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 14.

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

В контейнере могут находиться плоские геометрические фигуры, размещаемую 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).

Число интерфейсных модулей (заголовочных файлов): 6.

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

Общий размер исходных текстов: 1,59 Мб.

Полученный размер исполняемого кода: 1,59 Мб.

Время выполнения программы для различных тестовых наборов данных: 3,541 s.

Структурная схема изучаемой архитектуры вычислительной системы

Память данных

|  |  |
| --- | --- |
| Name | Size |
| Struct shape  enum key  key: k  union{  r: rectangle  t: triangle  c: circle  }  enum color | 64  4[0]  4[4]  16[8]  24[24]  12[48]  4[60] |
| Struct triangle  int x1  int x2  int x3  int y1  int y2  int y3 | 24  4[0]  4[4]  4[8]  4[12]  4[16]  4[20] |
| Struct rectangle  int x1  int x2  int y1  int y2 | 16  4[0]  4[4]  4[8]  4[12] |
| Struct circle  int x  int y  int r | 12  4[0]  4[4]  4[8] |
| Struct container  enum max\_len  int len  shape\* cont | 64008  4[0]  4[4]  64 \* 1000[8] |

Память команд

|  |  |
| --- | --- |
| main()  int argc  char\* argv  container c  int size  bool isFile  shape\* arr | 4[0]  8[4]  64008[12]  4[64020]  1[64024]  64008 \* size [64025] |

Исходный текст программы, разработанной в ходе выполнения задания

*Заголовояные файлы*

1) rnd.h

#ifndef \_\_rnd\_\_

#define \_\_rnd\_\_

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

// rnd.h - содержит генератор случайных чисел в диапазоне от 1 до 20

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

#include <cstdlib>

#include <iostream>

inline auto Random() {

return rand() % 20 + 1;

}

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

2) container.h

#ifndef \_\_container\_\_

#define \_\_container\_\_

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

// container.h - содержит тип данных,

// представляющий простейший контейнер

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

#include "shape.h"

#include <fstream>

#include <iostream>

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

// Простейший контейнер на основе одномерного массива

struct container {

enum { max\_len = 10000 }; // максимальная длина

int len; // текущая длина

shape\* cont;

};

// Инициализация контейнера

void Init(container& c, shape\* arr, int size);

// Очистка контейнера от элементов (освобождение памяти)

void Clear(container& c);

// Ввод содержимого контейнера из указанного потока

void In(container& c, ifstream& ifst);

// Случайный ввод содержимого контейнера

void InRnd(container& c, int size);

// Вывод содержимого контейнера в указанный поток

void Out(container& c, ofstream& ofst);

void QuickSort(container& c);

void qsortRecursive(double\* mas, int size, container& c);

#endif

3) shape.h

#ifndef \_\_shape\_\_

#define \_\_shape\_\_

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

// shape.h - содержит описание обобщающей геометрической фигуры.

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

#include "rectangle.h"

#include "triangle.h"

#include "circle.h"

#include <iostream>

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

// Структура, обобщающая все имеющиеся фигуры.

struct shape {

// Значения ключей для каждой из фигур.

enum key { RECTANGLE, TRIANGLE, CIRCLE };

key k; // Ключ.

// Используемые альтернативы.

union { // Используем простейшую реализацию.

rectangle r;

triangle t;

circle c;

};

enum color { RED, ORANGE, YELLOW, GREEN, BLUE, DARKBLUE, PURPLE };

};

// Ввод обобщенной фигуры

bool In(ifstream& ifdt, shape& s);

// Случайный ввод обобщенной фигуры

bool InRnd(shape& s);

// Вывод обобщенной фигуры

void Out(shape& s, ofstream& ofst);

// Вычисление периметра обобщенной фигуры

double Perimeter(shape& s);

#endif

4) rectangle.h

#ifndef \_\_rectangle\_\_

#define \_\_rectangle\_\_

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

// rectangle.h - содержит описание прямоугольника и его интерфейса

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

#include "rnd.h"

#include <fstream>

#include <iostream>

using namespace std;

// Прямоугольник

struct rectangle {

int x1, y1, x2, y2; // Координаты левого верхнего, правого нижнего.

};

// Ввод параметров прямоугольника из файла

void In(rectangle& r, ifstream& ifst);

// Случайный ввод параметров прямоугольника

void InRnd(rectangle& r);

// Вывод параметров прямоугольника в форматируемый поток

void Out(rectangle& r, ofstream& ofst);

// Вычисление периметра прямоугольника

double Perimeter(rectangle& r);

#endif //\_\_rectangle\_\_

5) triangle.h

#ifndef \_\_triangle\_\_

#define \_\_triangle\_\_

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

// triangle.h - содержит описание треугольника

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

#include "rnd.h"

#include <fstream>

#include <iostream>

using namespace std;

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

// Треугольник

struct triangle {

int x1, y1, x2, y2, x3, y3; // Координаты углов

};

// Ввод параметров треугольника из файла

void In(triangle& t, ifstream& ifst);

// Случайный ввод параметров треугольника

void InRnd(triangle& t);

// Вывод параметров треугольника в форматируемый поток

void Out(triangle& t, ofstream& ofst);

// Вычисление периметра треугольника

double Perimeter(triangle& t);

#endif //\_\_triangle\_\_

6) circle.h

#ifndef \_\_circle\_\_

#define \_\_circle\_\_

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

// circle.h - содержит описание треугольника

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

#include "rnd.h"

#include <fstream>

#include <iostream>

using namespace std;

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

// Круг

struct circle {

int x, y, r; // Координаты центра и радиус.

};

// Ввод параметров круга из файла

void In(circle& c, ifstream& ifst);

// Случайный ввод параметров круга

void InRnd(circle& c);

// Вывод параметров круга в форматируемый поток

void Out(circle& c, ofstream& ofst);

// Вычисление периметра круга

double Perimeter(circle& c);

#endif //\_\_circle\_\_

*Исполняемые файлы*

1) main.cpp

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

// main.cpp - содержит главную функцию,

// обеспечивающую простое тестирование

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

#include "container.h"

#include <iostream>

#include <fstream>

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

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

#include <cstring>

void errMessage1();

void errMessage2();

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

if (argc != 5) {

errMessage1();

return 1;

}

cout << "Start" << endl;

container c;

int size = 100;

bool isFile;

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

isFile = true;

}

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

isFile = false;

size = atoi(argv[2]);

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

cout << "incorrect numer of figures = "

<< size

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

return 3;

}

}

else {

errMessage2();

return 2;

}

// Автоматическое выделение памяти для массива

shape\* arr = new shape[size];

Init(c, arr, size);

std::cout << "Size = " << size << "\n";

// Проверка источника данных

if (isFile) {

ifstream ifst(argv[2]);

In(c, ifst);

}

else {

// системные часы в качестве инициализатора

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

// Заполнение контейнера генератором случайных чисел

InRnd(c, size);

}

// Вывод содержимого контейнера в файл

ofstream ofst1(argv[3]);

Out(c, ofst1);

// Сортировка контейнера (8 вариант)

ofstream ofst2(argv[4]);

QuickSort(c);

Out(c, ofst2);

// Очищение памяти.

Clear(c);

cout << "Stop" << endl;

return 0;

}

void errMessage1() {

cout << "Incorrect command line!\n"

" Waited:\n"

" command -f InFile FirstOutFile SecondOutFile\n"

" Or:\n"

" command -n number FirstOutFile SecondOutFile\n";

}

void errMessage2() {

cout << "Incorrect qualifier value!\n"

" Waited:\n"

" command -f infile outfile01 outfile02\n"

" Or:\n"

" command -n number outfile01 outfile02\n";

}

2) container.cpp

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

// container\_Constr.cpp - содержит функции обработки контейнера.

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

#include "container.h"

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

// Инициализация контейнера.

void Init(container& c, shape\* arr, int size) {

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

c.cont = arr;

c.len = size;

std::cout << "End Init\n";

}

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

// Очистка контейнера от элементов (освобождение памяти)

void Clear(container& c) {

delete[] c.cont;

c.len = 0;

}

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

// Ввод содержимого контейнера из указанного потока

void In(container& c, ifstream& ifst) {

while (!ifst.eof()) {

In(ifst, c.cont[c.len]);

c.len++;

}

}

// Случайный ввод содержимого контейнера

void InRnd(container& c, int size) {

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

c.len = size;

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

InRnd(c.cont[i]);

}

std::cout << "End Rnd\n";

}

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

// Вывод содержимого контейнера в указанный поток

void Out(container& c, ofstream& ofst) {

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

Out(c.cont[i], ofst);

}

}

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

// Быстрая сортировка контейнера по периметру.

void QuickSort(container& c) {

int size = c.len;

double\* mas = new double[size];

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

switch (c.cont->k) {

case shape::RECTANGLE:

mas[i] = Perimeter(c.cont->r);

break;

case shape::TRIANGLE:

mas[i] = Perimeter(c.cont->t);

break;

case shape::CIRCLE:

mas[i] = Perimeter(c.cont->c);

break;

default:

mas[i] = Perimeter(c.cont->r);

break;

}

}

qsortRecursive(mas, size, c);

}

void qsortRecursive(double\* mas, int size, container& c) {

//Указатели в начало и в конец массива

int i = 0;

int j = size - 1;

//Центральный элемент массива

double mid = mas[size / 2];

//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 (mas[i] < mid) {

i++;

}

//В правой части пропускаем элементы, которые больше центрального

while (mas[j] > mid) {

j--;

}

//Меняем элементы местами

if (i <= j) {

shape s = c.cont[i];

c.cont[i] = c.cont[j];

c.cont[j] = s;

double tmp = mas[i];

mas[i] = mas[j];

mas[j] = tmp;

i++;

j--;

}

} while (i <= j);

//Рекурсивные вызовы, если осталось, что сортировать

if (j > 0) {

//"Левый кусок"

qsortRecursive(mas, j + 1, c);

}

if (i < size) {

//"Првый кусок"

qsortRecursive(&mas[i], size - i, c);

}

}

3) shape.cpp

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

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

// и создания произвольной фигуры.

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

#include "shape.h"

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

// Ввод параметров обобщенной фигуры из файла

bool In(ifstream& ifst, shape& s) {

int k;

ifst >> k;

switch (k) {

case 1:

s.k = shape::RECTANGLE;

In(s.r, ifst);

return true;

case 2:

s.k = shape::TRIANGLE;

In(s.t, ifst);

return true;

case 3:

s.k = shape::CIRCLE;

In(s.c, ifst);

return true;

default:

return false;

}

}

// Случайный ввод обобщенной фигуры

bool InRnd(shape& s) {

auto k = rand() % 3 + 1;

switch (k) {

case 1:

s.k = shape::RECTANGLE;

InRnd(s.r);

return true;

case 2:

s.k = shape::TRIANGLE;

InRnd(s.t);

return true;

case 3:

s.k = shape::CIRCLE;

InRnd(s.c);

return true;

default:

return false;

}

}

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

// Вывод параметров текущей фигуры в поток

void Out(shape& s, ofstream& ofst) {

switch (s.k) {

case shape::RECTANGLE:

Out(s.r, ofst);

break;

case shape::TRIANGLE:

Out(s.t, ofst);

break;

case shape::CIRCLE:

Out(s.c, ofst);

break;

default:

ofst << "Incorrect figure!" << endl;

break;

}

}

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

// Вычисление периметра фигуры

double Perimeter(shape& s) {

switch (s.k) {

case shape::RECTANGLE:

return Perimeter(s.r);

case shape::TRIANGLE:

return Perimeter(s.t);

case shape::CIRCLE:

return Perimeter(s.c);

default:

return 0.0;

}

}

4) rectangle.cpp

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

// rectangle\_In.cpp - содержит процедуру ввода параметров

// для уже созданного прямоугольника

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

#include "rectangle.h"

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

// Ввод параметров прямоугольника из файла

void In(rectangle& r, ifstream& ifst) {

ifst >> r.x1 >> r.y1 >> r.x2 >> r.y2;

}

// Случайный ввод параметров прямоугольника

void InRnd(rectangle& r) {

r.x1 = Random();

r.y1 = Random();

r.x2 = Random();

r.y2 = Random();

}

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

// Вывод параметров прямоугольника в форматируемый поток

void Out(rectangle& r, ofstream& ofst) {

ofst << "It is Rectangle: x1 = " << r.x1 << ", y1 = " << r.y1

<< ", x2 = " << r.x2 << ", y2 = " << r.y2

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

}

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

// Вычисление периметра прямоугольника

double Perimeter(rectangle& r) {

return 2.0 \* abs(r.x2 - r.x1) + 2.0 \* abs(r.y2 - r.y1);

}

5) triangle.cpp

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

// triangle.cpp - содержит функции обработки треугольника

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

#include "triangle.h"

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

// Ввод параметров треугольника из потока

void In(triangle& t, ifstream& ifst) {

ifst >> t.x1 >> t.y1 >> t.x2 >> t.y2 >> t.x3 >> t.y3;

}

// Случайный ввод параметров треугольника

void InRnd(triangle& t) {

t.x1 = Random();

t.y1 = Random();

do {

t.x2 = Random();

t.y2 = Random();

} while (t.x1 == t.x2 || t.y1 == t.y2);

double a = sqrt((t.x2 - t.x1) \* (t.x2 - t.x1) + (t.y2 - t.y1) \* (t.y2 - t.y1));

double b, c;

do {

t.x3 = Random();

t.y3 = Random();

b = sqrt((t.x3 - t.x1) \* (t.x3 - t.x1) + (t.y3 - t.y1) \* (t.y3 - t.y1));

c = sqrt((t.x3 - t.x2) \* (t.x3 - t.x2) + (t.y3 - t.y2) \* (t.y3 - t.y2));

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

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

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

}

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

// Вывод параметров треугольника в поток

void Out(triangle& t, ofstream& ofst) {

ofst << "It is Triangle: x1 = " << t.x1 << ", y1 = " << t.y1

<< ", x2 = " << t.x2 << ", y2 = " << t.y2

<< ", x3 = " << t.x3 << ", y3 = " << t.y3

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

}

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

// Вычисление периметра треугольника

double Perimeter(triangle& t) {

double a = sqrt((t.x2 - t.x1) \* (t.x2 - t.x1) + (t.y2 - t.y1) \* (t.y2 - t.y1));

double b = sqrt((t.x3 - t.x1) \* (t.x3 - t.x1) + (t.y3 - t.y1) \* (t.y3 - t.y1));

double c = sqrt((t.x3 - t.x2) \* (t.x3 - t.x2) + (t.y3 - t.y2) \* (t.y3 - t.y2));

return double(a + b + c);

}

6) circle.cpp

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

// circle.cpp - содержит функции обработки треугольника.

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

#include "circle.h"

#include <iostream>

#define \_USE\_MATH\_DEFINES

#include <math.h>

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

// Ввод параметров треугольника из потока.

void In(circle& c, ifstream& ifst) {

ifst >> c.x >> c.y >> c.r;

}

// Случайный ввод параметров треугольника.

void InRnd(circle& c) {

c.x = Random();

c.y = Random();

c.r = Random();

}

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

// Вывод параметров треугольника в поток.

void Out(circle& c, ofstream& ofst) {

ofst << "It is Circle: x = "

<< c.x << ", y = " << c.y

<< ", r = " << c.r

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

}

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

// Вычисление периметра треугольника.

double Perimeter(circle& c) {

return 2.0 \* M\_PI \* c.r;

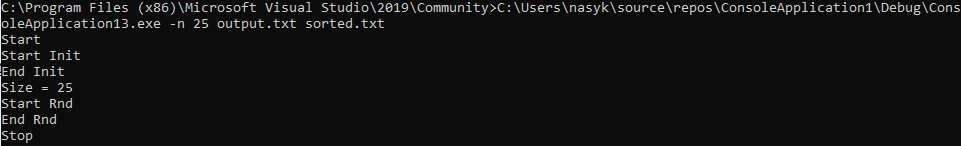
}

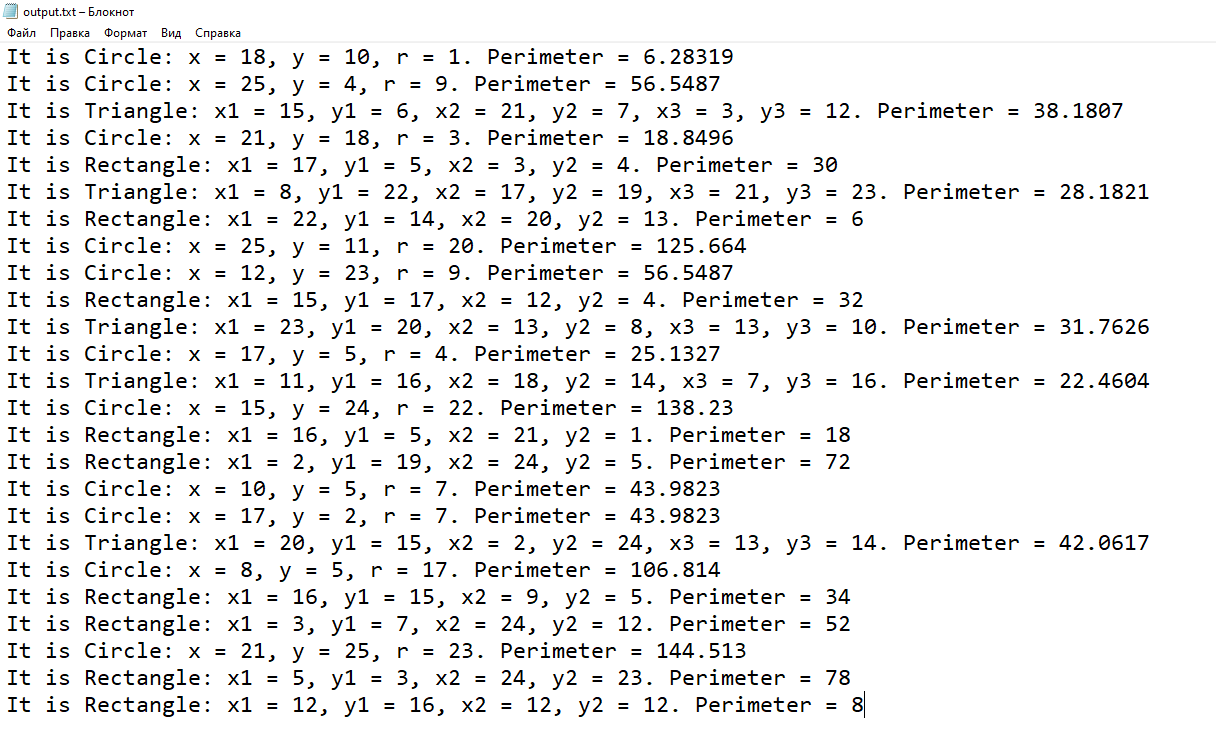
Тестовые наборы данных

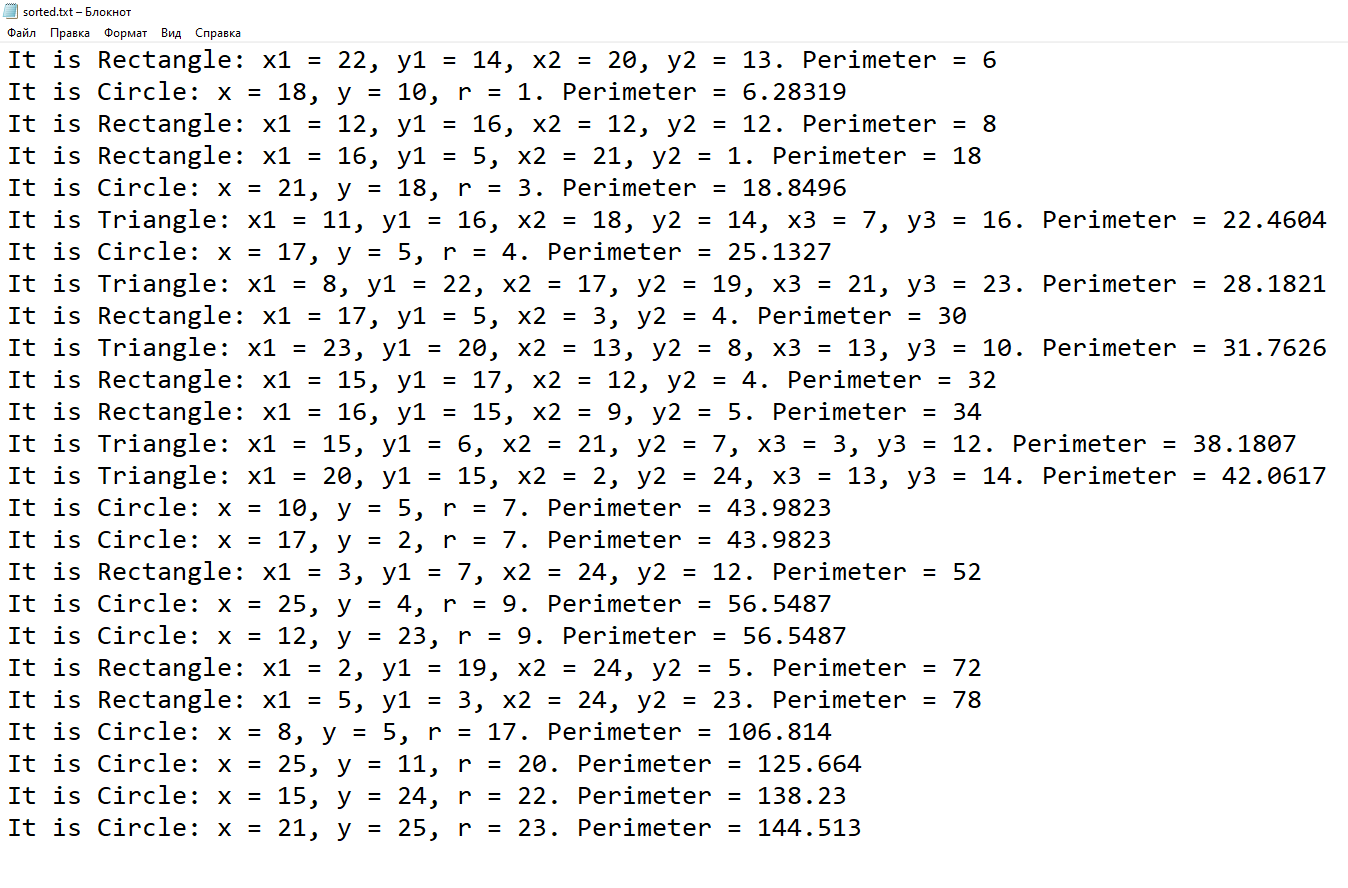
1. Command (путь к exe) -n 25 output.txt sorted.txt
2. Command -f input.txt output.txt sorted.txt
3. Command -n 15 output.txt sorted.txt
4. Command -n -1000 output.txt sorted.txt
5. Command

Результаты тестовых прогонов

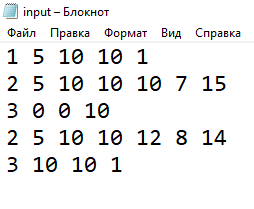
Тест 1:

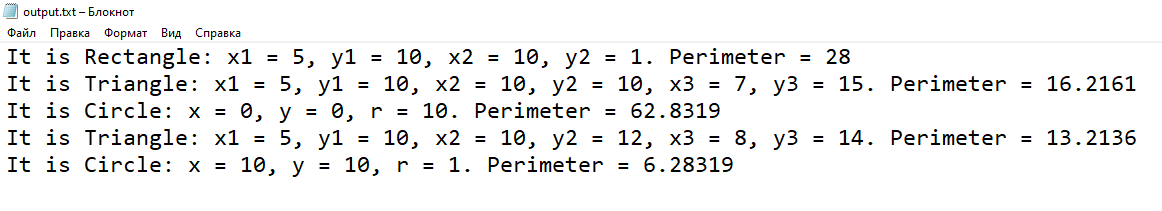


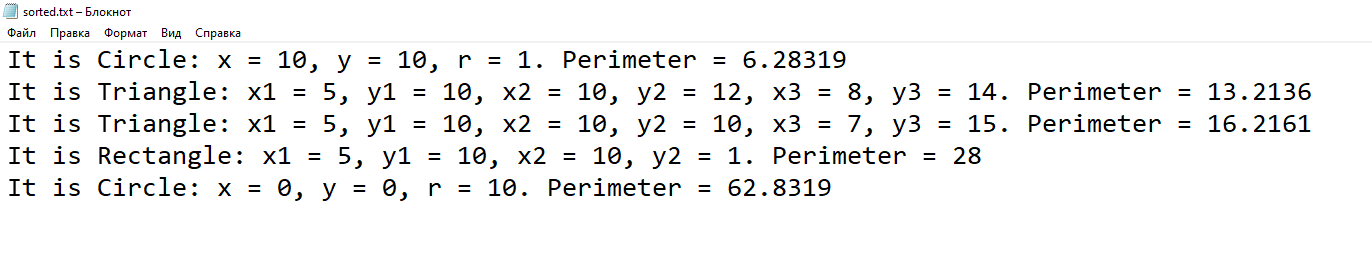




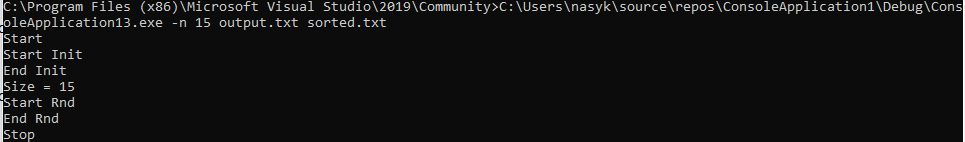
Тест 2:

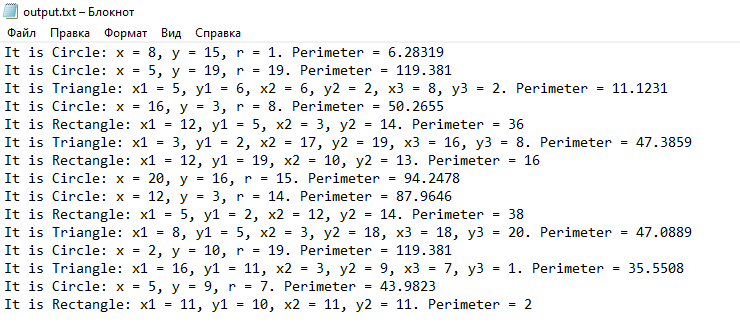


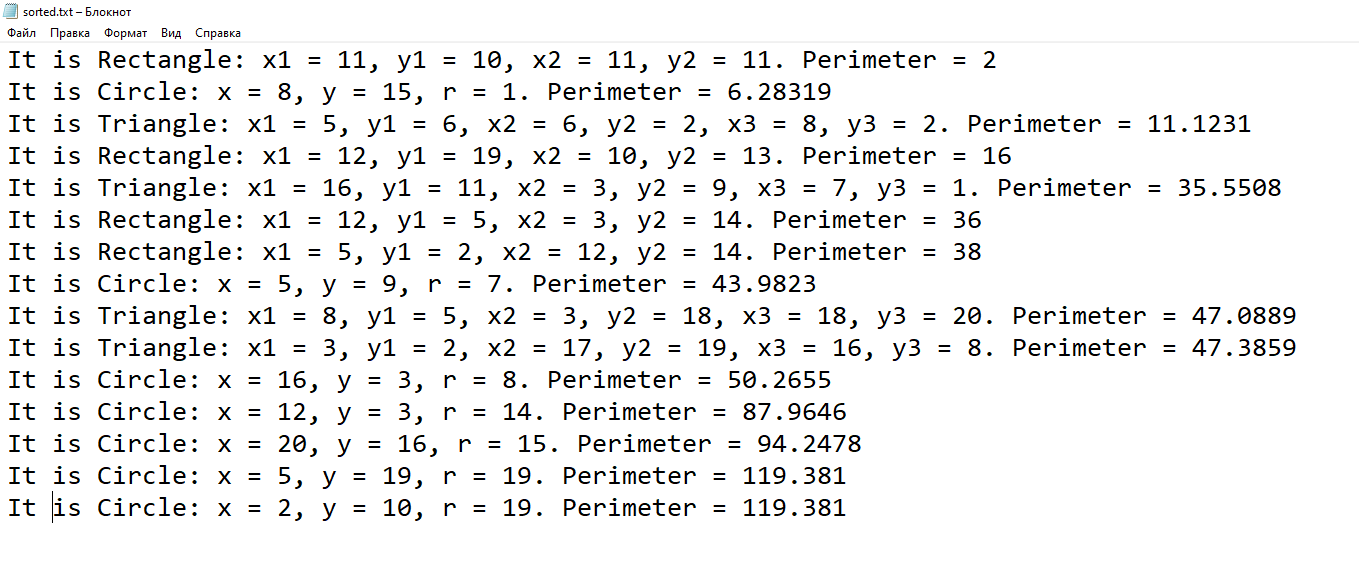




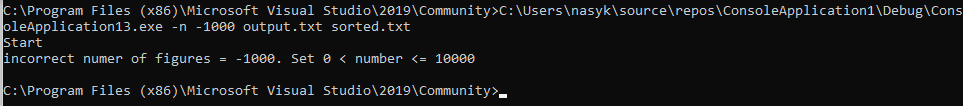
Тест 3:







Тест 4:



Тест 5:

