

Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Факультет безопасности информационных технологий

Дисциплина:
«Алгоритмы и структуры данных»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 1
«Сортировка Бабушкина»

Выполнил:

Сыралёв И. А., студент группы N3246

(подпись)

Проверил:

Ерофеев С.А.

(отметка о выполнении)

(подпись)

Санкт-Петербург
2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	3
1 ХОД РАБОТЫ	4
1.1 Описание алгоритма.....	4
1.2 Псевдокод.....	5
1.3 Блок-схемы	11
1.4 Спецификация переменных	14
1.5 Код программы	20
1.6 Тесты.....	25
ЗАКЛЮЧЕНИЕ	28

ВВЕДЕНИЕ

Цель:

Разработать программу сортировки Бабушкина для массива чисел из файла.

1 ХОД РАБОТЫ

1.1 Описание алгоритма

Алгоритм сортировки Бабушкина — псевдоалгоритм сортировки, разработанный Алексеем Бабушкиным (Автор FakeAV антивируса "Иммунитет"). Сортировка происходит в несколько шагов:

1. Создаётся копия исходного массива и сортируется любым рабочим способом.
2. Создаётся пустой массив индексов. Проходя по исходному массиву, вставляем индекс элемента в сортированном массиве, совпадающего с текущим элементом исходного массива. Заполненный массив индексов представляет собой число Бабушкина в N-ричной системе счисления. Каждый разряд этого числа определяет позицию соответствующего числа исходно массива в отсортированном массиве.
3. N-ричное число переводят в десятичную систему, получая таким образом число Бабушкина в десятичной форме. Это число делят на 10^{k+1} , где k - количество разрядов в десятичном числе.
4. Для полученной дроби ищут два взаимно простых числа, дающих при делении эту дробь. Один из простых способов - представить десятичную дробь обыкновенной и упростить. В таком случае числитель и знаменатель будут взаимно простыми.
5. Эти два числа теперь можно передать в функцию сортировки для обратного преобразования сначала в десятичную дробь, затем в десятичное представление числа Бабушкина, затем в N-ричное, и наконец осуществить сортировку.

Пример: Дан массив длиной 10: 22 44 66 88 77 99 100 55 33 11

1. Сортируем массив: 11 22 33 44 55 66 77 88 99 100
2. Создаём массив индексов: 1 3 5 7 6 8 9 4 2 0
3. массив индексов — число в N-ричной системе. В нашем случае N=10. Перевод из десятичной в десятичную не нужен, мы уже имеем число Бабушкина.

4. Число Бабушкина: 1357689420

Составим и упростим дробь $\frac{1357689420}{10000000000} = \frac{67884471}{5000000000}$

5. Теперь числитель и знаменатель можно передать в функцию сортировки, совершить обратные преобразования и отсортировать массив.

1.2 Псевдокод

структура Бабушкина

начало

 целое знаменатель

 целое числитель

 целое число_знаков_после_запятой

конец

начало

 открыть_файл для чтения

 если файл не существует то

 вывести("Не удалось открыть файл.")

 выйти из программы

 конец если

 считать размер_массива из файла и записать в

 ↪ переменную размер

 вывести "Размер массива: ", размер

 создать пустой массив исходный_массив размером

 ↪ размер

 считать массив из файла

 закреть файл

структура Бабушкина bnums :=

- ↪ получить_число_Бабушкина(исходный_массив,
- ↪ размер)

создать пустой массив старый_массив размером

- ↪ размер

копировать массив исходный_массив в старый_массив

- ↪ размером размер

сортировать_с_помощью_числа_Бабушкина(исходный_ма_

- ↪ ссив, размер,
- ↪ bnums)

вывести_массивы_для_сравнения(старый_массив,

- ↪ исходный_массив, размер)

конец

начало получить_число_Бабушкина(входной_массив,

- ↪ размер)

создать массив массив_Бабушкина размером размер

создать переменную дробь_Бабушкина и установить её

- ↪ в 0

создать переменную число_Бабушкина и установить её

- ↪ в 0

создать структуру Бабушкина bnums и установить

- ↪ поля bnums.знаменатель, bnums.числитель и
- ↪ bnums.число_знаков_после_запятой в 0

создать массив отсортированный_массив размером

- ↪ размер

скопировать массив входной_массив в

- ↪ отсортированный_массив

сортировать_пузырьком отсортированный_массив

↪ размером размер

для каждого элемента i от 0 до размер-1 делать

для каждого элемента j от 0 до размер-1 делать

если отсортированный_массив[j] ==

↪ входной_массив[i] то

массив_Бабушкина[i] := j

// комментарий: невозможное – значит

↪ не встречающееся в

↪ массив_Бабушкина

отсортированный_массив[j] :=

↪ невозможное_значение

прервать цикл

конец если

конец цикла

конец цикла

вывести_массив массив_Бабушкина

для каждого элемента i от размер-1 до 0 делать

число_Бабушкина := массив_Бабушкина[i] *

↪ размер в степени размер-1- i

если число_Бабушкина переполнен то

вывести "Ошибка переполнения! Число

↪ Бабушкина слишком большое"

выйти из программы

конец если

конец цикла

вывести число_Бабушкина

дробь_Бабушкина := число_Бабушкина

пока дробь_Бабушкина > 1 делать

```
    дробь_Бабушкина := дробь_Бабушкина / 10
    bnums.число_знаков_после_запятой =
        ↪ bnums.число_знаков_после_запятой + 1
конец цикла
```

```
bnums.знаменатель := 10 в степени
    ↪ bnums.число_знаков_после_запятой
bnums.числитель := число_Бабушкина
```

```
если bnums.знаменатель переполнен то
    вывести "Ошибка переполнения! Знаменатель
        ↪ дроби Бабушкина слишком большой"
    выйти из программы
конец если
```

```
упростить_дробь(адрес bnums.числитель, адрес
    ↪ bnums.знаменатель)
вывести "Дробь Бабушкина: ", bnums.числитель, "/",
    ↪ bnums.знаменатель
вернуть bnums
конец получить_число_Бабушкина()
```

```
начало упростить_дробь(адрес числитель, адрес
    ↪ знаменатель)
    создать переменную делитель и установить её в
        ↪ наибольший_общий_делитель(числитель,
        ↪ знаменатель)
    разделить numerator на делитель и записать
        ↪ результат по адресу числитель
    разделить denominator на делитель и записать
        ↪ результат по адресу знаменатель
конец упростить_дробь()
```


начало наибольший_общий_делитель(число1, число2)
создать переменную буфер

пока число2 \neq 0 делать
буфер := число2
число2 := остаток от деления число1 на число2
число1 := буфер
конец цикла

вернуть число1
конец

начало сортировать_с_помощью_числа_Бабушкина(входной_массив, размер, bnums)

создать пустой массив выходной_массив размером размер

создать массив массив_Бабушкина размером размер

создать переменную буфер и установить её в 10 в

↪ степени $\text{bnums.число_знаков_после_запятой} * \text{bnums.числитель} / \text{bnums.знаменатель}$

↪ $\text{bnums.числитель} / \text{bnums.знаменатель}$

создать переменную число_Бабушкина и установить её

↪ в значение буфер

создать переменную i и установить её в 0

массив_Бабушкина[0] := 0

пока число_Бабушкина > 0 делать

массив_Бабушкина[размер - i - 1] := остаток от деления число_Бабушкина на размер

число_Бабушкина := число_Бабушкина / размер

i = i + 1

конец цикла

вывести "Массив Бабушкина (сортировка):"
вывести_массив массив_Бабушкина размером размер

для каждого элемента i от 0 до размер-1 делать
 выходной_массив[массив_Бабушкина[i]] :=
 ↪ входной_массив[i]
конец цикла

скопировать массив выходной_массив в
 ↪ входной_массив размером размер
конец

1.3 Блок-схемы

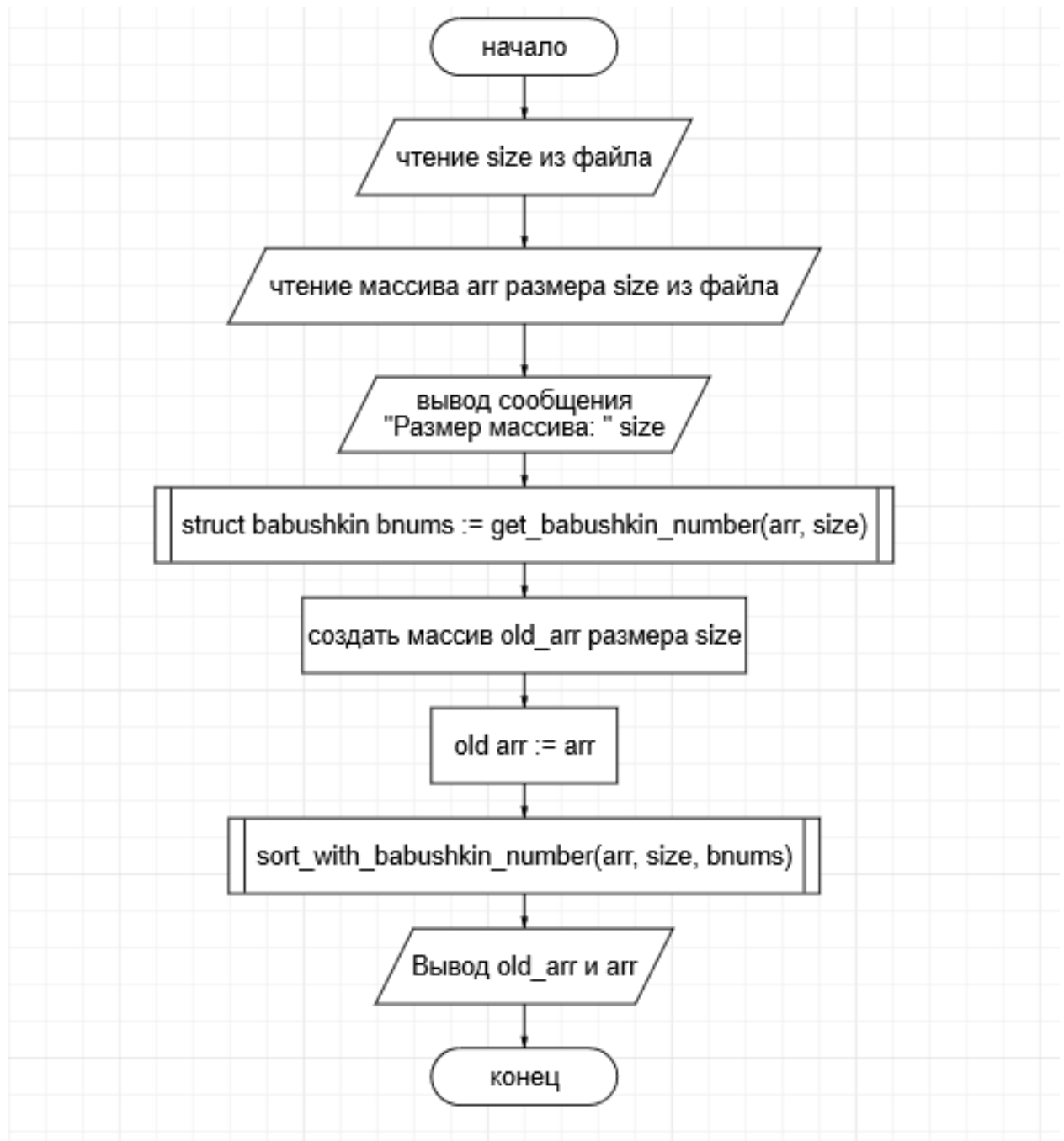


Рисунок 1 — Блок-схема главной функции

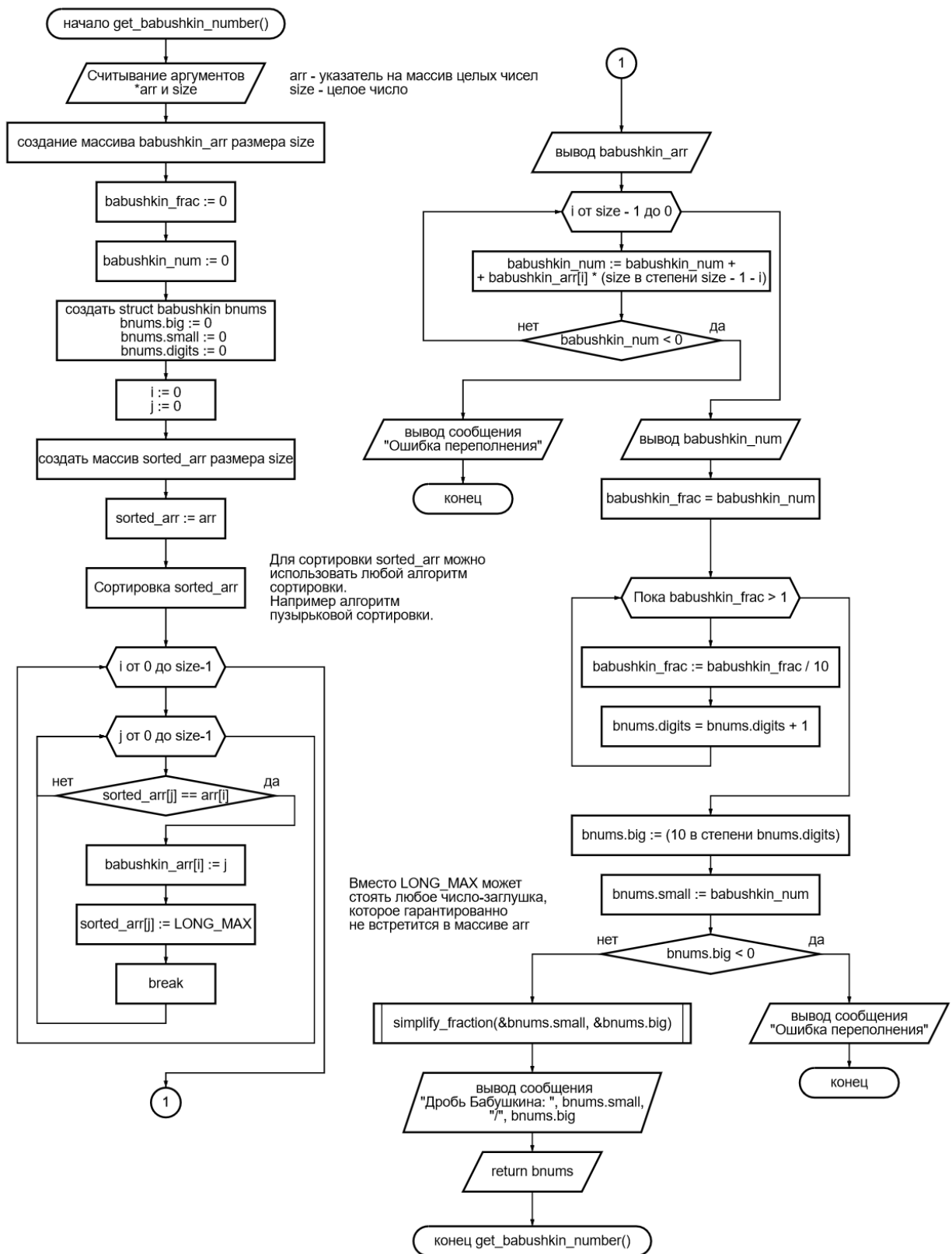


Рисунок 2 — Блок-схема функции для нахождения числа Бабушкина

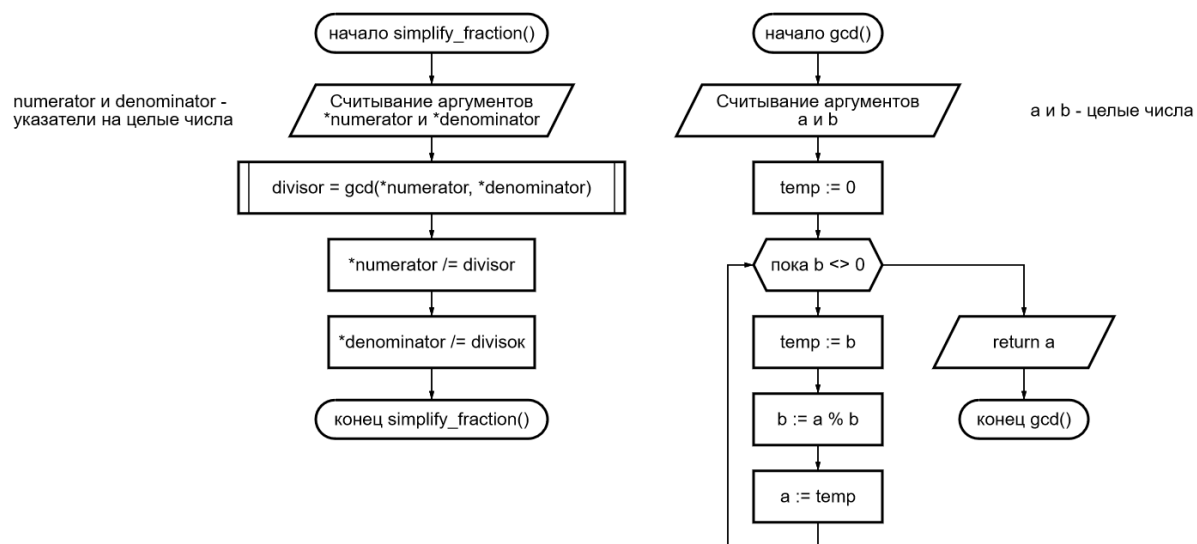


Рисунок 3 — Блок-схемы функций для упрощения обыкновенной дроби

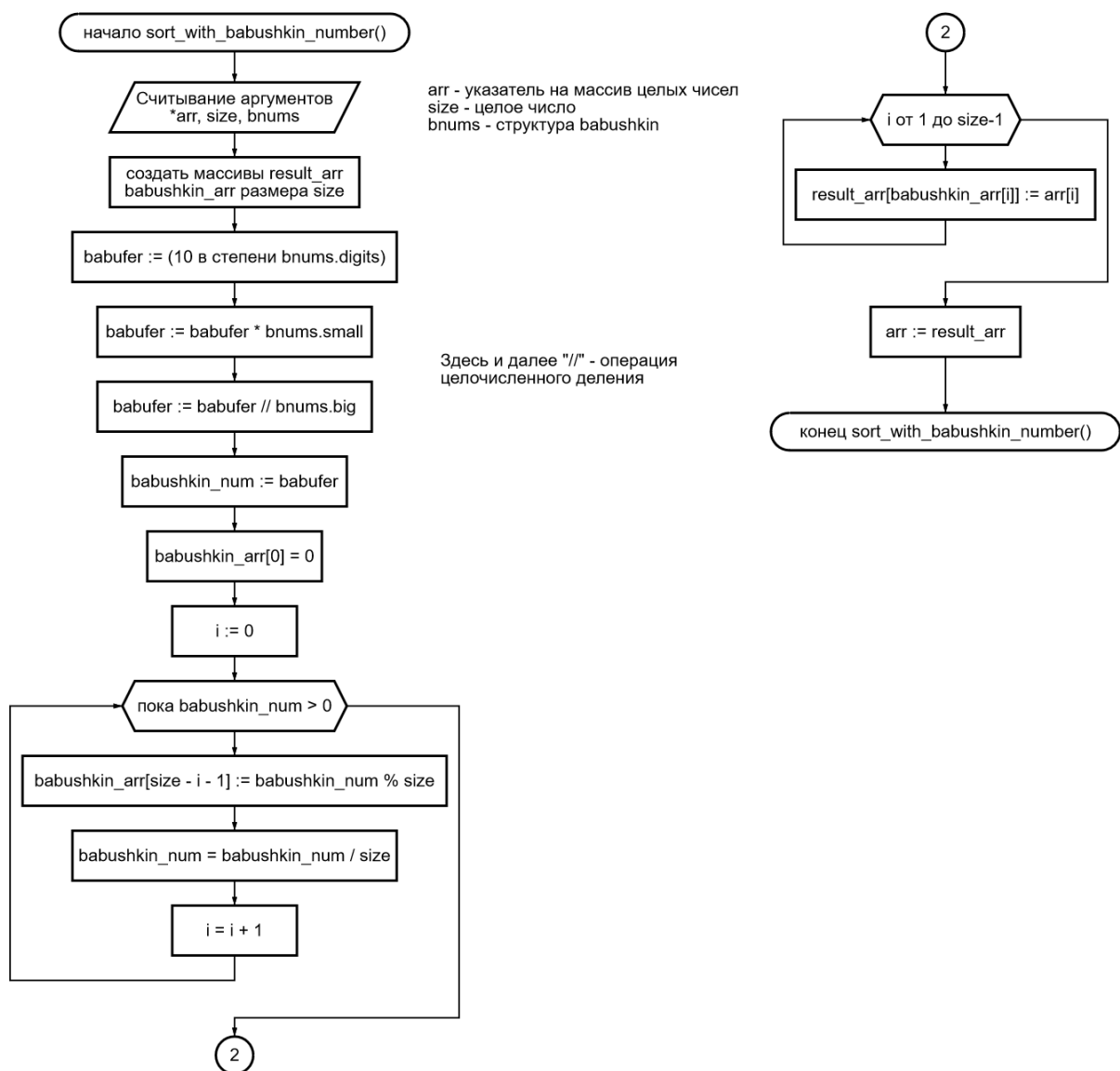


Рисунок 4 — Блок-схема функции сортировки Бабушкина

1.4 Спецификация переменных

Таблица 1 — Переменные структуры babushkin

Имя	Тип	Минимум	Значение
		Максимум	
big, small	long long	-9 223 372 036 854 775 808	числитель и знаменатель в дроби.
	int	9 223 327 036 854 775 807	
digits	long long	-9 223 372 036 854 775 808	количество разрядов в числе Бабушкина.
	int	9 223 327 036 854 775 807	

Таблица 2 — Переменные функции print_array

Имя	Тип	Минимум	Значение
		Максимум	
i	int	-2 147 483 648	итератор
		2 147 483 647	
length	int	-2 147 483 648	длина входного массива
		2 147 483 647	
arr	int*	0	указатель на входной массив
		1.8446744e+19	

Таблица 3 — Переменные функции gcd

Имя	Тип	Минимум	Значение
		Максимум	
a, b	long long int	-9 223 372 036 854 775 808	числа, для которых ищут НОД
		9 223 327 036 854 775 807	
temp	long long int	-9 223 372 036 854 775 808	буфер для обмена значений
		9 223 327 036 854 775 807	

Таблица 4 — Переменные функции simplify_fraction

Имя	Тип	Минимум	Значение
		Максимум	
divisor	long long int	-9 223 372 036 854 775 808	наибольший общий делитель
		9 223 327 036 854 775 807	
numerator	long long int*	0	указатель на числитель дроби
		1.8446744e+19	
denominator	long long int*	0	указатель на знаменатель дроби
		1.8446744e+19	

Таблица 5 — Переменные функции bubble_sort

Имя	Тип	Минимум	Значение
		Максимум	
size	int	-2 147 483 648	длина входного массива
		2 147 483 647	
arr	int*	0	указатель на входной массив
		1.8446744e+19	
i, j	int	-2 147 483 648	итераторы
		2 147 483 647	
temp	int	-2 147 483 648	буфер для обмена значений
		2 147 483 647	

Таблица 6 — Переменные функции сору_array

Имя	Тип	Минимум	Значение
		Максимум	
arr1	int*	0	указатель на массив, откуда копировать
		1.8446744e+19	
arr2	int*	0	указатель на массив, куда копировать
		1.8446744e+19	
size	int	-2 147 483 648	длина входных массивов
		2 147 483 647	

Таблица 7 — Переменные функции get_babushkin_number

Имя	Тип	Минимум	Значение
		Максимум	
arr	int*	0	указатель на входной массив
		1.8446744e+19	
size	int	-2 147 483 648	длина входного массива
		2 147 483 647	
babushkin_arr	int*	0	n-ричное число Бабушкина
		1.8446744e+19	
babushkin_frac	double	-2 147 483 648	дробное число Бабушкина
		2 147 483 647	
babushkin_num	long long int	-9 223 372 036 854 775 808	число Бабушкина
		9 223 327 036 854 775 807	
bnums	babushkin	-	данные для передачи числа Бабушкина
		-	
i, j	int	-2 147 483 648	итераторы
		2 147 483 647	
sorted_arr	int*	0	указатель на отсорт. массив
		1.8446744e+19	

Таблица 8 — Переменные функции sort_with_babushkin_number

Имя	Тип	Минимум	Значение
		Максимум	
arr	int*	0	указатель на входной массив
		1.8446744e+19	
size	int	-2 147 483 648	длина входного массива
		2 147 483 647	
babushkin_arr	int*	0	n-ричное число Бабушкина
		1.8446744e+19	
babufer	double	-2 147 483 648	дробное число Бабушкина
		2 147 483 647	
babushkin_num	long long int	-9 223 372 036 854 775 808	число Бабушкина
		9 223 327 036 854 775 807	
bnums	babushkin	-	данные для передачи числа Бабушкина
		-	
i	int	-2 147 483 648	итератор
		2 147 483 647	
result_arr	long int*	0	указатель на отсорт. массив
		1.8446744e+19	

Таблица 9 — Переменные функции print_array

Имя	Тип	Минимум	Значение
		Максимум	
arr1	int*	0	указатель на старый массив
		1.8446744e+19	
arr2	int*	0	указатель на новый массив
		1.8446744e+19	
length	int	-2 147 483 648	длина входных массивов
		2 147 483 647	
max_length	int	-2 147 483 648	максимальная длина числа в массиве
		2 147 483 647	

Таблица 10 — Переменные функции main

Имя	Тип	Минимум	Значение
		Максимум	
size	int	-2 147 483 648	длина входного массива
		2 147 483 647	
i	int	-2 147 483 648	итератор
		2 147 483 647	
file	FILE*	0	указатель на входной файл
		1.8446744e+19	
old_array	int*	0	указатель на старый массив
		1.8446744e+19	
bnums	babushkin	-	данные для передачи числа Бабушкина
		-	

1.5 Код программы

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4
5
6  // Сортировка Бабушкина - псевдоалгоритм сортировки. Для реализации
7  // этого алгоритма необходимо использовать другой, действительно
8  // работающий алгоритм сортировки.
9
10 // Структура, содержащая в себе два числа. При делении меньшего на большее
    ↪ получается
11 // число Бабушкина
12 struct babushkin {
13     long long int big, small;
14     // Количество знаков после запятой
15     long long int digits;
16 };
17
18 // Функция для вывода массива
19 void print_array(int *arr, int length) {
20
21     // Печать элементов массивов с учетом фиксированной ширины чисел
22     for (int i = 0; i < length; i++) {
23         printf("%d ", arr[i]);
24     }
25     printf("\n");
26 }
27
28 // Функция для нахождения наибольшего общего делителя (НОД)
29 long long int gcd(long long int a, long long int b) {
30     long long int temp;
31     while (b != 0) {
32         temp = b;
33         b = a % b;
34         a = temp;
35     }
36     return a;
37 }
38
39 // Функция для упрощения дроби
40 void simplify_fraction(long long int *numerator, long long int
    ↪ *denominator) {
41     long long int divisor = gcd(*numerator, *denominator);
42     *numerator /= divisor;
43     *denominator /= divisor;
44 }
```

```

45
46 // Вспомогательная функция сортировки. Я использую сортировку пузырьком.
47 void bubble_sort(int *arr, const int size) {
48     int i, j, temp;
49     // делаем size проходов по массиву, в результате каждого
50     //получаем 1 отсортированный элемент
51     for (i = 0; i < size-1; i++)
52         // Перемещаем элемент на его конечную позицию.
53         // Не проверяем уже отсортированные элементы.
54         for (j = 0; j < size - i - 1; j++) {
55             // Если текущий элемент меньше предыдущего, меняем их местами.
56             if (arr[j] > arr[j+1]){
57                 temp = arr[j];
58                 arr[j] = arr[j+1];
59                 arr[j+1] = temp;
60             }
61         }
62 }
63
64 // Функция для переноса значений из одного массива в другой
65 void copy_array(const int *arr1, int *arr2, const int size) {
66     // Поэлементно копируем элементы из одного массива в другой
67     for (int i = 0; i < size; i++)
68         arr2[i] = arr1[i];
69 }
70
71
72 // Функция для получения числа Бабушкина из массива
73 struct babushkin get_babushkin_number(int *arr, const int size) {
74     // n-ричное представление числа Бабушкина
75     int babushkin_arr[size];
76
77     // дробное представление числа Бабушкина
78     double babushkin_frac = 0;
79
80     // Само число Бабушкина
81     long long int babushkin_num = 0;
82
83     // Структура с двумя искомыми числами
84     struct babushkin bnums;
85     bnums.big = 0;
86     bnums.small = 0;
87     bnums.digits = 0;
88
89     // итераторы
90     int i, j;
91
92     // массив, в который будет записан отсортированный изначальный массив
93     long int sorted_arr[size];
94

```

```

95     copy_array(arr, (int*)sorted_arr, size);
96     bubble_sort((int*)sorted_arr, size);
97
98     // Теперь сравним все элементы в отсортированном и исходном массивах,
99     // чтобы выразить число Бабушкина в n-ичной форме
100    for (i = 0; i < size; i++)
101        for (j = 0; j < size; j++)
102            if (sorted_arr[j] == arr[i]) {
103                // Разряд i числа Бабушкина равен позиции j в отсортированном
104                // массиве.
105                babushkin_arr[i] = j;
106                // Чтобы не было дубликатов, меняем отсортированное значение
107                // на несуществующее значение
108                // Придумать действительно несуществующее значение
109                sorted_arr[j] = LONG_MAX;
110                break;
111            }
112    printf("Массив Бабушкина (вычисление):");
113    print_array(babushkin_arr, size);
114
115    // Перевод числа Бабушкина в десятичную систему счисления
116    for (i = size - 1; i > -1; i--) {
117        babushkin_num += babushkin_arr[i] * pow(size, size - 1 - i);
118        // Проверка на переполнение
119        if (babushkin_num < 0){
120            printf("Ошибка переполнения! Число Бабушкина слишком
121            ↪ большое\n");
122            exit(EXIT_FAILURE);
123        }
124    }
125    printf("Макс long long: %lld\n", LONG_LONG_MAX);
126    printf("Число Бабушкина: %lld\n", babushkin_num);
127    babushkin_frac = babushkin_num;
128
129    // Получаем дробь Бабушкина, попутно вычисляя количество знаков после
130    // запятой
131    while (babushkin_frac > 1) {
132        babushkin_frac /= 10;
133        bnums.digits += 1;
134    }
135
136    bnums.big = pow(10, bnums.digits);
137    bnums.small = babushkin_num;
138    // Проверка на переполнение
139    if (bnums.big < 0){
140        printf("Ошибка переполнения! Знаменатель дроби Бабушкина слишком
141        ↪ большой\n");
142        exit(EXIT_FAILURE);
143    }
144    simplify_fraction(&bnums.small, &bnums.big);

```

```

141     printf("Дробь Бабушкина: %lld/%lld\n", bnums.small, bnums.big);
142     return bnums;
143 }
144
145 // Функция для сортировки массива с использованием числа Бабушкина.
146 void sort_with_babushkin_number(int *arr, const int size, const struct
↪ babushkin bnums) {
147
148     // Создаем пустой массив для отсортированных элементов.
149     int result_arr[size];
150     // Массив с номерами позиций в сортированном массиве.
151     int babushkin_arr[size];
152     // Получаем число Бабушкина из частного двух чисел.
153     double babufer = pow(10, bnums.digits);
154     babufer *= (double)bnums.small;
155     babufer /= (double)bnums.big;
156     long long int babushkin_num = (long long int)babufer;
157     //Итератор
158     int i;
159
160
161     i = 0;
162     // Если число Бабушкина имеет ведущий ноль, то определим его здесь.
163     // Если нет - его перезапишет следующий цикл
164     babushkin_arr[0] = 0;
165     // Перевод числа бабушкина из десятичной в n-ричную систему счисления.
166     // Пока число больше нуля, продолжаем делить на основание системы
↪ счисления.
167     while (babushkin_num > 0) {
168         babushkin_arr[size - i - 1] = babushkin_num % size;
169         babushkin_num /= size;
170         i++;
171     }
172     printf("Массив Бабушкина (сортировка):");
173     print_array(babushkin_arr, size);
174     // Перебираем массив с числом бабушкина в n-ричной форме и перемещаем
175     // несортированные элементы на места, определяемые каждым разрядом этого
↪ числа.
176     // Получаем отсортированный массив.
177     for (i = 0; i < size; i++)
178         result_arr[babushkin_arr[i]] = arr[i];
179
180     copy_array(result_arr, arr, size);
181     return;
182 }
183
184 void print_arrays(int *arr1, int *arr2, int length) {
185     // Определение максимальной длины числа в массиве
186     int max_length = 0;
187     for (int i = 0; i < length; i++) {

```

```

188         int len = sprintf(NULL, 0, "%d", arr1[i]); // Получаем длину числа
189         if (len > max_length) {
190             max_length = len;
191         }
192     }
193
194     printf("\n*s\t\t*t*s\n\n", max_length, "До", max_length, "После");
195
196     // Печать элементов массивов с учетом фиксированной ширины чисел
197     for (int i = 0; i < length; i++) {
198         printf("%d\t\t\t*d\n", max_length, arr1[i], max_length, arr2[i]);
199     }
200 }
201
202
203 int main() {
204     // Длина массива
205     int size;
206     // итератор
207     int i;
208
209     FILE *file = fopen("D:\\Learning\\ALG\\ALG_lab_1\\source\\input.txt",
210         ↪ "r");
211     if (file == NULL) {
212         printf("Не удалось открыть файл.\n");
213         return 1;
214     }
215
216     // Считывание размера массива и зфайла
217     fscanf(file, "%d", &size);
218     printf("Размер массива: %d\n", size);
219     // Сортируемый массив
220     int arr[size];
221
222     // Чтение элементов массива из файла
223     i = 0;
224     while (fscanf(file, "%d", &arr[i]) != EOF) {
225         i++;
226     }
227
228     fclose(file);
229
230     // Вычисляем число Бабушкина
231     struct babushkin bnums = get_babushkin_number(arr, size);
232
233     int old_arr[size];
234     copy_array(arr, old_arr, size);
235
236     // Сортируем массив с использованием числа Бабушкина

```

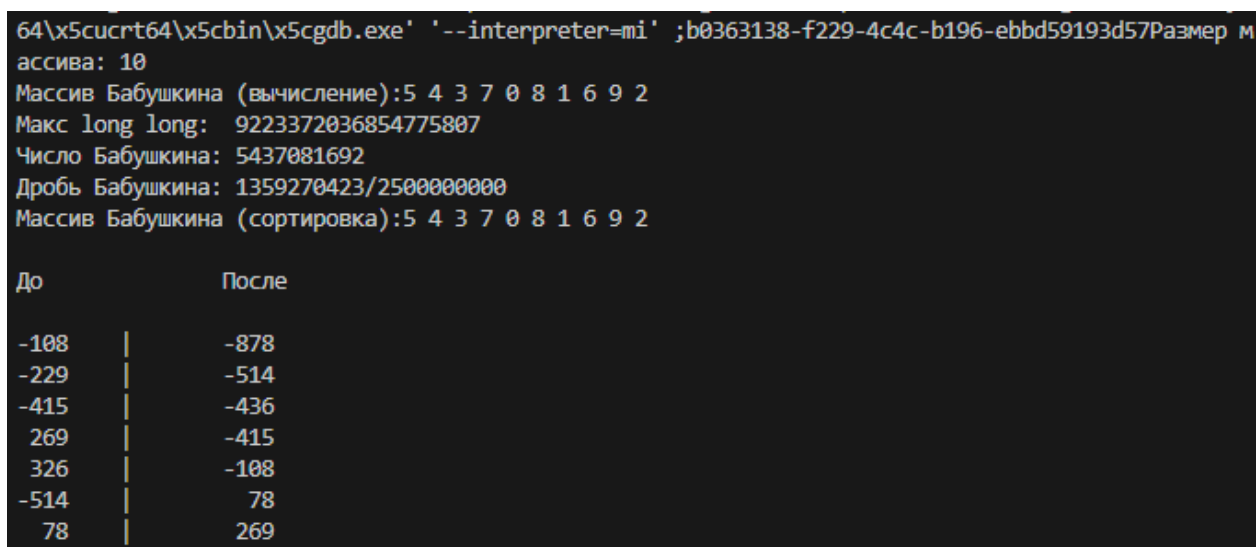


```

237 |     sort_with_babushkin_number(arr, size, bnums);
238 |
239 |     // Выводим исходный и отсортированный массив для сравнения
240 |     print_arrays(old_arr, arr, size);
241 |
242 |     return 0;
243 | }

```

1.6 Тесты



```

64\x5cucrt64\x5cbin\x5cgdb.exe' '--interpreter=mi' ;b0363138-f229-4c4c-b196-ebbd59193d57Размер м
ассива: 10
Массив Бабушкина (вычисление):5 4 3 7 0 8 1 6 9 2
Макс long long: 9223372036854775807
Число Бабушкина: 5437081692
Дробь Бабушкина: 1359270423/2500000000
Массив Бабушкина (сортировка):5 4 3 7 0 8 1 6 9 2

До          После
-108      |      -878
-229      |      -514
-415      |      -436
269       |      -415
326       |      -108
-514      |       78
78        |      269

```

Рисунок 5 — Корректная работа на массиве длиной 10

```
--interpreter=mi' ;b0363138-f229-4c4c-b196-ebbd59193d57Размер массива: 12
Массив Бабушкина (вычисление):3 10 8 4 11 5 2 7 1 6 0 9
Макс long long: 9223372036854775807
Число Бабушкина: 2891606649129
Дробь Бабушкина: 2891606649129/10000000000000
Массив Бабушкина (сортировка):3 10 8 4 11 5 2 7 1 6 0 9

До      После

-256    |      -604
 834    |      -416
 604    |      -351
 137    |      -256
 961    |      137
 256    |      256
-351    |      271
 491    |      491
-416    |      604
 271    |      750
-604    |      834
 750    |      961

PS D:\Learning\ALG\ALG_lab_1> 
```

Рисунок 6 — Корректная работа на массиве длиной 12

```
rt64\x5cbin\x5cdbg.exe' '--interpreter=mi' ;b0363138-f229-4c4c-b196-ebbd59193d57Размер массива: 15
Массив Бабушкина (вычисление):14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Макс long long: 9223372036854775807
Число Бабушкина: 435659737878916224
Дробь Бабушкина: 3403591702179033/7812500000000000
Массив Бабушкина (сортировка):14 13 12 11 10 9 8 7 6 5 4 3 2 1 9

До      После

15      |      2147483647
14      |      2
13      |      3
12      |      4
11      |      5
10      |      6
 9      |      7
 8      |      8
 7      |      9
 6      |      1
 5      |      11
 4      |      12
 3      |      13
 2      |      14
 1      |      15

PS D:\Learning\ALG\ALG_lab_1> 
```

Рисунок 7 — Ошибка: переполнение на массиве длиной 15

```

rosoft-MIEngine-Error-1pjl2zqk.54u' '--pid=Microsoft-MIEngine-Pid-dcsot34a.g3o' '--dbgExe=C:\x5cmsys64\x5cu
Массив Бабушкина (вычисление):11 4 13 12 7 9 10 8 3 5 2 14 1 6 15 0 -ebbd59193d57rosoft-MIEngine-Error-1pjl
Ошибка переполнения! Число Бабушкина слишком большое 5cmsys64\x5cucrt64\x5cbin\x5cgdb.exe' '
PS D:\Learning\ALG\ALG_lab_1> 9-4c4c-b196-ebbd59193d57P
Массив Бабушк

```

Рисунок 8 — Ошибка: переполнение числа Бабушкина

```

--interpreter=mi' ;b0363138-f229-4c4c-b196-ebbd59193d57Размер массива: 16
Массив Бабушкина (вычисление):0 8 15 5 6 11 2 14 4 12 3 13 9 1 10 7
Макс long long: 9223372036854775807
Число Бабушкина: 645539968189305216
Дробь Бабушкина: 5043281001478947/7812500000000000
Массив Бабушкина (сортировка):0 8 15 5 6 11 2 14 4 12 3 13 9 1 8 0

```

До	После
-987	176
213	-795
952	-713
69	-516
131	-6
555	69
-713	131
901	32759
-6	382
709	331
-516	15
883	555
331	709
-795	883
382	901
176	952

```

PS D:\Learning\ALG\ALG_lab_1>

```

Рисунок 9 — Ошибка: неизвестное переполнение на массиве длиной 16

ЗАКЛЮЧЕНИЕ

В ходе работы я интерпретировал алгоритм сортировки Бабушкина и разработал программу, максимально соответствующую ему и при этом имеющую хоть какой-то смысл. Стоит отметить, что ввиду значительного роста числа Бабушкина в зависимости от длины массива, переменные стандартных типов языка C быстро переполняются, что делает невозможным корректную работу программы для длин массивов больше 16. Также я столкнулся с огромным количеством переполнений неизвестной мне природы на длинах массива от 14 до 16, что ещё больше ограничивает использование программы.

Подводя итог, можно сказать, что данная реализация алгоритма показала себя как крайне непрактичная и ненадёжная. Вероятно реализация программы с поддержкой неограниченно больших чисел позволит работать с большими длинами, хотя в таком случае требования к памяти будут колоссальными.

Часть тестов закончена успешно.

Компилятор: g++.exe (tdm64-1) 5.1.0

Редактор:

Visual Studio Code 1.88.0