**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ**

**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**

**«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук

Департамент программной инженерии

**КОНСОЛЬНОЕ ПРИЛОЖЕНИЕ, НАХОДЯЩЕЕ ТРОЙКИ КОМПЛАНАРНЫХ ВЕКТОРОВ СРЕДИ ЗАДАННЫХ С ИСПОЛЬЗОВАНИЕМ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ**

Пояснительная записка

**Исполнитель**:

Студентка группы БПИ195

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Зубарева Н.Д./

«15» ноября 2020 г.

**Оглавление**

[1. Текст задания 2](#_Toc56374657)

[2. Применяемые расчетные методы 3](#_Toc56374658)

[2.1. Теория решения задания 3](#_Toc56374659)

[2.2. Организация многопоточности 3](#_Toc56374660)

[2.3. Ввод входных данных 3](#_Toc56374661)

[2.4. Вывод данных 3](#_Toc56374662)

[3. Тестирование программы 4](#_Toc56374663)

[3.1. Корректные значения 4](#_Toc56374666)

[3.2. Некорректные значения 5](#_Toc56374667)

[4. Список литературы 7](#_Toc56374668)

[5. Приложение кода 8](#_Toc56374669)

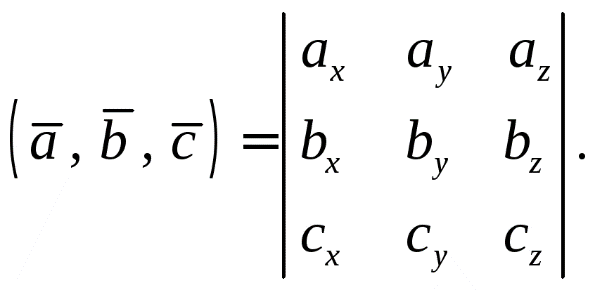
# Текст задания

Вариант 10: Найти все возможные тройки компланарных векторов. Входные данные: множество не равных между собой векторов (x, y, z), где x, y, z – числа. Оптимальное количество потоков выбрать самостоятельно.

# Применяемые расчетные методы

## Теория решения задания

По условию требуется находить компланарные тройки векторов среди данных. Согласно [2], для этого можно использовать значение смешанного произведения векторов, а именно, оно должно быть равно нулю. Также использована формула вычисления смешанного произведения по координатам трех данных векторов [3]:



## Организация многопоточности

Программа реализует модель итеративного параллелизма [4]: для каждого вектора создается поток, который далее перебирает все вторые и третьи элементы тройки из векторов с индексами после предыдущего в тройке и проверяет для полученной тройки равенство смешанного произведения нулю. Потоки, таким образом, фиксированы по индексам векторов и не вступают в конфликт, благодаря чему можно избежать использование блокировок и семафоров. Программа написана на языке Си и использует POSIX Threads.

## Ввод входных данных

Ввод данных осуществлен через командную строку и чтение из файла. В командной строке задается путь к файлу, из которого нужно считать данные. Далее в файле должно быть указано число векторов. Было принято решение ограничить количество 3 векторами снизу (1 и 2 вектора всегда компланарны) и 50 векторами сверху (при большем количестве работа программы в среднем занимает больше 10 секунд). При нехватке векторов программа завершается, при избытке – считывание не осуществляется после 50 векторов. Далее в файле должны быть записаны векторы в указанном выше количестве. Считывание всех чисел осуществляется с помощью функции fscanf в формате %ld для числа векторов и %lf для элементов векторов.

## Вывод данных

Тройки компланарных векторов выводятся в консоль с помощью функции printf с форматированием %g.

# Тестирование программы

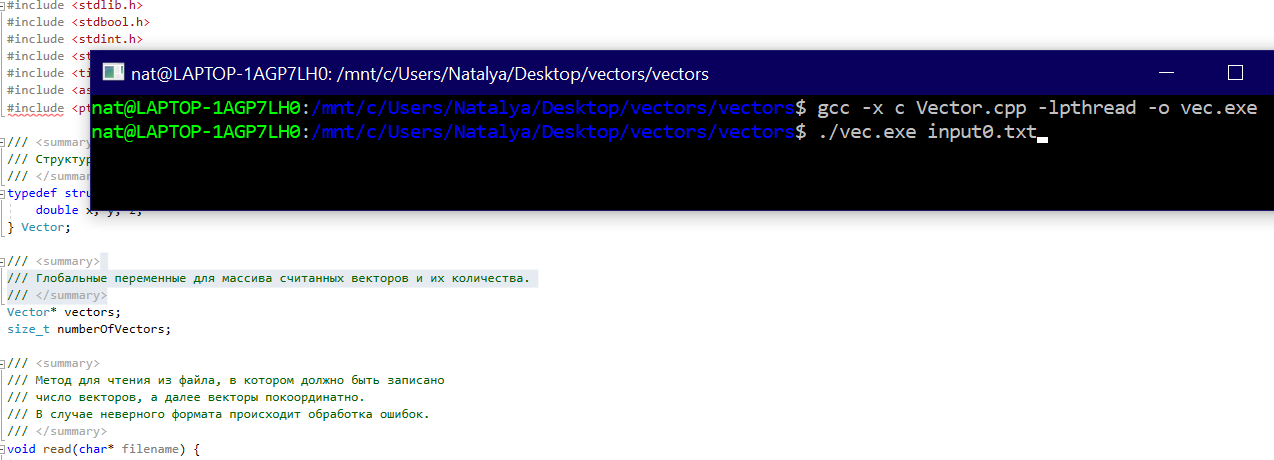
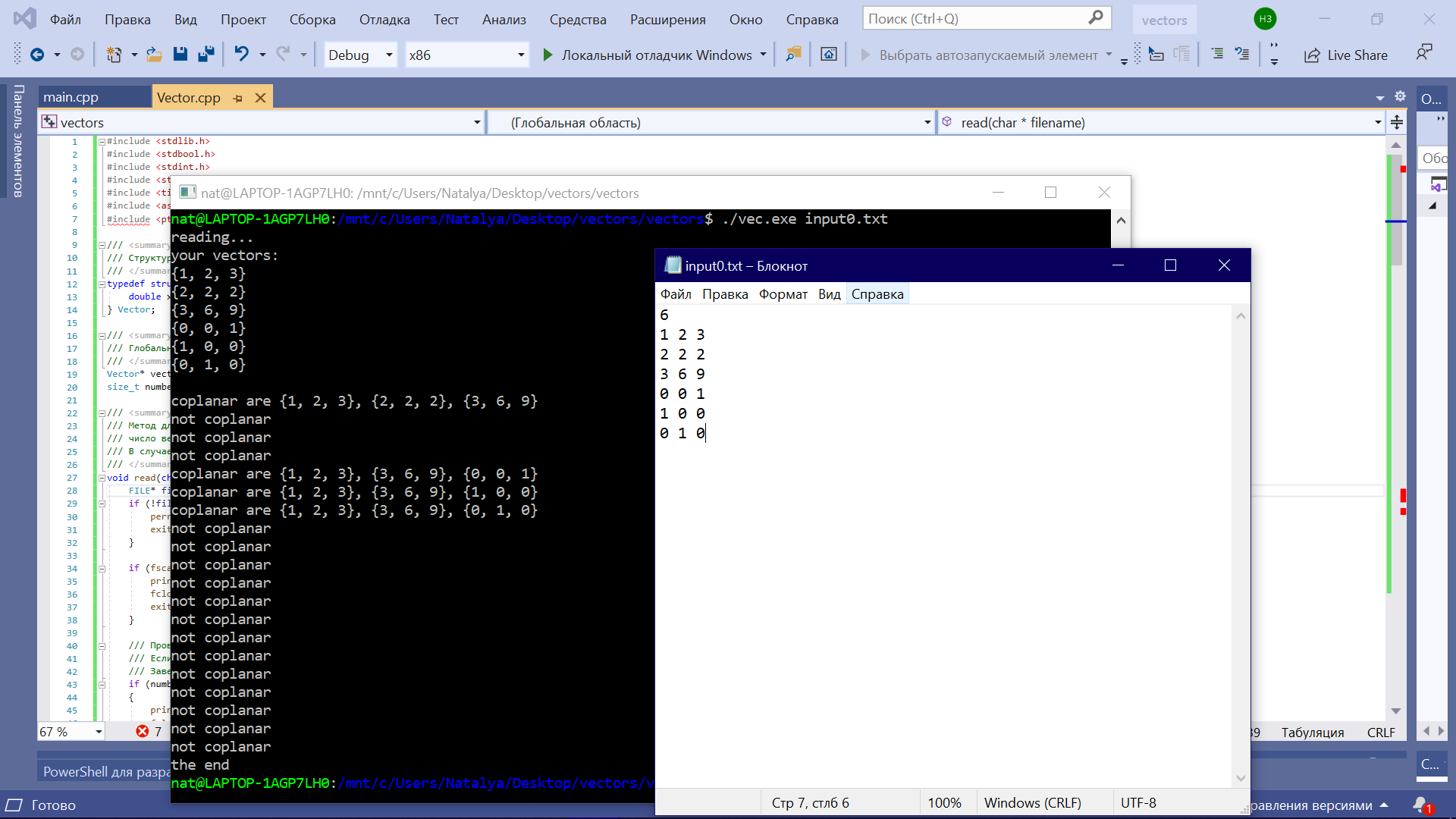
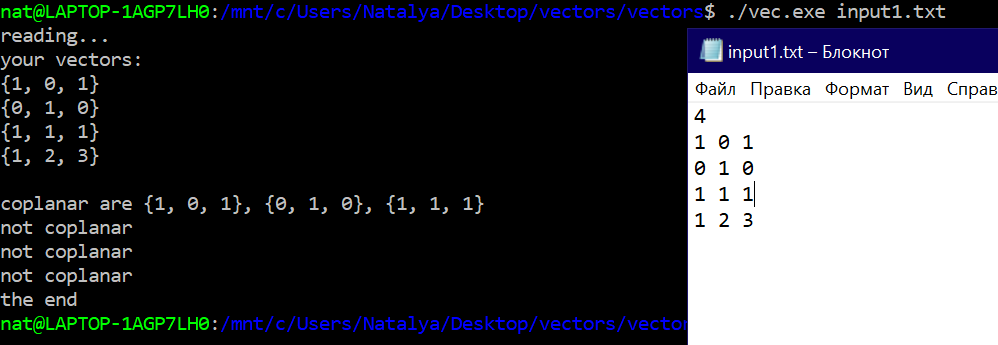
Программа компилируется и запускается следующим образом из командной строки (рисунок 1).

Рисунок 1 Команды компиляции и запуска программы



## Корректные значения

Программа осуществляет перебор троек векторов для нахождения компланарных. При некомпланарности тройки выводится сообщение без указания номера тройки, при компланарности – векторы, входящие в нее (рисунок 2, рисунок 3).

*Рисунок 2 Работа программы при корректных данных*

*Рисунок 3 Работа программы при корректных данных*

## Некорректные значения

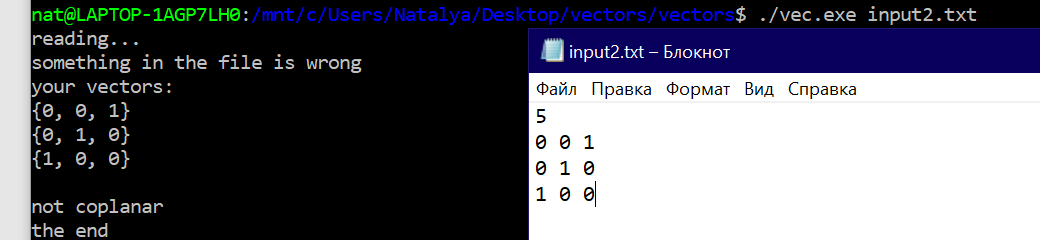
Программа также обрабатывает случаи ввода некорректных данных, например когда число векторов меньше указанного в файле числа (в этом случае считывается максимально возможное число векторов) (рисунок 4).

Рисунок 4 Работа программы при количестве векторов меньше указанного

Если указанное число не считывается верно, выводится сообщение об ошибке, и работа программы завершается (рисунок 5).

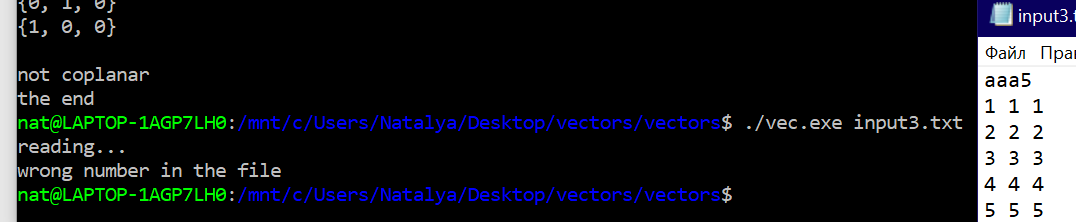


Рисунок 5 Работа программы при некорректном количестве векторов

Если какой-то из элементов векторов задан не числом, работа осуществляется со считанными до этого векторами (рисунок 6).

**

Рисунок 6 Работа программы при некорректном элементе вектора

В случае, когда векторов меньше трех, выводится сообщение о нехватке векторов и работа программы завершается (рисунок 7).

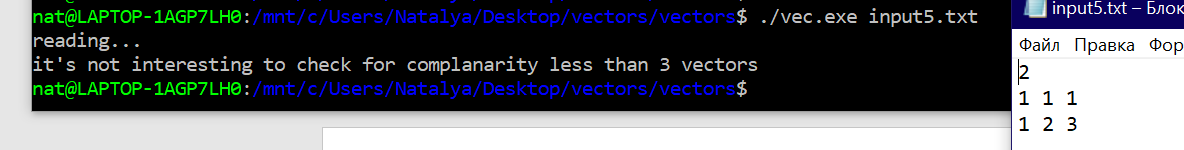
****

Рисунок 7 Сообщение об ошибке и завершение работы при недостаточном количестве векторов

Если векторов задано слишком много (больше 50), выводится сообщение об избытке и работа осуществляется с 50 векторами (рисунок 8).

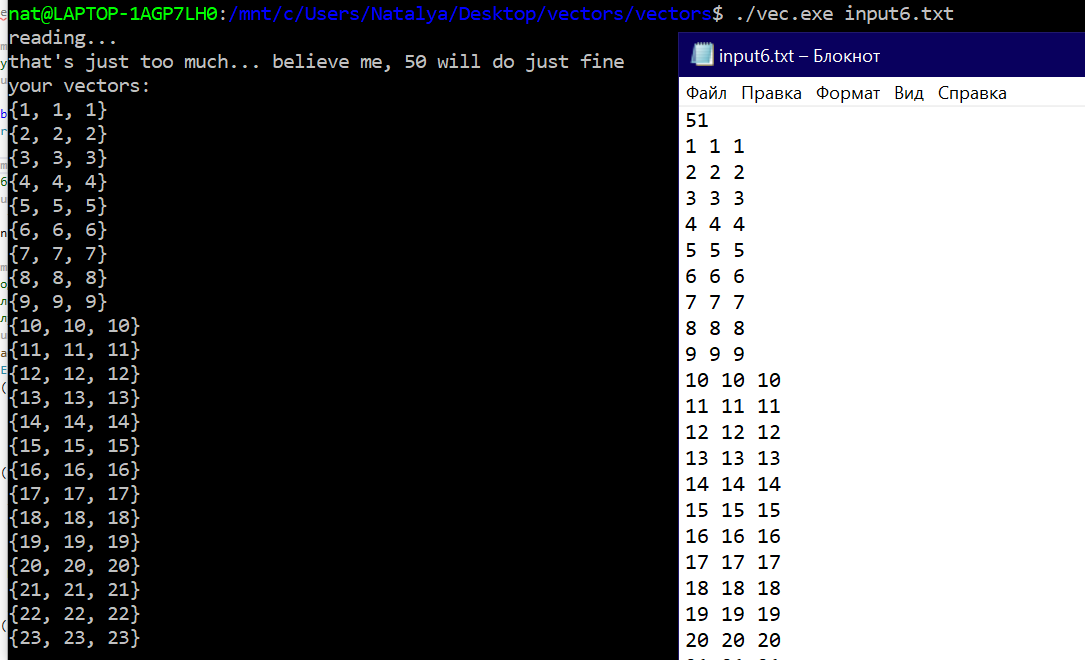


Рисунок 8 Работа программы при числе векторов больше 50

# Список литературы

[1] Инструкция по составлению пояснительной записки [Электронный ресурс]. //URL: http://softcraft.ru/edu/ comparch/tasks/mp01/ (Дата обращения: 30.10.2020, режим доступа: свободный)

[2] Статья «Coplanarity» Wikipedia.org //URL: https://en.wikipedia.org/wiki/Coplanarity (Дата обращения: 15.11.2020, режим доступа: свободный)

[3] Статья «Triple product» Wikipedia.org //URL: https://en.wikipedia.org/wiki/Triple\_product#Scalar\_triple\_product (Дата обращения: 15.11.2020, режим доступа: свободный)

[4] Практические приемы построения многопоточных приложений [Электронный ресурс].  
//URL: [http://softcraft.ru/edu/comparch/tasks/t03/](https://vk.com/away.php?utf=1&to=http%3A%2F%2Fsoftcraft.ru%2Fedu%2Fcomparch%2Ftasks%2Ft03%2F) (Дата обращения: 15.11.2020, режим доступа: свободный)

# Приложение кода

#include <stdlib.h>

#include <stdbool.h>

#include <stdint.h>

#include <stdio.h>

#include <time.h>

#include <assert.h>

#include <pthread.h>

/// <summary>

/// Структура для вектора, состоящая из трех координат.

/// </summary>

typedef struct Vector {

double x, y, z;

} Vector;

/// <summary>

/// Глобальные переменные для массива считанных векторов и их количества.

/// </summary>

Vector\* vectors;

size\_t numberOfVectors;

/// <summary>

/// Метод для чтения из файла, в котором должно быть записано

/// число векторов, а далее векторы покоординатно.

/// В случае неверного формата происходит обработка ошибок.

/// </summary>

void read(char\* filename) {

FILE\* file = fopen(filename, "r");

if (!file) {

perror(filename);

exit(1);

}

if (fscanf(file, "%ld", &numberOfVectors) != 1) {

printf("wrong number in the file\n");

fclose(file);

exit(1);

}

/// Проверка на то, что количество векторов не меньше 3.

/// Если меньше, то искать компланарные тройки бесполезно.

/// Завершаем программу.

if (numberOfVectors < 3)

{

printf("it's not interesting to check for complanarity less than 3 vectors\n");

fclose(file);

exit(0);

}

/// Проверка на то, что количество векторов не превышает 50.

/// Если превышает, мы все же не будем считать больше 50 векторов.

if (numberOfVectors>50)

{

printf("that's just too much... believe me, 50 will do just fine\n");

numberOfVectors = 50;

}

/// Выделяем память для векторов в глобальной переменной и считываем.

vectors = (Vector\*)calloc(numberOfVectors, sizeof(Vector));

for (size\_t i = 0; i < numberOfVectors; i++) {

if (fscanf(file, "%lf", &vectors[i].x) != 1) {

numberOfVectors = i;

printf("something in the file is wrong\n");

break;

}

if (fscanf(file, "%lf", &vectors[i].y) != 1) {

numberOfVectors = i;

printf("something in the file is wrong\n");

break;

}

if (fscanf(file, "%lf", &vectors[i].z) != 1) {

numberOfVectors = i;

printf("something in the file is wrong\n");

break;

}

}

fclose(file);

}

/// <summary>

/// Метод для печати считанных векторов в консоль.

/// </summary>

void printVectors(Vector\* vectors) {

printf("your vectors:\n");

for (size\_t i = 0; i < numberOfVectors; i++)

{

printf("{%g, %g, %g}\n", vectors[i].x, vectors[i].y, vectors[i].z);

}

printf("\n");

}

/// <summary>

/// Метод для проверки, является ли тройка векторов компланарной через

/// равенство смешанного произведения нулю.

/// </summary>

bool coplanar(Vector a, Vector b, Vector c) {

int value = (a.x \* b.y \* c.z) + (a.y \* b.z \* c.x) + (a.z \* b.x \* c.y) -

(a.z \* b.y \* c.x) - (a.x \* b.z \* c.y) - (b.x \* a.y \* c.z);

return value == 0;

}

/// <summary>

/// Функция, выполняемая потоком. Каждый поток прикреплен к

/// первому вектору в тройке и внутри него происходит подбор

/// второго и третьего векторов, проверка их на компланарность

/// и вывод результата.

/// </summary>

void\* threadFunction(void\* index) {

size\_t i = (size\_t)index;

for (size\_t j = i + 1; j < numberOfVectors; ++j) {

for (size\_t k = j + 1; k < numberOfVectors; ++k) {

if (coplanar(vectors[i], vectors[j], vectors[k])) {

printf("coplanar are {%g, %g, %g}, {%g, %g, %g}, {%g, %g, %g}\n",

vectors[i].x, vectors[i].y, vectors[i].z,

vectors[j].x, vectors[j].y, vectors[j].z,

vectors[k].x, vectors[k].y, vectors[k].z);

}

else { printf("not coplanar\n"); }

}

}

return NULL;

}

/// <summary>

/// Метод для организации потоковой работы. Выделяется и затем

/// освобождается память под потоки, по одному на каждый вектор.

/// Далее массив заполняется потоками, у которых есть описанная

/// выше функция. Запускается работа потоков.

/// </summary>

void threadWork() {

pthread\_t\* threads = malloc(numberOfVectors \* sizeof(pthread\_t));

for (size\_t i = 0; i < numberOfVectors; i++) {

pthread\_t thread;

pthread\_create(&thread, NULL, threadFunction, (void\*)i);

threads[i] = thread;

}

for (size\_t i = 0; i < numberOfVectors; ++i) {

pthread\_join(threads[i], NULL);

}

free(threads);

}

/// <summary>

/// Точка входа, если аргументы входной строки верны,

/// отсюда вызываются методы чтения, вывода

/// считанных векторов, проверки компланарности.

/// </summary>

int main(int argc, char\*\* argv) {

if (argc != 2) {

printf("wrong number of args %d\n", argc);

return 1;

}

char\* input = argv[1];

printf("reading...\n");

read(input);

printVectors(vectors);

threadWork();

printf("the end\n");

/// Освобождение памяти, выделенной для векторов.

free(vectors);

return 0;

}