Правительство Российской Федерации

Национальный исследовательский университет

«Высшая школа экономики»

Департамент программной инженерии

Факультет компьютерных наук

Плоские геометрические фигуры, размещаемые в координатной сетке.

(Вариант 2, вариант 8)

Отчёт

Дисциплина: «Архитектура вычислительных систем»

Исполнитель:

студент группы БПИ203

Насыхова Анастасия Артемовна

Москва 2021

Описание полученного задания

Вариант 2, варинат 8.

***Текст задания:*** Необходимо реализовать консольное приложение для обработки элементов контейнера, используя объектно-ориентированный подход.

В контейнере могут находиться плоские геометрические фигуры, размещаемые в координатной сетке. Базовыми альтернативами являются такие фигуры:

1. Треугольник (парметры: целочисленные координаты всех трех углов);
2. Прямоугольник (параметры: целочисленные координаты левого верхнего и правого нижнего углов);
3. Круг (параметры: целочисленные координаты центра и радиус).

Для каждой фигуры необходимо реализовать следующий функционал:

1. Ввод параметров фигуры из файла;
2. Случайный ввод параметров фигуры;
3. Вывод параметров в файл;
4. Вычисление периметра.

Для контейнера необходимо реализовать следующий функционал:

1. Инициализация контейнера;
2. Очистка контейнера;
3. Ввод содержимого контейнера из указанного потока;
4. Слуйчайный ввод содержимого контейнера;
5. Вывод содержимого контейнера в указанный поток;
6. Сортировка (QuickSort) содержимого контейнера (элементы сортируются по периметру).

Используемые метрики:

Число интерфейсных модулей (заголовочных файлов): 6.

Число модулей реализации (файлов с определением программных объектов): 6.

Общий размер исходных текстов: 807 строк.

Полученный размер исполняемого кода: 23,8 Kb.

Время выполнения программы для различных тестовых наборов данных:

1. Test 1: 254 824 нс
2. Test 2: 389 388 нс
3. Test 3: 216 466 нс
4. Test 4: 8 472 217 нс
5. Test 5: exception (обработанный)
6. Test 6: exception (обработанный)
7. Test 7: exception (обработанный)
8. Test 8: 627 630 нс
9. Test 9: 61 679 468 нс
10. Test 10: exception (обработанный)

Сравнительный анализ полученных характеристик

1. Число интерфейнсыйх модулей и модулей реализации не изменилось.
2. Общий размер исходных текстов уменьшился.
3. Размер исполняемого файла увеличился.
4. Время работы программы немного уменьшилось.

Структурная схема изучаемой архитектуры вычислительной системы

Таблица типов

|  |  |
| --- | --- |
| Name | Size |
| int | 4 |
| double | 8 |
| enum | 4 |
| char\* | 8 |
| Указатель | 8 |

Память данных

|  |  |
| --- | --- |
| Name | Size |
| Class Shape | 0 |
| Class Triangle  int x1  int x2  int x3  int y1  int y2  int y3  Color triangle\_color | 28  4[0]  4[4]  4[8]  4[12]  4[16]  4[20]  4[24] |
| Class Rectangle  int x1  int x2  int y1  int y2  Color rectangle\_color | 20  4[0]  4[4]  4[8]  4[12]  4[16] |
| Class circle  int x  int y  int r  Color circle\_color | 16  4[0]  4[4]  4[8]  4[12] |
| Class container  enum max\_len  int len  Shape\* cont | 8008  4[0]  4[4]  8 \* 1000[8] |

Память команд

|  |  |
| --- | --- |
| main()  int argc  char\* argv  container c  int size | 8024  4[0]  8[4]  8008[12]  4[8020] |
| Triangle::In(std::ifstream &ifst)  Triangle\* this  int color | 12  8[0]  4[8] |
| Triangle::InRnd()  Triangle\* this  double a  double b  double c | 20  8[0]  4[8]  4[12]  4[16] |
| Triangle::Out(std::ofstream &ofst)  Triangle\* this | 8  8[0] |
| Triangle::Perimeter()  Triangle\* this  double a  double b  double c | 32  8[0]  8[8]  8[16]  8[24] |

Исходный текст программы, разработанной в ходе выполнения задания

*Заголовочные файлы*

**1) rnd.h**

#ifndef \_\_rnd\_\_

#define \_\_rnd\_\_

#include <cstdlib>

//------------------------------------------------------------------------------

// rnd.h - содержит генератор случайных чисел в диапазоне от 0 до 30

//------------------------------------------------------------------------------

inline auto Random() {

return rand() % 30 + 1;

}

#endif //\_\_rnd\_\_

**2) Circle.h**

#ifndef \_\_Circle\_\_

#define \_\_Circle\_\_

//------------------------------------------------------------------------------

// Circle.h - содержит описание цветка

//------------------------------------------------------------------------------

#include <fstream>

#include <cstring>

#include "math.h"

#include "rnd.h"

#include "Shape.h"

//------------------------------------------------------------------------------

// Круг.

class Circle : public Shape {

public:

~Circle();

// Ввод круга из файла.

void In(std::ifstream &ifst) override;

// Случайный ввод круга.

void InRnd() override;

// Вывод круга в форматируемый поток.

void Out(std::ofstream &ofst) override;

// Вычисление периметра круга.

double Perimeter() override;

private:

// Цвет круга.

enum Color {

RED,

ORANGE,

YELLOW,

GREEN,

BLUE,

DARKBLUE,

PURPLE,

INVALID\_COLOR

};

Color circle\_color;

// Координаты круга и его радиус.

int x, y, r;

static Color GetColor(const int& color);

};

#endif //\_\_Circle\_\_

**3) Container.h**

#ifndef \_\_Container\_\_

#define \_\_Container\_\_

//------------------------------------------------------------------------------

// Container.h - содержит тип данных, представляющий простейший контейнер.

//------------------------------------------------------------------------------

#include "Shape.h"

#include "Rectangle.h"

#include "Circle.h"

#include "Triangle.h"

//------------------------------------------------------------------------------

// Простейший контейнер на основе одномерного массива

class Container {

public:

// Инициализация контейнера.

void Init();

// Очистка контейнера от элементов (освобождение памяти).

void Clear();

// Ввод содержимого контейнера из указанного потока.

void In(std::ifstream &ifst);

// Случайный ввод содержимого контейнера.

void InRnd(int size);

// Вывод содержимого контейнера в указанный поток.

void Out(std::ofstream &ofst);

// Сортировка контейнера по возрастанию QuickSort.

void Sort();

void quickSort(Shape\* cont[], int size);

private:

// Максимальная длина.

enum {

max\_len = 10000

};

// Текущая длина.

int len;

Shape\* cont[max\_len];

};

#endif //\_\_Container\_\_

**4) Rectangle.h**

#ifndef \_\_Rectangle\_\_

#define \_\_Rectangle\_\_

//------------------------------------------------------------------------------

// Rectangle.h - содержит описание дерева

//------------------------------------------------------------------------------

#include <fstream>

#include <cstring>

#include "math.h"

#include "rnd.h"

#include "Shape.h"

//------------------------------------------------------------------------------

// Прямоугольник.

class Rectangle : public Shape {

public:

~Rectangle();

// Ввод прямоугольника из файла.

void In(std::ifstream &ifst) override;

// Случайный ввод прямоугольника.

void InRnd() override;

// Вывод прямоугольника в форматируемый поток.

void Out(std::ofstream &ofst) override;

// Вычисление периметра прямоугольника.

double Perimeter() override;

private:

// Цвет прямоугольника.

enum Color {

RED,

ORANGE,

YELLOW,

GREEN,

BLUE,

DARKBLUE,

PURPLE,

INVALID\_COLOR

};

Color rectangle\_color;

// Координаты прямоугольника.

int x1, x2, y1, y2;

static Color GetColor(const int& color);

};

#endif //\_\_Rectangle\_\_

**5) Shape.h**

#ifndef \_\_Shape\_\_

#define \_\_Shape\_\_

//------------------------------------------------------------------------------

// Shape.h - содержит описание растения(обобщающего типа данных)

//------------------------------------------------------------------------------

#include <fstream>

#include <cstring>

#include "rnd.h"

//------------------------------------------------------------------------------

// Класс, обобщающий все имеющиеся фигуры.

class Shape {

public:

virtual ~Shape() = default;

// Ввод обобщенной фигуры из файла.

virtual void In(std::ifstream &ifst) = 0;

// Случайный ввод обобщенной фигуры.

virtual void InRnd() = 0;

// Вывод обобщенной фигуры в форматируемый поток.

virtual void Out(std::ofstream &ofst) = 0;

// Вычисление периметра фигуры.

virtual double Perimeter() = 0;

};

#endif //\_\_Shape\_\_

**6) Triangle.h**

#ifndef \_\_Triangle\_\_

#define \_\_Triangle\_\_

//------------------------------------------------------------------------------

// Triangle.h - содержит описание кустарника

//------------------------------------------------------------------------------

#include <fstream>

#include <cstring>

#include "math.h"

#include "rnd.h"

#include "Shape.h"

//------------------------------------------------------------------------------

// Треугольник.

class Triangle : public Shape {

public:

~Triangle();

// Ввод кустаринка из файла

void In(std::ifstream &ifst) override;

// Случайный ввод кустарника

void InRnd() override;

// Вывод кустарника в форматируемый поток

void Out(std::ofstream &ofst) override;

// Вычисление периметра треугольника.

double Perimeter() override;

private:

// Цвет треугольника.

enum Color {

RED,

ORANGE,

YELLOW,

GREEN,

BLUE,

DARKBLUE,

PURPLE,

INVALID\_COLOR

};

Color triangle\_color;

// Координаты треугольника.

int x1, x2, x3, y1, y2, y3;

static Color GetColor(const int &color);

};

#endif //\_\_Triangle\_\_

*Исполняемые файлы*

**1) main.cpp**

//------------------------------------------------------------------------------

// main.cpp - содержит главную функцию,

// обеспечивающую простое тестирование

//------------------------------------------------------------------------------

#include <iostream>

#include <cstdlib> // для функций rand() и srand()

#include <ctime> // для функции time()

#include <cstring>

#include <chrono> // для вычисления времени

typedef std::chrono::high\_resolution\_clock Clock;

#include "Container.h"

void errMessage1() {

std::cout << "Incorrect command line!\n"

" Waited:\n"

" command -f InputFile OutputFile SortedFile\n"

" Or:\n"

" command -n number OutputFile SortedFile\n";

}

void errMessage2() {

std::cout << "Incorrect qualifier value!\n"

" Waited:\n"

" command -f InputFile OutputFile SortedFile\n"

" Or:\n"

" command -n number OutputFile SortedFile\n";

}

//------------------------------------------------------------------------------

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

auto t1 = Clock::now();

if (argc != 5) {

errMessage1();

return 1;

}

std::cout << "Start\n";

Container c;

c.Init();

if (!strcmp(argv[1], "-f")) {

std::ifstream ifst(argv[2]);

c.In(ifst);

} else if (!strcmp(argv[1], "-n")) {

auto size = atoi(argv[2]);

if ((size < 1) || (size > 10000)) {

std::cout << "incorrect numer of figures = "

<< size

<< ". Set 0 < number <= 10000\n";

return 3;

}

// системные часы в качестве инициализатора

srand(static\_cast<unsigned int>(time(0)));

// Заполнение контейнера генератором случайных чисел

c.InRnd(size);

} else {

errMessage2();

return 2;

}

// Вывод содержимого контейнера в файл

std::ofstream ofst1(argv[3]);

ofst1 << "Filled container:\n";

c.Out(ofst1);

// Сортировка контейнера по периметру.

c.Sort();

std::ofstream ofst2(argv[4]);

ofst2 << "Container after sorting:\n";

c.Out(ofst2);

// Очищение памяти.

c.Clear();

// Завершение программы.

std::cout << "Stop\n";

auto t2 = Clock::now();

std::cout << std::chrono::duration\_cast<std::chrono::nanoseconds>

(t2 - t1).count() << " nanoseconds\n";

return 0;

}

**2) Circle.cpp**

//------------------------------------------------------------------------------

// Circle.cpp - содержит функции обработки круга

//------------------------------------------------------------------------------

#include "Circle.h"

#define \_USE\_MATH\_DEFINES

#include <math.h>

//------------------------------------------------------------------------------

// Функция для определения цвета круга. по его номеру.

Circle::Color Circle::GetColor(const int& color) {

switch (color) {

case 1:

return Circle::Color::RED;

case 2:

return Circle::Color::ORANGE;

case 3:

return Circle::Color::YELLOW;

case 4:

return Circle::Color::GREEN;

case 5:

return Circle::Color::BLUE;

case 6:

return Circle::Color::DARKBLUE;

case 7:

return Circle::Color::PURPLE;

default:

return Circle::Color::INVALID\_COLOR;

}

}

// Ввод параметров круга из потока.

void Circle::In(std::ifstream &ifst) {

// Переменная для определения цвета.

int color;

ifst >> this->x >> this->y >> this->r >> color;

this->circle\_color = GetColor(color);

}

// Случайный ввод параметров круга.

void Circle::InRnd() {

this->x = Random();

this->y = Random();

this->r = Random();

int color = Random() % 7 + 1;

this->circle\_color = GetColor(color);

}

//------------------------------------------------------------------------------

// Вывод параметров круга в поток.

void Circle::Out(std::ofstream &ofst) {

ofst << "It is Circle: x = " << this->x

<< ", y = " << this->y

<< ", r = " << this->r

<< ", Color = ";

switch (this->circle\_color) {

case Circle::Color::RED:

ofst << "Red";

break;

case Circle::Color::ORANGE:

ofst << "Orange";

break;

case Circle::Color::YELLOW:

ofst << "Yellow";

break;

case Circle::Color::GREEN:

ofst << "Green";

break;

case Circle::Color::BLUE:

ofst << "Blue";

break;

case Circle::Color::DARKBLUE:

ofst << "Darkblue";

break;

case Circle::Color::PURPLE:

ofst << "Purple";

break;

default:

ofst << "Invalid color!";

}

ofst << ". Perimeter = " << Perimeter() << "\n";

}

//------------------------------------------------------------------------------

// Вычисление периметра круга.

double Circle::Perimeter() {

return 2.0 \* M\_PI \* this->r;

}

Circle::~Circle() {

}

**3) Rectangle.cpp**

//------------------------------------------------------------------------------

// Rectangle.cpp - содержит функции обработки прямоугольника.

//------------------------------------------------------------------------------

#include "Rectangle.h"

//------------------------------------------------------------------------------

// Функция для определения цвета прямоугольника по его номеру.

Rectangle::Color Rectangle::GetColor(const int& color) {

switch (color) {

case 1:

return Rectangle::Color::RED;

case 2:

return Rectangle::Color::ORANGE;

case 3:

return Rectangle::Color::YELLOW;

case 4:

return Rectangle::Color::GREEN;

case 5:

return Rectangle::Color::BLUE;

case 6:

return Rectangle::Color::DARKBLUE;

case 7:

return Rectangle::Color::PURPLE;

default:

return Rectangle::Color::INVALID\_COLOR;

}

}

// Ввод параметров прямоугольника из потока.

void Rectangle::In(std::ifstream &ifst) {

// Переменная для определения цвета.

int color;

ifst >> this->x1 >> this->y1 >> this->x2 >> this->y2 >> color;

this->rectangle\_color = GetColor(color);

}

// Случайный ввод параметров прямоугольника.

void Rectangle::InRnd() {

this->x1 = Random();

this->y1 = Random();

do {

this->x2 = Random();

this->y2 = Random();

} while (this->x1 == this->x2 && this->y1 == this->y2);

int color = Random() % 7 + 1;

this->rectangle\_color = GetColor(color);

}

//------------------------------------------------------------------------------

// Вывод параметров прямоугольника в поток.

void Rectangle::Out(std::ofstream &ofst) {

ofst << "It is Rectangle: x1 = " << this->x1 << ", y1 = " << this->y1

<< ", x2 = " << this->x2 << ", y2 = " << this->y2

<< ", Color = ";

switch (this->rectangle\_color) {

case Rectangle::Color::RED:

ofst << "Red";

break;

case Rectangle::Color::ORANGE:

ofst << "Orange";

break;

case Rectangle::Color::YELLOW:

ofst << "Yellow";

break;

case Rectangle::Color::GREEN:

ofst << "Green";

break;

case Rectangle::Color::BLUE:

ofst << "Blue";

break;

case Rectangle::Color::DARKBLUE:

ofst << "Darkblue";

break;

case Rectangle::Color::PURPLE:

ofst << "Purple";

break;

default:

ofst << "Invalid color!";

}

ofst << ". Perimeter = " << this->Perimeter() << "\n";

}

//------------------------------------------------------------------------------

// Вычисление периметра прямоугольника.

double Rectangle::Perimeter() {

return 2.0 \* abs(this->x2 - this->x1) + 2.0 \* abs(this->y2 - this->y1);

}

Rectangle::~Rectangle() {

}

**4) Triangle.cpp**

//------------------------------------------------------------------------------

// Triangle.cpp - содержит функции обработки треугольника

//------------------------------------------------------------------------------

#include "Triangle.h"

//------------------------------------------------------------------------------

// Функция для определения цвета треугольника по его номеру.

Triangle::Color Triangle::GetColor(const int &color) {

switch (color) {

case 1:

return Triangle::Color::RED;

case 2:

return Triangle::Color::ORANGE;

case 3:

return Triangle::Color::YELLOW;

case 4:

return Triangle::Color::GREEN;

case 5:

return Triangle::Color::BLUE;

case 6:

return Triangle::Color::DARKBLUE;

case 7:

return Triangle::Color::PURPLE;

default:

return Triangle::Color::INVALID\_COLOR;

}

}

// Ввод параметров треугольника из потока.

void Triangle::In(std::ifstream &ifst) {

// Переменная для определения цвета.

int color;

ifst >> this->x1 >> this->y1 >> this->x2 >> this->y2 >>

this->x3 >> this->y3 >> color;

this->triangle\_color = GetColor(color);

}

// Случайный ввод параметров треугольника.

void Triangle::InRnd() {

this->x1 = Random();

this->y1 = Random();

do {

this->x2 = Random();

this->y2 = Random();

} while (this->x1 == this->x2 && this->y1 == this->y2);

double a = sqrt((this->x2 - this->x1) \* (this->x2 - this->x1) +

(this->y2 - this->y1) \* (this->y2 - this->y1));

double b, c;

do {

this->x3 = Random();

this->y3 = Random();

b = sqrt((this->x3 - this->x1) \* (this->x3 - this->x1) +

(this->y3 - this->y1) \* (this->y3 - this->y1));

c = sqrt((this->x3 - this->x2) \* (this->x3 - this->x2) +

(this->y3 - this->y2) \* (this->y3 - this->y2));

} while ((c >= (a + b))

|| (a >= (c + b))

|| (b >= (c + a)));

int color = Random() % 7 + 1;

this->triangle\_color = GetColor(color);

}

//------------------------------------------------------------------------------

// Вывод параметров треугольника в поток.

void Triangle::Out(std::ofstream &ofst) {

ofst << "It is Triangle: x1 = " << this->x1 << ", y1 = " << this->y1

<< ", x2 = " << this->x2 << ", y2 = " << this->y2

<< ", x3 = " << this->x3 << ", y3 = " << this->y3

<< ", Color = ";

switch (this->triangle\_color) {

case Triangle::Color::RED:

ofst << "Red";

break;

case Triangle::Color::ORANGE:

ofst << "Orange";

break;

case Triangle::Color::YELLOW:

ofst << "Yellow";

break;

case Triangle::Color::GREEN:

ofst << "Green";

break;

case Triangle::Color::BLUE:

ofst << "Blue";

break;

case Triangle::Color::DARKBLUE:

ofst << "Darkblue";

break;

case Triangle::Color::PURPLE:

ofst << "Purple";

break;

default:

ofst << "Invalid color!";

}

ofst << ". Perimeter = " << this->Perimeter() << "\n";

}

//------------------------------------------------------------------------------

// Вычисление периметра треугольника.

double Triangle::Perimeter() {

double a = sqrt((this->x2 - this->x1) \* (this->x2 - this->x1) +

(this->y2 - this->y1) \* (this->y2 - this->y1));

double b = sqrt((this->x3 - this->x1) \* (this->x3 - this->x1)

+ (this->y3 - this->y1) \* (this->y3 - this->y1));

double c = sqrt((this->x3 - this->x2) \* (this->x3 - this->x2)

+ (this->y3 - this->y2) \* (this->y3 - this->y2));

return double(a + b + c);

}

Triangle::~Triangle() {

}

**5) Container.cpp**

//------------------------------------------------------------------------------

// Container.cpp - содержит функции обработки контейнера.

//------------------------------------------------------------------------------

#include "Container.h"

//------------------------------------------------------------------------------

// Инициализация контейнера.

void Container::Init() {

this->len = 0;

}

//------------------------------------------------------------------------------

// Очистка контейнера от элементов (освобождение памяти).

void Container::Clear() {

for (int i = 0; i < this->len; ++i) {

delete this->cont[i];

}

this->len = 0;

}

//------------------------------------------------------------------------------

// Ввод содержимого контейнера из указанного потока.

void Container::In(std::ifstream &ifst) {

while (!ifst.eof()) {

int input\_type;

ifst >> input\_type;

switch (input\_type) {

case 1:

this->cont[this->len] = new Rectangle();

this->cont[this->len]->In(ifst);

++(this->len);

break;

case 2:

this->cont[this->len] = new Triangle();

this->cont[this->len]->In(ifst);

++(this->len);

break;

case 3:

this->cont[this->len] = new Circle();

this->cont[this->len]->In(ifst);

++(this->len);

break;

}

}

}

//------------------------------------------------------------------------------

// Случайный ввод содержимого контейнера.

void Container::InRnd(int size) {

while (this->len < size) {

int input\_type = Random() % 3;

switch (input\_type) {

case 0:

this->cont[this->len] = new Rectangle();

this->cont[this->len]->InRnd();

++(this->len);

break;

case 1:

this->cont[this->len] = new Triangle();

this->cont[this->len]->InRnd();

++(this->len);

break;

case 2:

this->cont[this->len] = new Circle();

this->cont[this->len]->InRnd();

++(this->len);

break;

}

}

}

//------------------------------------------------------------------------------

// Вывод содержимого контейнера в указанный поток.

void Container::Out(std::ofstream &ofst) {

ofst << "Container contains " << this->len << " elements.\n";

for (int i = 0; i < this->len; ++i) {

ofst << i + 1 << ": ";

this->cont[i]->Out(ofst);

}

}

//------------------------------------------------------------------------------

// Быстрая сортировка контейнера по периметру.

void Container::Sort() {

quickSort(this->cont, this->len);

}

void Container::quickSort(Shape\* cont[], int size) {

//Указатели в начало и в конец массива

int i = 0;

int j = size - 1;

//Центральный элемент массива

double mid = cont[size / 2]->Perimeter();

//Делим массив

do {

//Пробегаем элементы, ищем те, которые нужно перекинуть

//в другую часть.

//В левой части массива пропускаем(оставляем на месте)

// элементы, которые меньше центрального

while (cont[i]->Perimeter() < mid) {

i++;

}

//В правой части пропускаем элементы, которые больше центрального

while (cont[j]->Perimeter() > mid) {

j--;

}

//Меняем элементы местами

if (i <= j) {

std::swap(cont[i], cont[j]);

i++;

j--;

}

} while (i <= j);

//Рекурсивные вызовы, если осталось, что сортировать

if (j > 0) {

//"Левый кусок"

quickSort(cont, j + 1);

}

if (i < size) {

//"Првый кусок"

quickSort(&cont[i], size - i);

}

}

**6) Shape.cpp**

//------------------------------------------------------------------------------

// Shape.cpp - содержит процедуры связанные с обработкой обобщенной фигуры

// и создания произвольной фигуры

//------------------------------------------------------------------------------

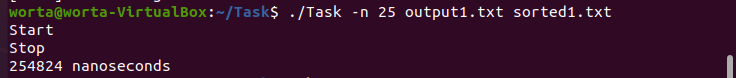
#include "Shape.h"

Тестовые наборы данных

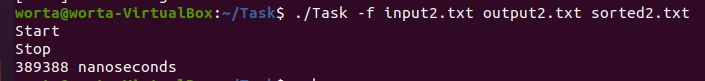
1. Command (путь к exe) -n 25 output1.txt sorted1.txt
2. Command -f input2.txt output2.txt sorted2.txt (5 элементов)
3. Command -n 15 output3.txt sorted3.txt
4. Command -n -1000 output4.txt sorted4.txt
5. Command
6. Command -n 0 output6.txt sorted6.txt
7. Command -n 10001 output7.txt sorted7.txt
8. Command -f input8.txt output8.txt sorted8.txt (20 элементов)
9. Command -n 10000 output9.txt sorted9.txt
10. Command er r o r

Результаты тестовых прогонов

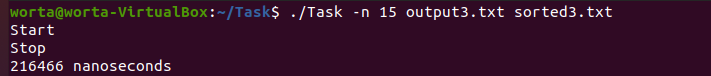
Тест 1:



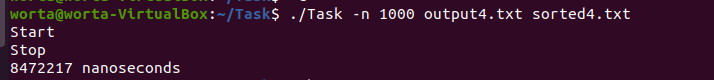
Тест 2:



Тест 3:



Тест 4:

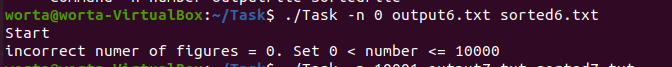


Тест 5:

Изображение выглядит как текст

Автоматически созданное описание

Тест 6:



Тест 7:

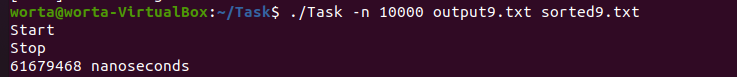


Тест 8:

Изображение выглядит как текст

Автоматически созданное описание

Тест 9:

Тест 10:

Изображение выглядит как текст

Автоматически созданное описание