 15 | lines | int | Количество строк |
| 16 | text | char[][] | Двумерный массив для хранения текста |
| 17 | start | char\* | Указатель на начало слова |
| 18 | min\_line | int | Минимальная длина слова в строке |
| 19 | cnt\_words\_bef | int | Количество найденных слов до этого момента |
| 20 | cnt\_words\_now | int | Количество найденных слов сейчас |
| 21 | min\_word | char\*[][] | Двумерный массив для хранения указателей на начало и конец минимального слова в каждой строке |
| 22 | type\_of\_output | char | Символьная переменная для типа вывода |

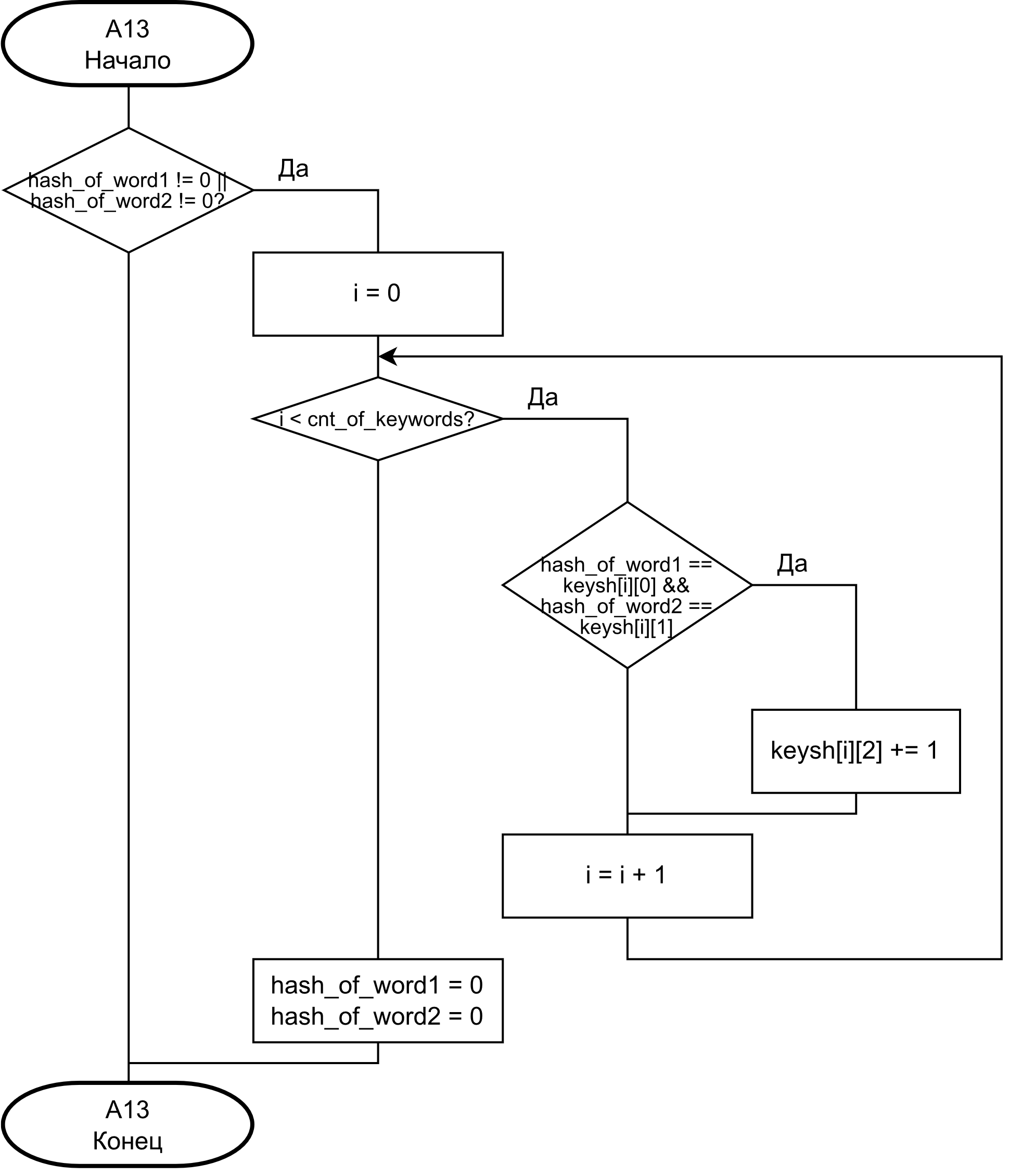
# **Схема алгоритма**

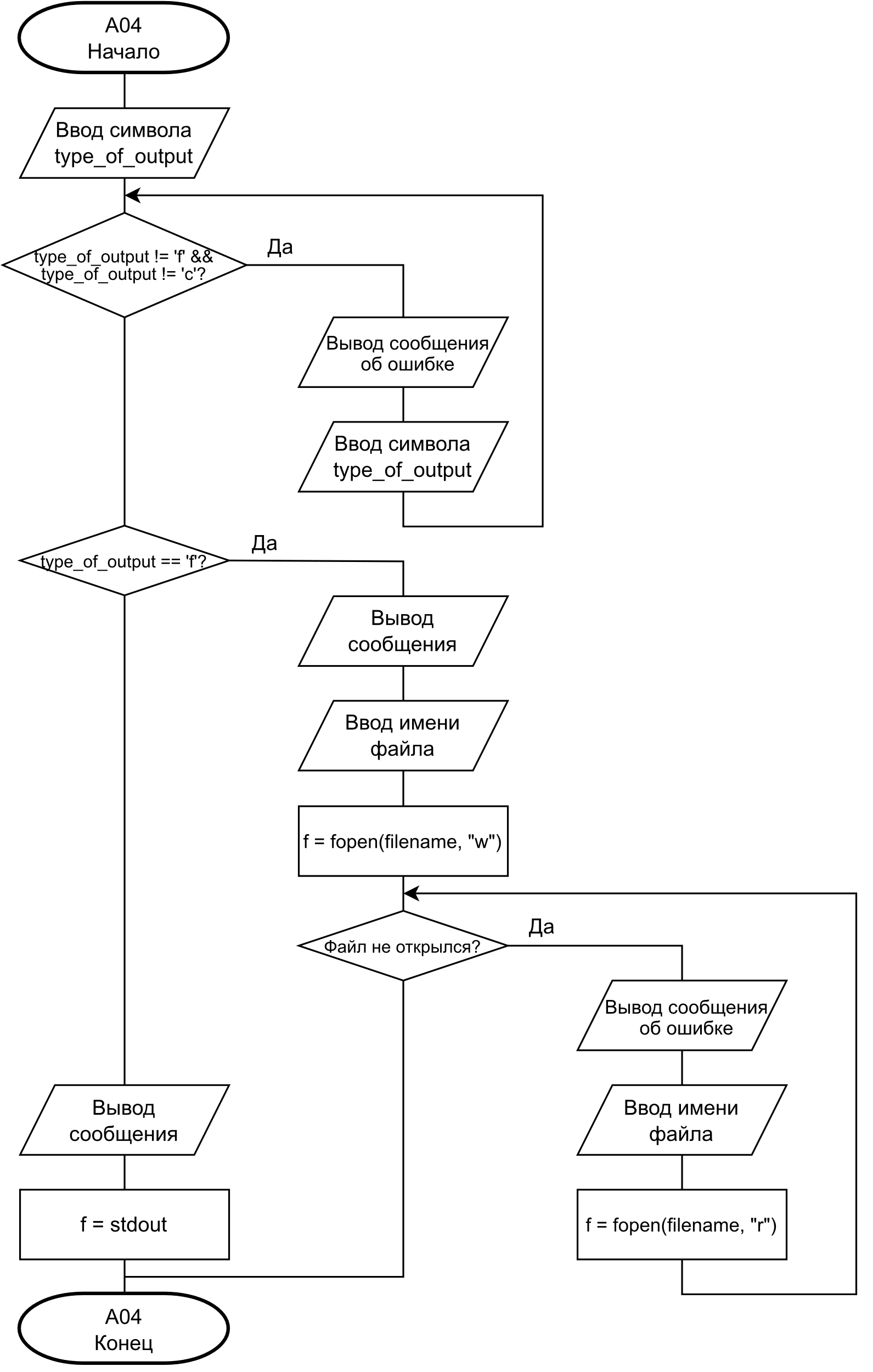


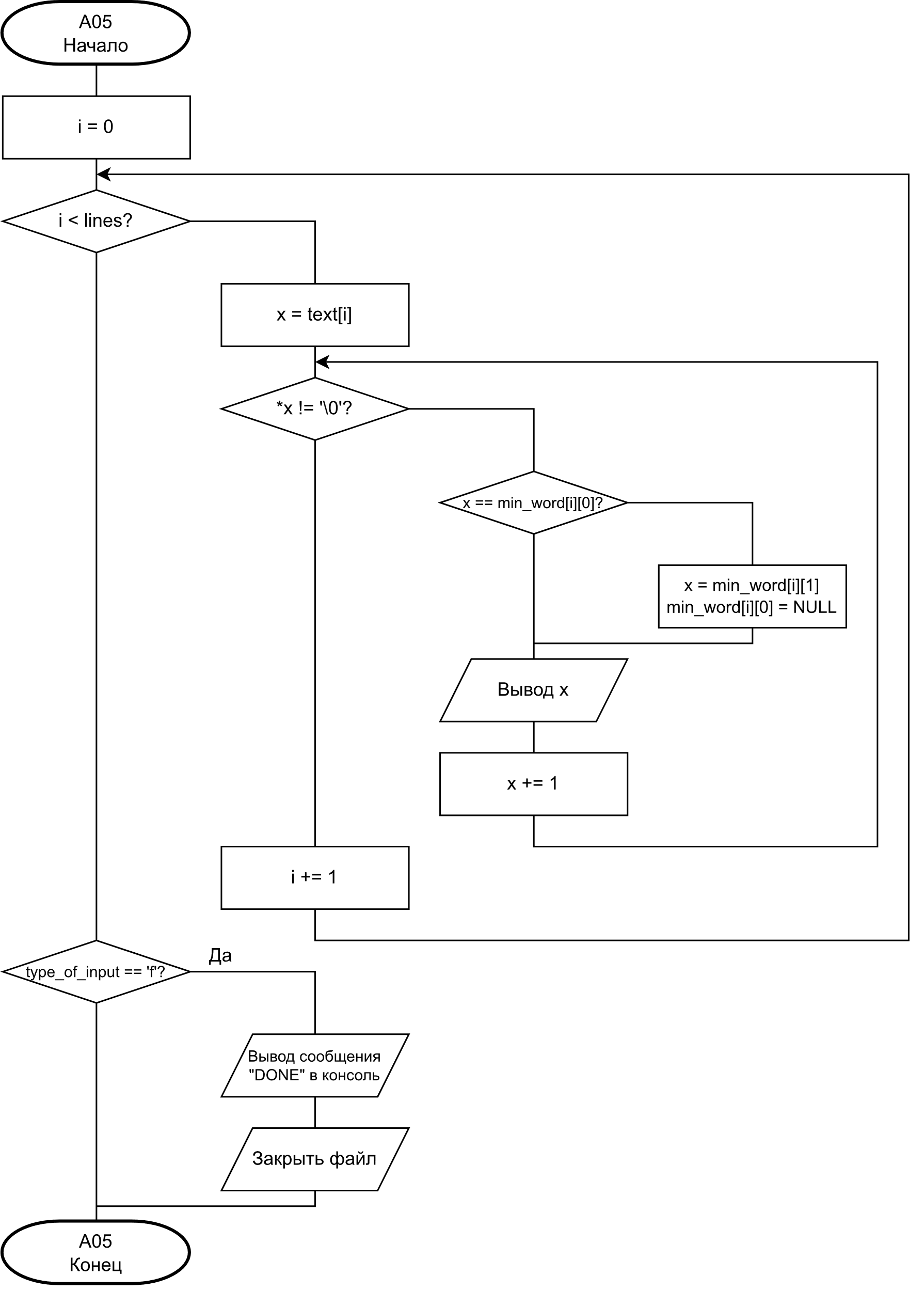












# Текст программы

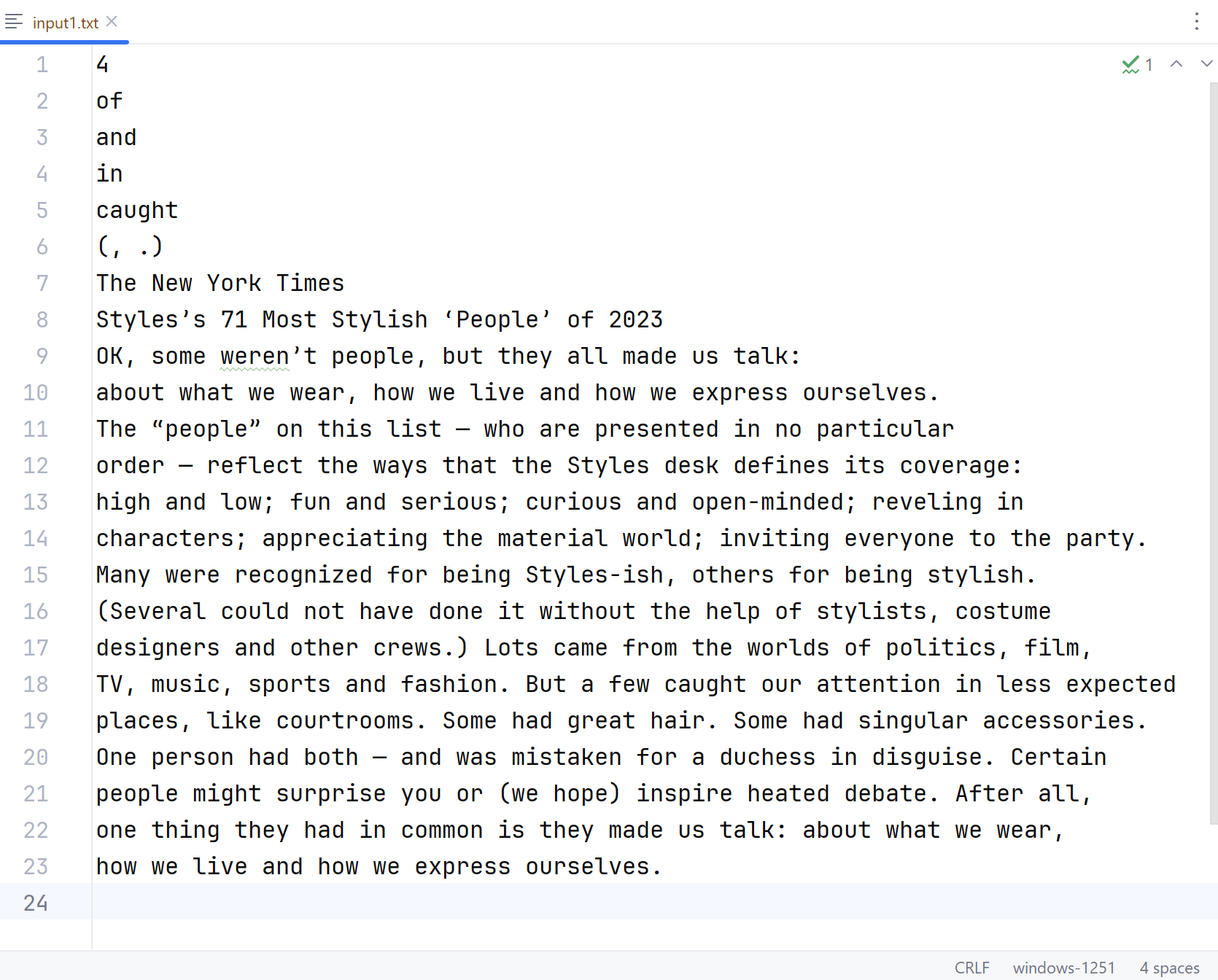
|  |
| --- |
| **#include <stdio.h>  #define HASH\_MOD 1000000009 */\* Hash module \*/* #define HASH\_B1 43 */\* Hash base 1 \*/* #define HASH\_B2 47 */\* Hash base 2 \*/* #define MAXLEN\_N 128 */\* Max file name length \*/* #define MAXLEN\_K 128 */\* Max keyword length \*/* #define MAXKEYS 256 */\* Max keywords count \*/* #define MAXLEN\_D 64 */\* Max delimiters length \*/* #define MAXLEN\_S 512 */\* Max text lines length \*/* #define MAXLINES 512 */\* Max lines count \*/  /\* The function checks if there is "c" in the array delimiters \*/* int is\_delimiter(char c, char \*delimiters);  int main() {  */\* Block of variables \*/*  int r; */\* Module of hash \*/*  int t; */\* First hash base \*/*  int b; */\* Second hash base \*/*  char type\_of\_input; */\* Character variable for input type ('f' or 'c') \*/*  char filename[MAXLEN\_N]; */\* String of filename \*/*  FILE \*f; */\* File pointer \*/*  int cnt\_of\_keywords; */\* Count of keywords \*/*  char keyword[MAXLEN\_K]; */\* String of keyword \*/*  char \*x; */\* Character pointer \*/*  long long hash\_of\_word1; */\* Long integer for first hash value \*/*  long long hash\_of\_word2; */\* Long integer for second hash value \*/*  int keysh[MAXKEYS][3]; */\* 2D array for storing hash values and counts \*/*  char delimiters[MAXLEN\_D]; */\* Array to store delimiters \*/*  int flag; */\* Flag for control input \*/*  int lines; */\* Count of lines \*/*  char text[MAXLINES][MAXLEN\_S]; */\* 2D array to store text \*/*  char \*start; */\* Character pointer for the start of a word \*/*  int min\_line; */\* Minimum word length in line \*/*  int cnt\_words\_bef; */\* Count of founded words before \*/*  int cnt\_words\_now; */\* Count of founded words now \*/*  char \*min\_word[MAXLINES][2]; */\* 2D array to store pointers to the beginning and end of the minimum word in each line \*/*  char type\_of\_output; */\* Character variable for output type ('f' or 'c') \*/***  ***/\* Block of initialization \*/* r = HASH\_MOD;  t = HASH\_B1;  b = HASH\_B2;  flag = 1;  lines = 0;  cnt\_words\_bef = 0;   */\* Info \*/* printf("Hello! The program removes the minimum words in the lines that contain keywords.\n");   */\* Block of input \*/  /\* Prompt the user for input type (file or console) \*/* printf("Input from file or console? (f/c)\n");  type\_of\_input = getchar();   */\* Validate the input type; loop until a valid input is provided \*/* while (type\_of\_input != 'f' && type\_of\_input != 'c') {  printf("Something went wrong! Please enter \'f\' or \'c\':\n");  type\_of\_input = getchar();  }   */\* Process based on the input type \*/* if (type\_of\_input == 'f') {  */\* Prompt user for the file name and attempt to open the file \*/* printf("Please enter the file name (limit: %i chars):\n",  MAXLEN\_N);  scanf("%s", filename);  f = fopen(filename, "r");   */\* Continue prompting until a valid file is opened \*/* while (f == NULL) {  printf("Something went wrong! Perhaps such a file does not exist."  "\nPlease enter the file name again:\n");  scanf("%s", filename);  f = fopen(filename, "r");  }  } else {  */\* If console input, set the file pointer to standard input \*/* f = stdin;  }   */\* For console input, prompt for the count of keywords \*/* if (type\_of\_input == 'c')  printf("Please enter the count of keywords (limit: %i):\n",  MAXKEYS);  fscanf(f, "%i", &cnt\_of\_keywords);   */\* For console input, prompt for each keyword and compute hash values \*/* if (type\_of\_input == 'c')  printf("Please enter the keywords (limit: %i chars):\n", MAXLEN\_K);  for (int i = 0; i < cnt\_of\_keywords; ++i) {  fscanf(f, "%s", keyword);  x = keyword;  hash\_of\_word1 = 0;  hash\_of\_word2 = 0;   */\* Compute double hash values for each character in the keyword \*/* while (\*x != '\0') {  hash\_of\_word1 = (hash\_of\_word1 \* t + (int) \*x) % r;  hash\_of\_word2 = (hash\_of\_word2 \* b + (int) \*x) % r;  ++x;  }   */\* Store the double hash values and initialize count in the keysh array \*/* keysh[i][0] = hash\_of\_word1;  keysh[i][1] = hash\_of\_word2;  keysh[i][2] = 0;  }   */\* For console input, prompt for the line of delimiter characters \*/* if (type\_of\_input == 'c')  printf("Please enter the line of delimiter characters (limit: %i chars):\n",  MAXLEN\_D);  fgets(delimiters, MAXLEN\_D, f);  fgets(delimiters, MAXLEN\_D, f);   */\* For console input, prompt for the lines of text with specified limits \*/* if (type\_of\_input == 'c')  printf("Please enter the lines of text (limit of line\'s length: %i; limit\n"  "count of lines: %i):\n", MAXLEN\_S, MAXLINES);    */\* Block of main logic \*/  /\* Loop through lines of text from the input file or console \*/* while (flag == 1 && fgets(text[lines], MAXLEN\_S, f)) {  x = text[lines];  start = text[lines];  hash\_of\_word1 = 0;  hash\_of\_word2 = 0;  min\_line = MAXLEN\_S;   */\* Iterate through characters in the current line \*/* while (\*x != '\0') {  */\* Check if the character is a delimiter \*/* if (is\_delimiter(\*x, delimiters) == 1) {  */\* Process the word if hash values are not zero \*/* if (hash\_of\_word1 != 0 || hash\_of\_word2 != 0) {  */\* Check if the word matches any keyword and update counts \*/* for (int i = 0; i < cnt\_of\_keywords; ++i) {  if (hash\_of\_word1 == keysh[i][0] &&  hash\_of\_word2 == keysh[i][1]) {  ++keysh[i][2];  }  }  */\* Reset hash values for the next word \*/* hash\_of\_word1 = 0;  hash\_of\_word2 = 0;  }   */\* Update minimum word information if conditions are met \*/* if (x - start != 0 && min\_line > x - start) {  min\_line = x - start;  min\_word[lines][0] = start;  min\_word[lines][1] = x;  }  start = x +  1; */\* Move start pointer to the next character \*/* } else {  */\* Update hash values for the current word \*/* hash\_of\_word1 = (hash\_of\_word1 \* t + (int) \*x) % r;  hash\_of\_word2 = (hash\_of\_word2 \* b + (int) \*x) % r;  }  ++x;  }   */\* Update flag and word count information \*/* flag = 0;  cnt\_words\_now = 0;   */\* Check if any keyword count is still zero, set the flag accordingly \*/* for (int i = 0; i < cnt\_of\_keywords; ++i) {  if (keysh[i][2] == 0) flag = 1;  cnt\_words\_now += keysh[i][2];  }   */\* Update minimum word information if the current word count matches the previous one \*/* if (cnt\_words\_now == cnt\_words\_bef) {  min\_word[lines][0] = text[lines];  min\_word[lines][1] = text[lines];  }   cnt\_words\_bef = cnt\_words\_now; */\* Update the previous word count \*/* lines++; */\* Move to the next line in the text \*/* }   */\* Close the file if input is from a file \*/* if (type\_of\_input == 'f') fclose(f);    */\* Block of output \*/  /\* Prompt the user for output type (file or console) \*/* printf("Output to file or console? (f/c)\n");   */\* If the input was from a file, consume an extra newline character from the buffer \*/* if (type\_of\_input == 'f') getchar();   */\* Get the user's choice for output type \*/* type\_of\_output = getchar();   */\* Validate the output type; loop until a valid input is provided \*/* while (type\_of\_output != 'f' && type\_of\_output != 'c') {  printf("Something went wrong! Please enter \'f\' or \'c\':\n");  type\_of\_output = getchar();  }   */\* Process based on the output type \*/* if (type\_of\_output == 'f') {  */\* Prompt user for the file name and attempt to open the file for writing \*/* printf("Please enter the file name (limit: %i chars):\n",  MAXLEN\_N);  scanf("%s", filename);  f = fopen(filename, "w");   */\* Continue prompting until a valid file is opened \*/* while (f == NULL) {  printf("Something went wrong! Please enter the file name again:\n");  scanf("%s", filename);  f = fopen(filename, "w");  }  } else {  */\* If console output, set the file pointer to standard output \*/* printf("Processed text:\n");  f = stdout;  }   */\* Process and output each line of text, excluding the minimum words \*/* for (int i = 0; i < lines; ++i) {  x = text[i];  while (\*x != '\0') {  */\* Skip characters within the minimum words and update the pointer \*/* if (x == min\_word[i][0]) {  x = min\_word[i][1];  min\_word[i][0] = NULL;  }  */\* Output the character to the file or console \*/* fprintf(f, "%c", \*x);  ++x;  }  }   */\* If the output type is a file, display a completion message, close the file, and finalize \*/* if (type\_of\_output == 'f') {  printf("DONE");  fclose(f);  }   return 0; }  int is\_delimiter(char c, char \*delimiters) {  char \*x;  int ans;  x = delimiters;  ans = 0;  while (\*x != '\0') { */\* Iterating until the end of delimiters \*/* if (c == \*x) ans = 1;  ++x;  }  return ans; */\* Return 1 if the element in the array 0 else \*/* }** |

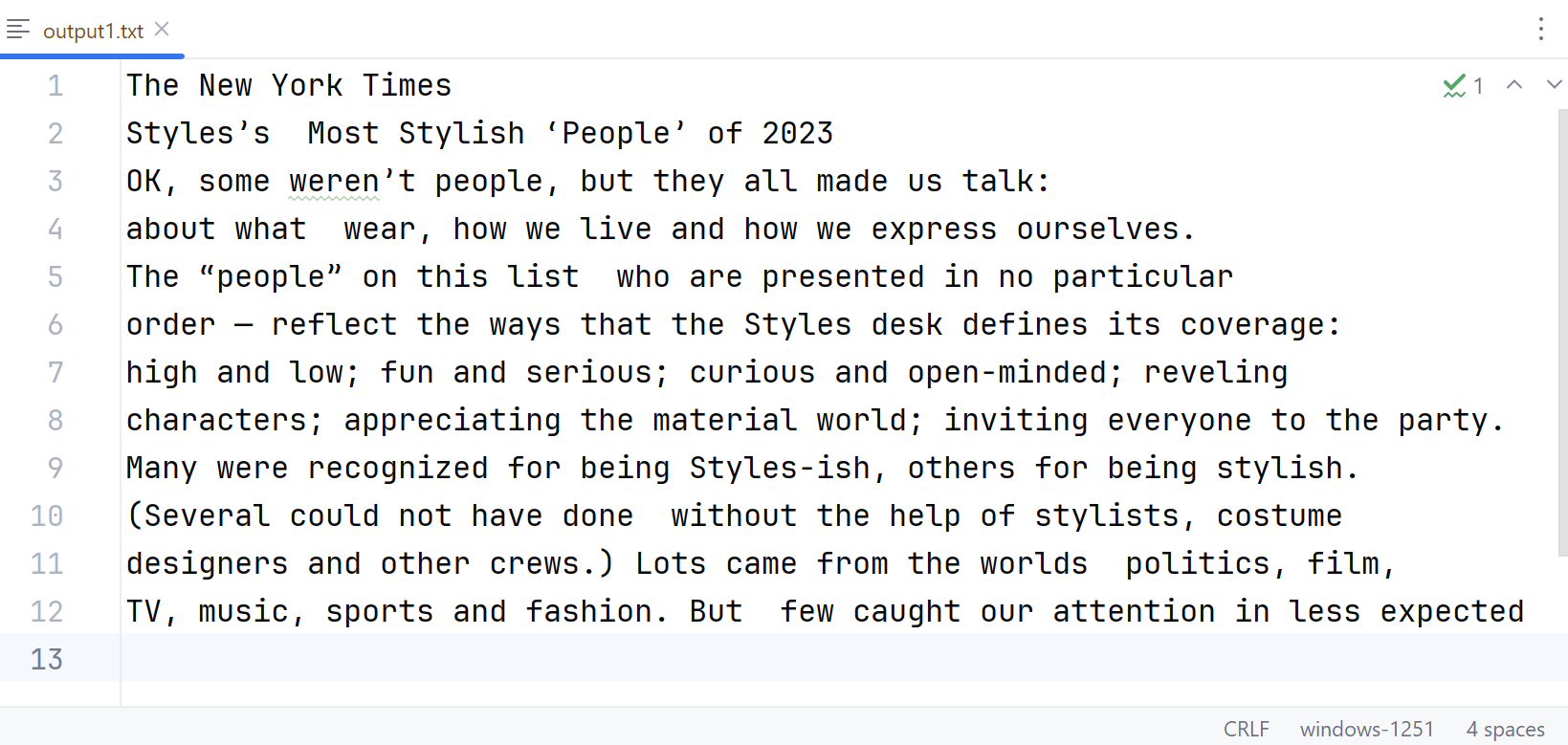
# Контрольные примеры

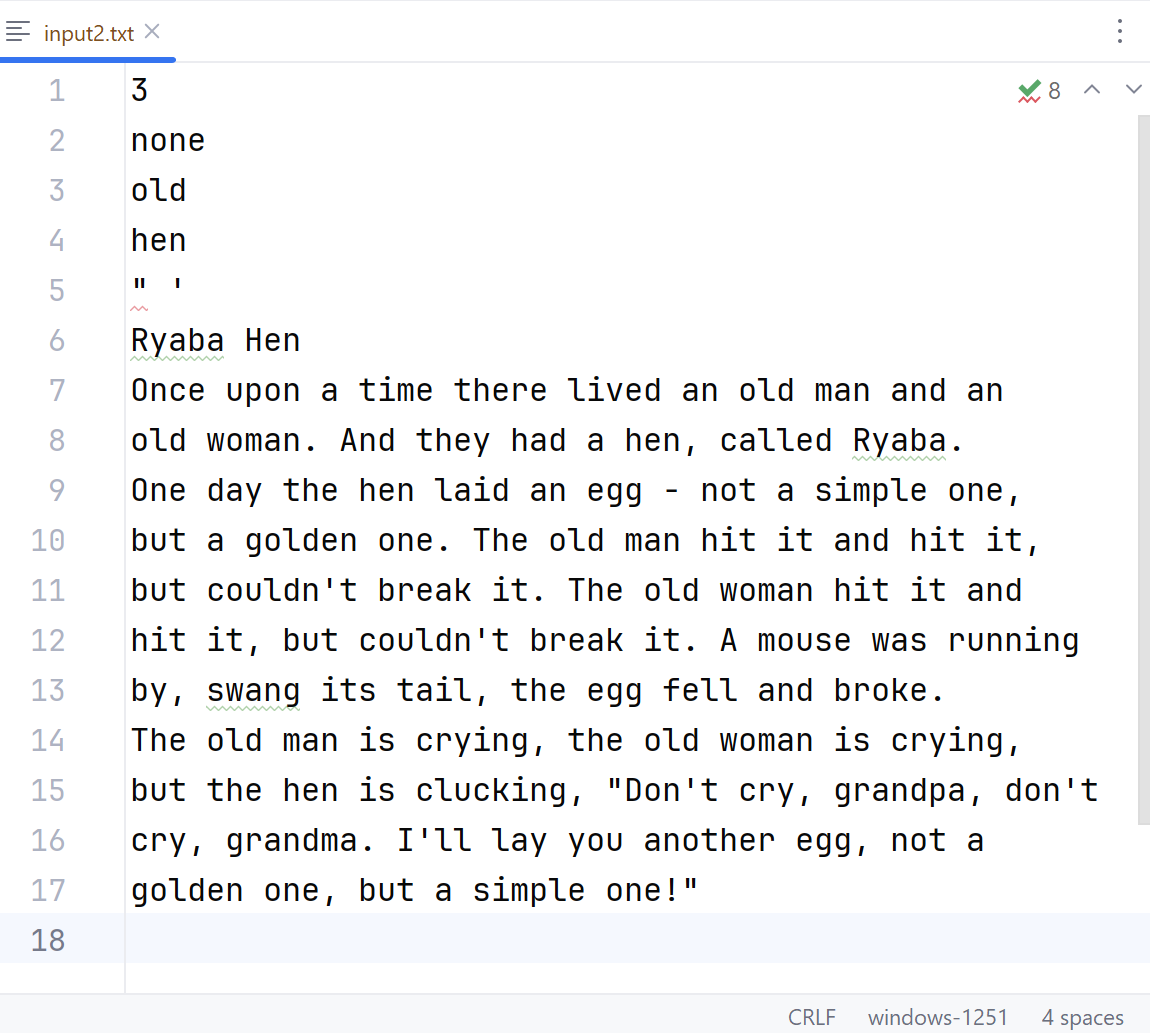
|  |  |  |
| --- | --- | --- |
| № | Исходные данные | Результаты |
| 1 | f  input1.txt  f  output1.txt | DONE |
| 2 | f  input2.txt  c | Processed text:  Ryaba Hen  Once upon time there lived an old man and an  old woman. And they had hen, called Ryaba.  One day the hen laid an egg not a simple one,  but golden one. The old man hit it and hit it,  but couldn' break it. The old woman hit it and  hit it, but couldn't break it. A mouse was running  by, swang its tail, the egg fell and broke.  The old man crying, the old woman is crying,  but the hen is clucking, "Don' cry, grandpa, don't  cry, grandma. I'll lay you another egg, not a  golden one, but a simple one!" |

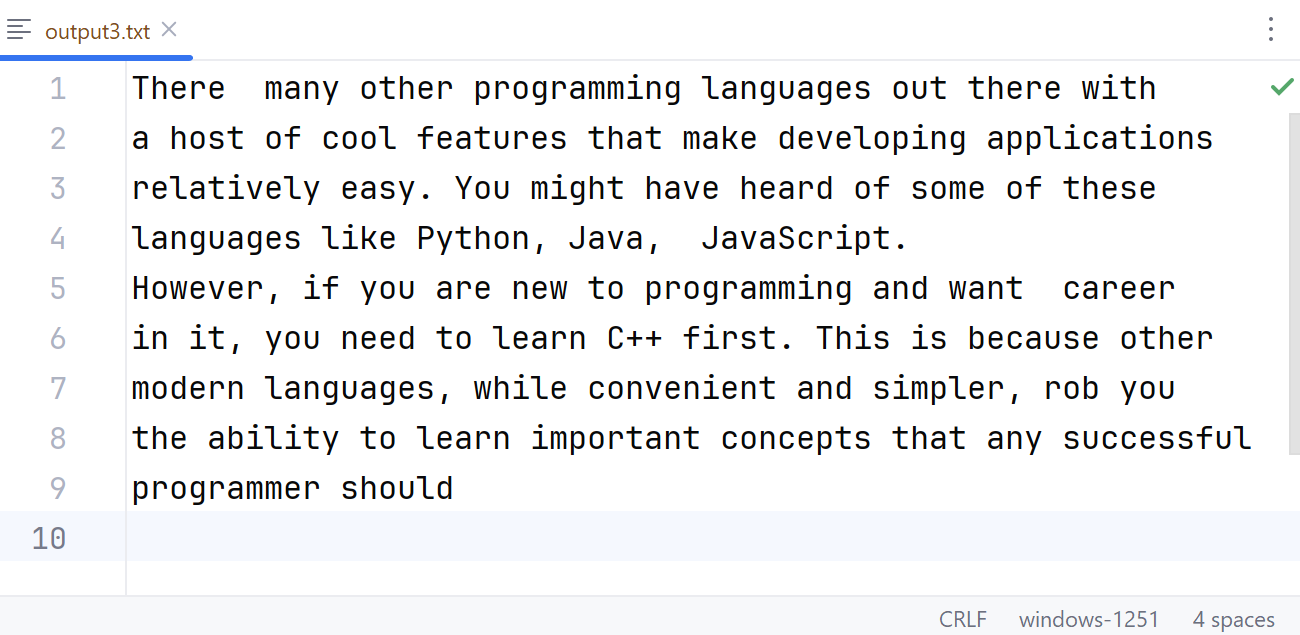
|  |  |  |
| --- | --- | --- |
| 3 | c  4  are  simpler,  programmer  Java,    There are many other programming languages out there with  a host of cool features that make developing applications  relatively easy. You might have heard of some of these  languages like Python, Java, and JavaScript.  However, if you are new to programming and want a career  in it, you need to learn C++ first. This is because other  modern languages, while convenient and simpler, rob you of  the ability to learn important concepts that any successful  programmer should know.  f  output3.txt | DONE |

# Содержимое файлов

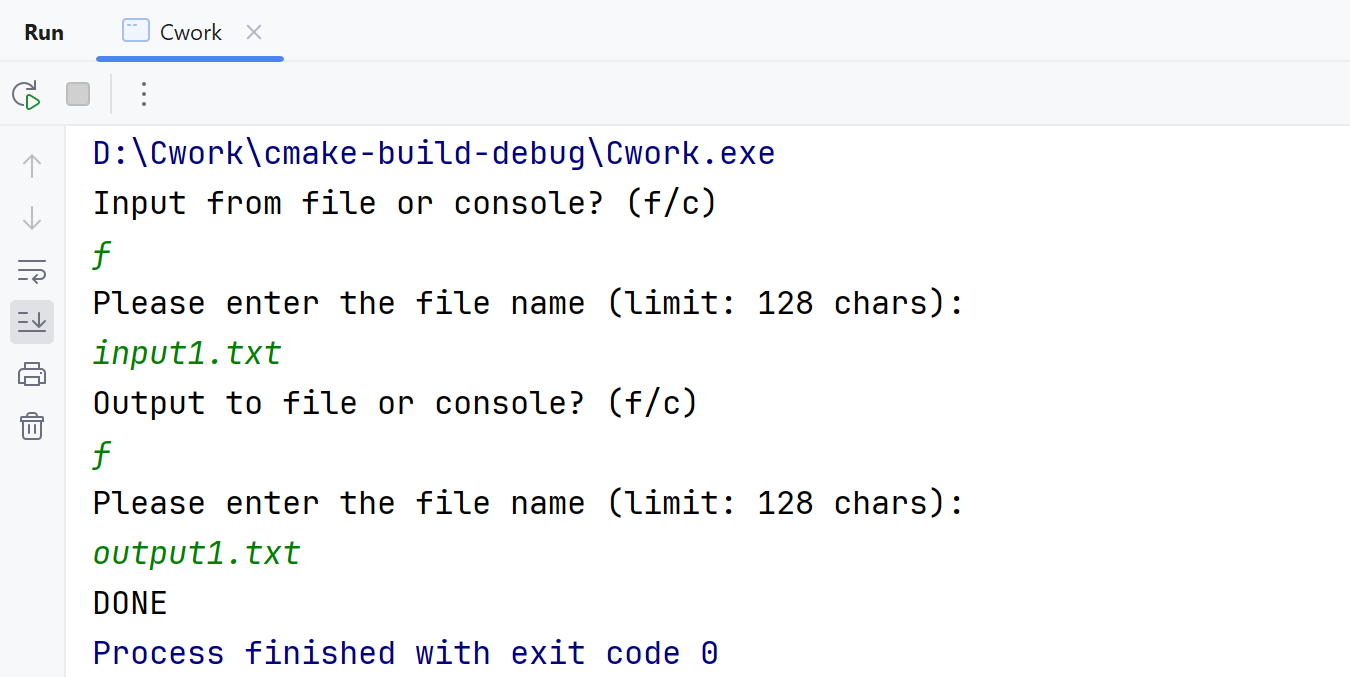


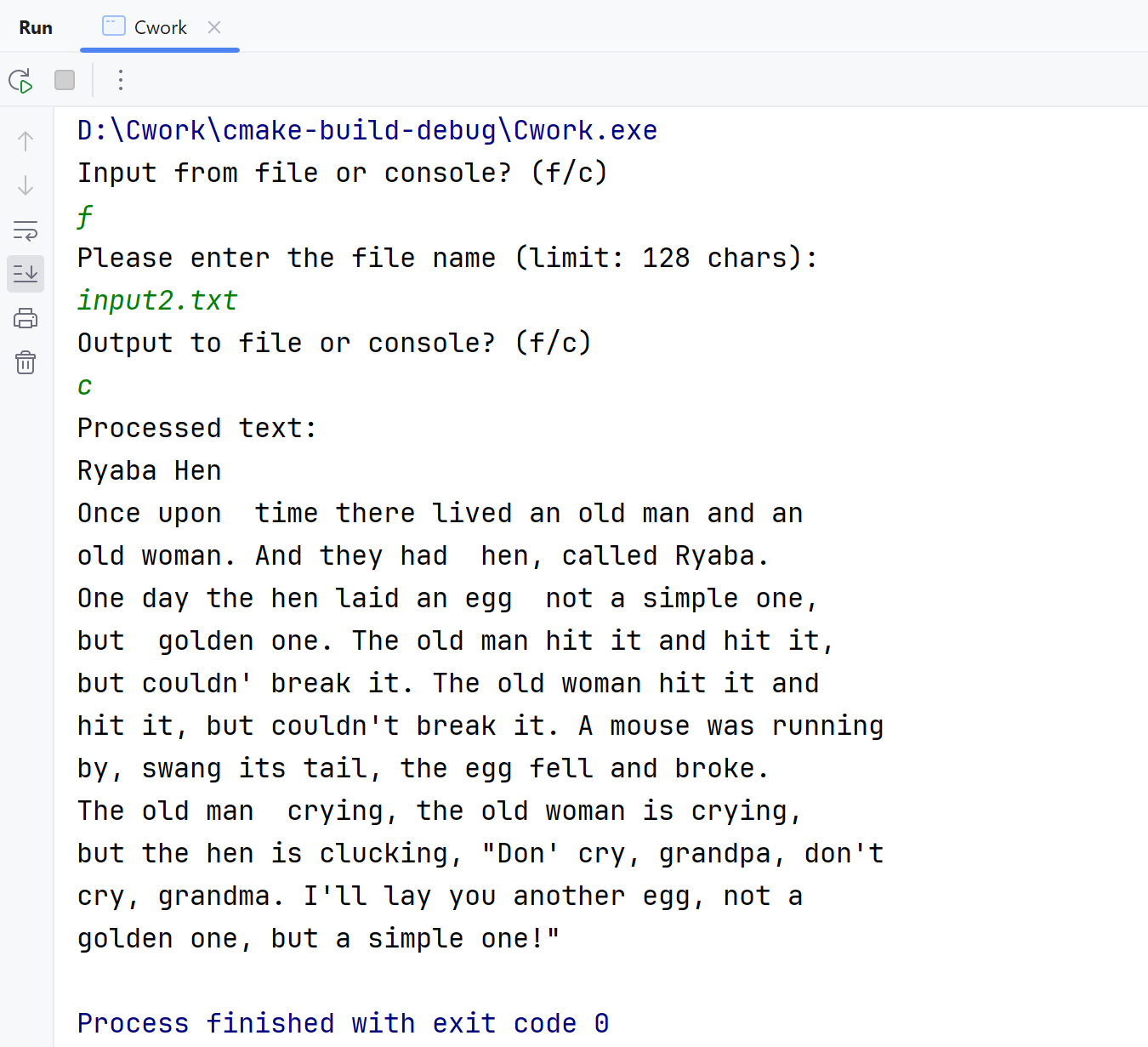


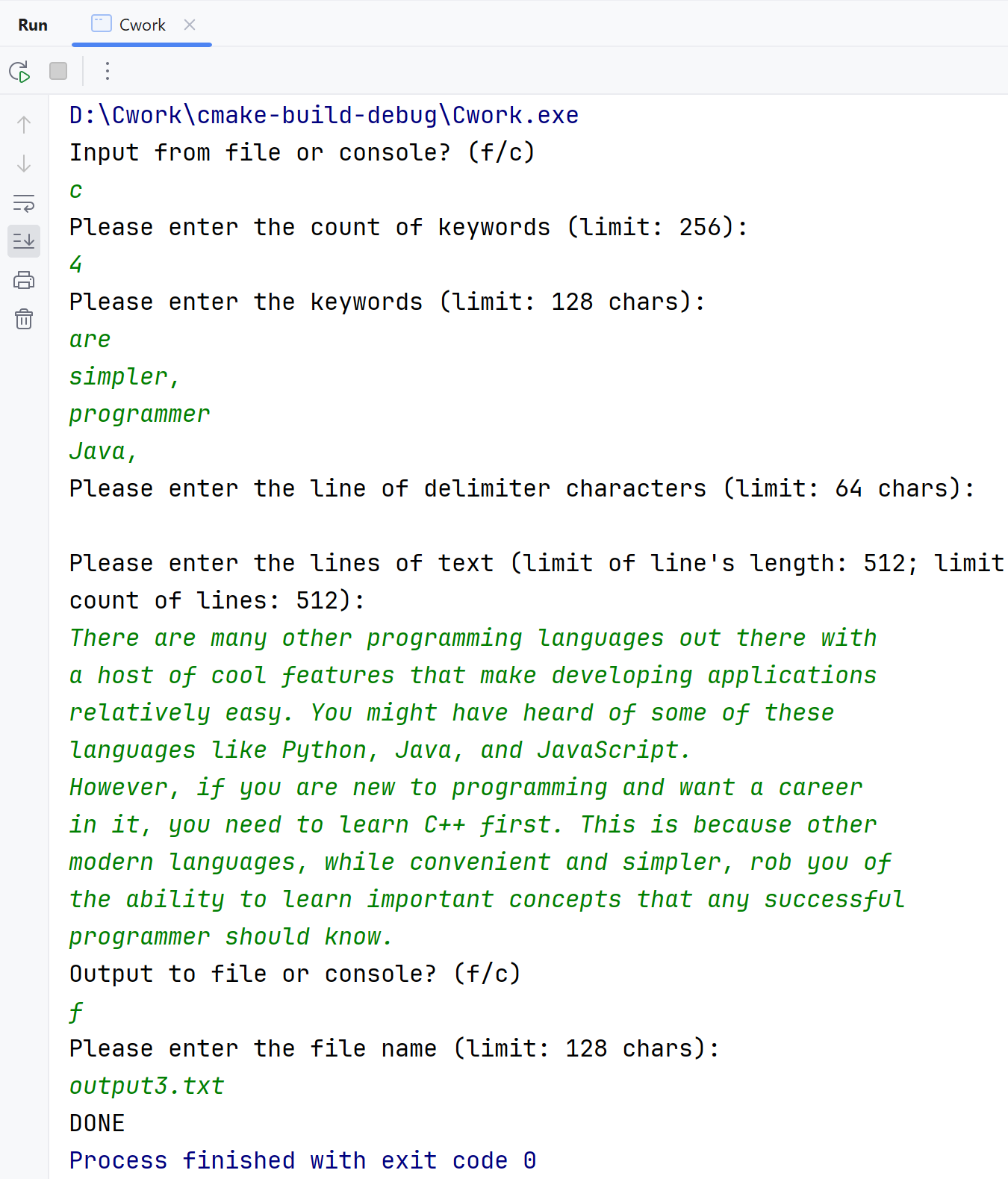




# Примеры выполнения программы







# Выводы

В работе использован только один заголовочный файл стандартной библиотеки. *<stdio.h>* используется для ввода и вывода из файла и консоли.