МИНИСТЕРСТВО ОБРАЗОВАНИЯ

И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

ИНЖЕНЕРНО-ТЕХНОЛОГИЧЕСКАЯ АКАДЕМИЯ

Институт компьютерных технологий и информационной безопасности

Кафедра математического обеспечения и применения ЭВМ

Лабораторная работа № 4

по курсу «Объектно-ориентированное программирование»

«Классы и объекты в C++»

13 ВАРИАНТ

Выполнили:

студенты гр. КТбо2-6

Шевченко А. В.

Проверил:

Тарасов С. А.

Таганрог 2020

Цель работы: ознакомление с классами-контейнерами библиотеки STL.

1. Вариант задания №13

Взять за основу программу из лабораторной работы 3, изменить класс Sheet таким образом, чтобы он был приспособлен для оптимального поиска фигур, содержащих введённую точку – вершину.

1. Спецификация классов с комментариями.

Особых изменений основной код не претерпел, лишь классы Point, Shape, Console были несколько изменены для оптимальной работы с новым классом Sheet – перегрузка оператора сравнения для Point, добавления уникального идентификатора в класс Shape и некоторые методы для загрузки, поиска, вывода и хранения информации в классе Console.

Класс Shape был полностью преобразован. Его полями являются три контейнера. Основной является контейнером map, хранящий указатели типа Shape, ключом является идентификатор фигуры. Вспомогательным контейнером для быстрого поиска является контейнер map, но в свою очередь он хранит по ключу типа Point контейнер типа set, в котором хранятся идентификаторы фигур, содержащих данную точку. Также был разработан контейнер типа list, хранящий идентификаторы фигур, которые хранятся в классе Sheet. Были разработаны методы для доступа к этим контейнерам, методы, проверяющие наличие в контейнерах введённый айди или точку, методы для поиска по айди или точке (перегруженный оператор []), метод для добавления фигуры в класс и методы удаляющие из контейнеров данные по айди или точке, метод для отчистки данных из контейнеров и метод для изменения ключей фигуры после её изменения (издержки того, что базовый класс абстрактный)

3. Используемые математические зависимости и алгоритмы.

Для перегрузки оператора “<” был применён следующий алгоритм:

1. Определение и сравнение номера четверти плоскости, где находятся точки. Если номер четверти первой точки меньше второй, то считается, что первая точка “меньше” второй (у точки O, нулевая четверть). Если номер четверти первой точки больше второй, то считается что первая точка “больше” второй. Если четверти совпадают, то переходим к пункту 2.
2. Для сравнения положения точек используется логика тригонометрии: чем меньший угол образует отрезок, соединяющий точку с началом координат, тем “меньше” положение точки. Какая точка “правее”, определяется с помощью векторного произведения и правила “правой тройки” (формула 1), если векторное произведение векторов двух точек, соединённых с началом координат, положительно, значит они образуют правую тройку, значит положение первой точки является “меньше”, чем положение второй. Если образуется левая тройка, то считается, что положение первой точки является “больше”, чем положение второй. Если векторы коллинеарны переходим к пункту 3.
3. Сравниваются расстояние от точек до начала координат с помощью метода Section (лаб 3), если расстояние первой точки до начала координат, меньше расстояния второй, то положение первой точки считается “меньше” второй, иначе считается “не меньше”

4. Диаграмма классов предоставлена на рисунке 1.

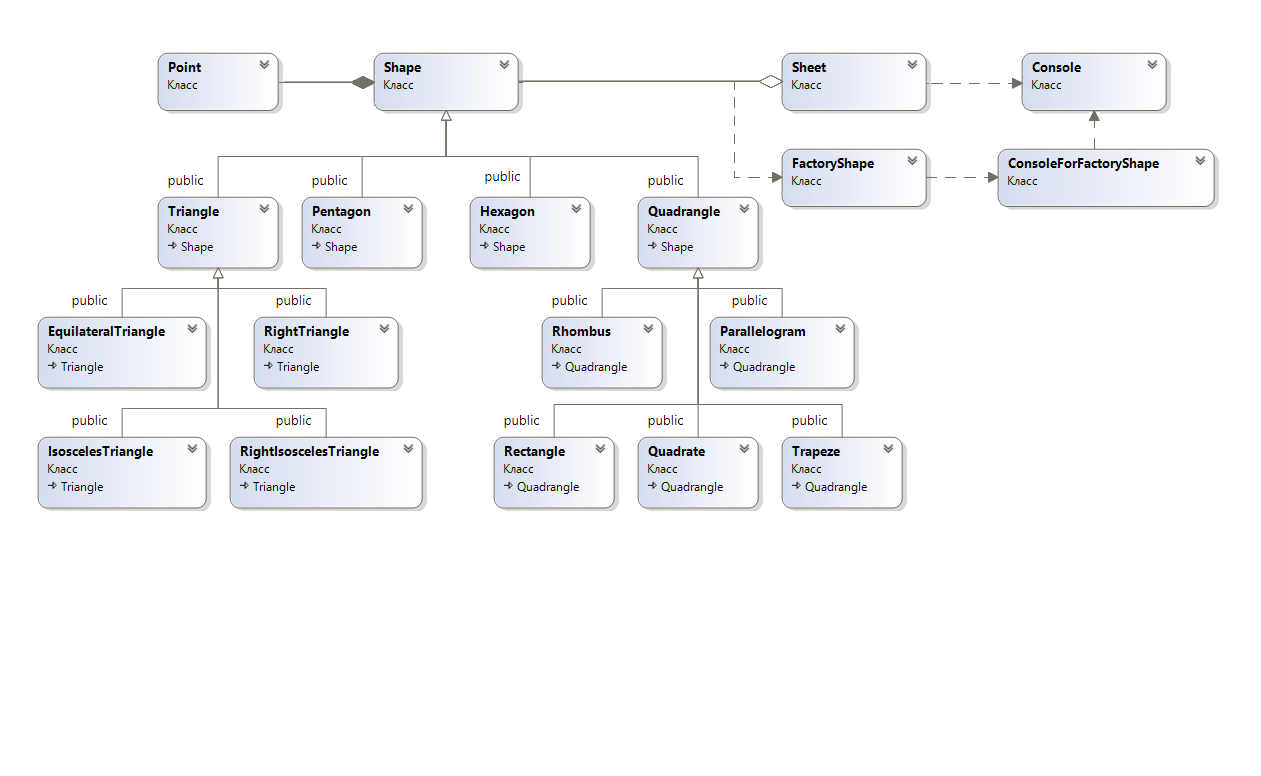


Рисунок 1 – Диаграмма классов

5. Листинг программы

Ниже будет приведены файлы, содержащие следующие классы: Point, Sheet, Console.

Point.h:

#include <iostream>

using std::ostream;

using std::istream;

class Point

{

friend class Shape;

public:

Point(double x = 0, double y = 0) : \_x(x), \_y(y) {}

double Section(const Point& B) const;

int Quarter() const;

friend double VectorsProduct(const Point& A, const Point& B);

friend bool LinesCross(const Point& A, const Point& B, const Point& C, const Point& D);

friend const Point operator+(const Point& left, const Point& right);

friend const Point operator-(const Point& left, const Point& right);

bool operator==(const Point& right) const;

bool operator!=(const Point& right) const;

bool operator<(const Point& B) const;

bool operator>(const Point& B) const;

friend istream& operator >> (istream& is, Point& B);

friend ostream& operator << (ostream& os, const Point& A);

double X() const {return \_x; }

double Y() const {return \_y; }

private:

double \_x, \_y;

};

Point.cpp:

#include "Point.h"

double Point::Section(const Point& B = 0) const // растояние между точкам

{

return sqrt((\_x - B.\_x) \* (\_x - B.\_x) + (\_y - B.\_y) \* (\_y - B.\_y));

}

int Point::Quarter() const

{

if (\_x == 0 && \_y == 0)

return 0; // иначе никак

if (\_x > 0 && \_y >= 0)

return 1;

if (\_x <= 0 && \_y > 0)

return 2;

if (\_x <= 0 && \_y < 0)

return 3;

return 4;

}

double VectorsProduct(const Point & A, const Point & B)

{

return A.\_x\* B.\_y - A.\_y \* B.\_x;

}

bool LinesCross(const Point & A, const Point & B, const Point & C, const Point & D)

{

Point A1, B1, C1, D1, A2, B2;

A1 = D - C;

B1 = A - C;

double v1 = VectorsProduct(A1, A - C);

double v2 = VectorsProduct(A1, B - C);

double v3 = VectorsProduct(C1, C - A);

double v4 = VectorsProduct(C1, D - A);

if ((v1 \* v2 < 0) && (v3 \* v4 < 0))

{

return 1;

}

else

{

return 0;

}

}

const Point operator+(const Point & left, const Point & right)

{

return Point(left.\_x + right.\_x, left.\_y + right.\_y);

}

const Point operator-(const Point & left, const Point & right)

{

return Point(left.\_x - right.\_x, left.\_y - right.\_y);

}

bool Point::operator==(const Point & right) const

{

return (\_x == right.\_x && \_y == right.\_y);

}

bool Point::operator!=(const Point & right) const

{

return !(\*this == right);

}

/\*

сравниваются положение точек по четвертям плоскости (1<2<3<4)

если равны, сравнивается углы с осью OX (с помощью знака векторного произведения и понятия правой тройки)

если равны, растояние до точки О

\*/

bool Point::operator<(const Point & B) const

{

int a = this->Quarter(), b = B.Quarter();

if (a == b)

{

double right3 = VectorsProduct(\*this, B);

if (abs(right3)>1e-9) // сравнение с 0, учитывая погрешность

return right3 > 0;

else

return this->Section() < B.Section();

}

else return a < b;

}

bool Point::operator>(const Point & B) const

{

return !(\*this < B || \*this == B);

}

istream& operator >> (istream & is, Point & B)

{

is >> B.\_x >> B.\_y;

return is;

}

ostream& operator << (ostream & os, const Point & A)

{

os << "(" << A.\_x << ";" << A.\_y << ")";

return os;

}

Sheet.h:

#pragma once

#include <map>

#include <set>

#include <list>

#include "Shape.h"

class Sheet

{

public:

std::list<int> GetIdIHave() const { return \_iHaveTheseId; }

std::map<int, Shape\*> GetMapById() const { return \_shapeById; }

std::map<Point, std::set<int>> GetMapByPoint() const { return \_shapesByPoint; }

void Insert(Shape\* ref);

Shape\* Extract(int id);

void EraseById(int id);

void EraseByPoint(const Point& point);

Shape& operator[](int index) const;

std::list<Shape\*> operator[](const Point& point) const;

void Clear();

int GetSize() const { return \_shapeById.size(); }

void IdEr(int id) const;

void PointEr(const Point& point) const;

~Sheet() { Clear(); }

private:

// попытка оптимизации по быстродействию в ущерб памяти, учитывая погрешности

std::map<int, Shape\*> \_shapeById;

std::map<Point, std::set<int>> \_shapesByPoint;

std::list<int> \_iHaveTheseId;

};

Point.cpp:

#include "Sheet.h"

///////////////////////////////////////////////////////////////////////////////////

// костыль, от которого не могу избавиться, //

// нужен для изменения ключей в \_ShapesByPoint после изменения параметров фигуры //

// создать копию и изменять её не могу, абстрактный класс, //

// а создавать обьект с помощью фабрики считаю расточительным //

// хранить указатели на точки тоже считаю не логичным, так как он должен быть //

// отсортирован и затрудняется работа в прицципе //

//

Shape\* Sheet::Extract(int id) //

{ //

for (auto it = \_shapesByPoint.begin(); it != \_shapesByPoint.end();) //

{ //

it->second.erase(id); //

//

if (it->second.empty()) //

it = \_shapesByPoint.erase(it); //

else //

it++; //

} //

//

Shape\* temp = \_shapeById[id]; //

//

int size = temp->GetSize(); //

//

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

{ //

\_shapesByPoint[temp->GetPtr()[i]].insert(id); //

} //

//

for (auto it : \_shapesByPoint) //

{ //

for (auto it1 : it.second) //

{ //

std::cout << it1; //

} //

} //

//

return temp; //

} //

///////////////////////////////////////////////////////////////////////////////////

void Sheet::Insert(Shape\* shape)

{

int id = shape->GetId();

\_iHaveTheseId.push\_back(id);

\_shapeById.insert(std::make\_pair(id, shape));

int size = shape->GetSize();

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

{

\_shapesByPoint[shape->GetPtr()[i]].insert(id);

}

}

void Sheet::EraseById(int id)

{

IdEr(id);

\_iHaveTheseId.remove(id);

Shape\* temp = \_shapeById[id];

int size = temp->GetSize();

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

{

\_shapesByPoint[temp->GetPtr()[i]].erase(id);

}

delete \_shapeById[id];

\_shapeById.erase(id);

}

void Sheet::EraseByPoint(const Point& point)

{

PointEr(point);

std::list<Shape\*> temp = operator[](point);

for (auto it : temp)

{

int size = it->GetSize();

int id = it->GetId();

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

{

\_shapesByPoint[it->GetPtr()[i]].erase(id);

}

}

\_shapesByPoint.erase(point);

for (auto it : temp)

{

int id = it->GetId();

delete \_shapeById[id];

\_shapeById.erase(id);

\_iHaveTheseId.remove(id);

}

}

Shape& Sheet::operator[](int id) const

{

IdEr(id);

return \*\_shapeById.at(id);

}

std::list<Shape\*> Sheet::operator[](const Point& point) const

{

PointEr(point);

std::list<Shape\*> temp;

for (auto it = \_shapesByPoint.at(point).begin(); it != \_shapesByPoint.at(point).end(); ++it)

{

temp.push\_back(\_shapeById.find(\*it)->second);

}

return temp;

}

void Sheet::Clear()

{

if (\_shapeById.size()) for (auto p : \_shapeById) delete p.second;

\_iHaveTheseId.clear();

\_shapeById.clear();

\_shapesByPoint.clear();

}

void Sheet::IdEr(int id) const

{

auto it = std::find(\_iHaveTheseId.begin(), \_iHaveTheseId.end(), id);

if (it == \_iHaveTheseId.end()) throw std::logic\_error("Не существующий Id!!");

}

void Sheet::PointEr(const Point & point) const

{

if (\_shapesByPoint.find(point) == \_shapesByPoint.end()) throw std::logic\_error("Фигур с такой вершиной нет!!");

}

Console.h:

#pragma once

#include <iostream>

#include "Sheet.h"

#include "ConcoleForFactoryShape.h"

class Console

{

friend class ConsoleForSheetMaager;

public:

void Run(); // исполнитель

private:

void Menu(); // функционал

void AddShapeToSheet(Sheet& XoY) const; // №1 добавить треугольник

void EraseById(Sheet& XoY); // №2 удалить по Id

void EraseByPoint(Sheet& XoY); // №3 удалить по точке

void Clear(Sheet& XoY); // №4 отчистить плоскость

void ShowSheet(const Sheet& XoY) const; // №5 показать координаиы треугольников

void ShowShapeByPoint(const Sheet& XoY) const; // №6 информация о треугольнике по точке

void ShowAllInformation(const Sheet& XoY) const; // №7 информация о всех треугольниках

void ShowShapeInformation(const Shape\* shape) const; // №8 информация о треугольнике по номеру

void ShowShapeInformationByPoint(const Sheet& XoY) const; // №9 информация о треугольнике по точке

void RotationShape(Sheet& XoY); // №10 вращение фигуры

void ShiftShape(Sheet& XoY); // №11 перемещение фигуры

void ComparisonShape(const Sheet& XoY) const; // №12 сравнение площади фигур

void IntersectionShape(const Sheet& XoY) const; // №13 пересечение фигур

void IncludeShape(const Sheet& XoY) const; // №14 вложение фигур

int ReadComand() const;

void ReadDataFromFile(Sheet& XoY, const string& nameFile);

void DounlandDataToFile(Sheet& XoY);

Shape\* ReadShape() const; // установка параметров треугольника

int ReadId(const Sheet& XoY) const; // ввести существующий индекс

void SetTwoId(int& i1, int& i2, const Sheet& XoY) const; // ввод пары существующих индексов

bool Confirmation() const; // запрос на подтверждение

bool ThereIsShape(const Sheet& XoY) const;

bool ThereAreShapes(const Sheet& XoY) const;

};

Console.cpp:

#include "Console.h"

#include <fstream>

using std::cout;

using std::cin;

using std::endl;

void Console::Run()

{

Sheet XoY;

try

{

cout << "Хотите использовать данные из файла?\n";

if (Confirmation())

{

string namefile;

cout << "Чтобы использовть данные с прошлого запуска нажимите 1\n";

cout << "Чтобы использовать файл с параметрами по умолчанию нажимите 2\n";

switch (ReadComand())

{

case 1: namefile = "result.txt"; break;

case 2: namefile = "source.txt"; break;

default: throw std::logic\_error("Вы что-то не так нажали");

}

ReadDataFromFile(XoY, namefile);

}

}

catch (std::exception & err)

{

std::cerr << endl << err.what() << endl << endl;

}

int operation;

do {

try {

Menu();

operation = ReadComand();

switch (operation)

{

case 1: AddShapeToSheet(XoY); break;

case 2: EraseById(XoY); break;

case 3: EraseByPoint(XoY); break;

case 4: Clear(XoY); break;

case 5: ShowSheet(XoY); break;

case 6: ShowShapeByPoint(XoY); break;

case 7: ShowAllInformation(XoY); break;

case 8:

if (ThereIsShape(XoY))

{

ShowShapeInformation(&XoY[ReadId(XoY)]);

}

break;

case 9: ShowShapeInformationByPoint(XoY); break;

case 10: RotationShape(XoY); break;

case 11: ShiftShape(XoY); break;

case 12: ComparisonShape(XoY); break;

case 13: IntersectionShape(XoY); break;

case 14: IncludeShape(XoY); break;

}

}

catch (std::exception & err)

{

std::cerr << endl << endl << err.what() << endl << endl;

}

} while (operation != 0);

try

{

cout << "Хотите сохранить данные в файла?\n";

if (Confirmation()) DounlandDataToFile(XoY);

}

catch (std::exception & err)

{

std::cerr << endl << endl << err.what() << endl << endl;

}

}

void Console::Menu()

{

cout << "Выберете действие\n";

cout << " 0: завершить работу\n";

cout << " 1: добавить фигуру на плоскость\n";

cout << " 2: удалить фигуру по Id\n";

cout << " 3: удалить фигуры содержащие введённую вершину\n";

cout << " 4: почистить плоскость\n";

cout << " 5: показать фигуру на плоскости\n";

cout << " 6: показать фигуры содержащие введённую вершину\n";

cout << " 7: показать информацию о всех фигурах на плоскости\n";

cout << " 8: показать информацию о фигурах по номеру\n";

cout << " 9: показать информацию о фигурах содержащие введённую вершину\n";

cout << "10: вращение выбранной фигуры на введённый угол\n";

cout << "11: перемещение выбранной фигуры на введённый вектор\n";

cout << "12: сравнить площади двух выбронных фигур\n";

cout << "13: проверить на пересечение две фигуры\n";

cout << "14: проверить на включене одной фигуры в другую\n";

}

void Console::AddShapeToSheet(Sheet& XoY) const

{

XoY.Insert(ReadShape());

cout << "\nФигура занесёна на плоскость.\n\n";

}

void Console::EraseById(Sheet& XoY)

{

if (ThereIsShape(XoY))

{

int index = ReadId(XoY);

if (Confirmation())

{

XoY.EraseById(index);

cout << "Фигура удалена!\n\n";

}

}

}

void Console::EraseByPoint(Sheet& XoY)

{

if (ThereIsShape(XoY))

{

cout << "Введите точку!\n";

Point point;

cin >> point;

if (Confirmation())

{

XoY.EraseByPoint(point);

cout << "\nФигуры содержащие эту точку удалены!\n\n";

}

}

}

void Console::Clear(Sheet& XoY)

{

if (ThereIsShape(XoY))

{

if (Confirmation())

{

XoY.Clear();

cout << "Плоскость отчищена!\n\n";

}

}

}

void Console::ShowSheet(const Sheet& XoY) const

{

if (ThereIsShape(XoY))

{

for (const auto &p : XoY.GetMapById())

{

cout << "Id " << p.second->GetId() << " " << \*p.second << endl << endl;

}

}

}

void Console::ShowShapeByPoint(const Sheet& XoY) const

{

if (ThereIsShape(XoY))

{

cout << "Введите точку!\n";

Point point;

cin >> point;

for (const auto& p : XoY[point])

{

cout << "Id " << p->GetId() << " " << \*p << endl << endl;

}

}

}

void Console::ShowAllInformation(const Sheet& XoY) const

{

if (ThereIsShape(XoY))

{

for (const auto& p : XoY.GetMapById())

{

ShowShapeInformation(p.second);

}

}

}

void Console::ShowShapeInformation(const Shape\* shape) const

{

cout << "Id " << shape->GetId() << endl;

cout << shape->TypeOfShape() << endl;

cout << \*shape << endl;

cout << "S= " << shape->Squear() << endl;

cout << "P= " << shape->Perimeter() << endl;

cout << "Координыты центра " << shape->Center() << endl << endl;

}

void Console::ShowShapeInformationByPoint(const Sheet& XoY) const

{

if (ThereIsShape(XoY))

{

cout << "Введите точку!\n";

Point point;

cin >> point;

for (const auto& p : XoY[point])

{

ShowShapeInformation(p);

}

}

}

void Console::RotationShape(Sheet& XoY)

{

if (ThereIsShape(XoY))

{

int index = ReadId(XoY);

int angle;

cout << "Введите угол поворота в градусах\n";

cin >> angle;

if (Confirmation())

{

cout << "\nФигура успешно повёрнута.\n" << XoY[index].Rotation(angle) << endl;

XoY.Extract(index);

}

}

}

void Console::ShiftShape(Sheet& XoY)

{

if (ThereIsShape(XoY))

{

int index = ReadId(XoY);

Point vector;

cout << "Введите вектор смещения\n";

cin >> vector;

if (Confirmation())

{

cout << "\nФигура сдвинута на введённый вектор.\n" << XoY[index].Shift(vector) << endl << endl;

XoY.Extract(index);

}

}

}

void Console::ComparisonShape(const Sheet& XoY) const

{

if (ThereAreShapes(XoY))

{

int i1, i2;

if (XoY.GetSize() == 2)

{

i1 = XoY.GetMapById().cbegin()->first;

i2 = XoY.GetMapById().crbegin()->first;

}

else

{

SetTwoId(i1, i2, XoY);

}

cout << "Первая фигура:\n" << XoY[i1] << endl;

cout << "Вторая фигура:\n" << XoY[i2] << endl;

double Comparison = XoY[i1].Comparison(XoY[i2]);

if (Comparison > 1)

{

cout << "площадь первой фигуры больше второй в " << Comparison << " раз\n\n";

}

else if (Comparison < -1)

{

cout << "площадь второй фигуры больше первой в " << -Comparison << " раз\n\n";

}

else

{

cout << "площадь первой фигуры равна площади второй\n\n";

}

}

}

void Console::IntersectionShape(const Sheet& XoY) const

{

if (ThereAreShapes(XoY))

{

int i1, i2;

if (XoY.GetSize() == 2)

{

i1 = XoY.GetMapById().cbegin()->first;

i2 = XoY.GetMapById().crbegin()->first;

}

else

{

SetTwoId(i1, i2, XoY);

}

cout << "Первая фигура:\n" << XoY[i1] << endl;

cout << "Вторая фигура:\n" << XoY[i2] << endl;

if (XoY[i1].Intersection(XoY[i2]))

{

cout << "Есть пересечение.\n\n";

}

else

{

cout << "Нет пересечения.\n\n";

}

}

}

void Console::IncludeShape(const Sheet& XoY) const

{

if (ThereAreShapes(XoY))

{

int i1, i2;

SetTwoId(i1, i2, XoY);

cout << "Первая фигура:\n" << XoY[i1] << endl;

cout << "Вторая фигура:\n" << XoY[i2] << endl;

if (XoY[i2].Include(XoY[i1]))

{

cout << "Первая фигура входит во вторую.\n\n";

}

else

{

cout << "Первая фигура НЕ входит во вторую.\n\n";

}

}

}

Shape\* Console::ReadShape() const

{

ConsoleForFactoryShape F;

return F.ReadShape();

}

void Console::ReadDataFromFile(Sheet& XoY, const string& nameFile)

{

std::ifstream fin;

fin.open(nameFile);

FactoryShape Factory;

if (fin.is\_open())

{

while (true)

{

int size;

fin >> size;

if (fin.eof()) break;

Point\* ptr = new Point[size];

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

{

fin >> ptr[i];

}

XoY.Insert(Factory.CreateShape(size, ptr));

delete[] ptr;

}

fin.close();

}

else throw std::logic\_error("Ошибка чтения файла!!");

}

void Console::DounlandDataToFile(Sheet& XoY)

{

std::ofstream fout;

fout.open("result.txt");

if (fout.is\_open())

{

if(XoY.GetMapById().empty()) throw std::logic\_error("К сожалению плоскость пуска, ничего не записалось!!");

for (auto it : XoY.GetMapById())

{

int size = it.second->GetSize();

fout << size;

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

{

fout << ' ' << it.second->GetPtr()[i].X()<<' '<< it.second->GetPtr()[i].Y();

}

fout << endl;

}

fout.close();

}

else throw std::logic\_error("Ошибка чтения файла!!");

}

int Console::ReadComand() const

{

std::string s;

cout << ">> ";

cin >> s;

cout << endl;

for (auto c : s)

{

if (!isdigit(c))

return -1;

}

return atoi(s.c\_str());

}

int Console::ReadId(const Sheet& XoY) const

{

if (XoY.GetSize() == 1) return XoY.GetMapById().cbegin()->first;

if (XoY.GetSize() > 1)

{

cout << "Доступные Id:";

for (auto it : XoY.GetIdIHave())

{

cout << " " << it;

}

cout << "\nВведите Id интересующей фигуры\n";

int id = ReadComand();

XoY.IdEr(id);

return id;

}

}

void Console::SetTwoId(int& i1, int& i2, const Sheet& XoY) const

{

i1 = ReadId(XoY);

do {

cout << "Введите Id друго фигуры, кроме " << i1 << endl;

i2 = ReadId(XoY);

} while (i1 == i2);

}

bool Console::Confirmation() const

{

string i;

cout << "Введите 0 для подтверждения, иначе любую другую комбинацию клавишь: ";

cin >> i;

if(i != "0") cout << "\nОтмена команды!\n\n";

return i == "0";

}

bool Console::ThereIsShape(const Sheet& XoY) const

{

if (XoY.GetSize() > 0) return true;

cout << "Плоскость пуста, добавте фигуру (команда 1)\n\n";

return false;

}

bool Console::ThereAreShapes(const Sheet& XoY) const

{

ThereIsShape(XoY);

if (XoY.GetSize() > 1) return true;

cout << "На плоскости меньше двух фигур, добавте фигуру (команда 1)\n\n";

return false;

}

source.txt:

3 0 0 0 1 1 0

4 -1 -1 -1 1 1 1 1 -1

3 -5 -5 -5 -3 -2 -5

3 53 -63 37 346 3468 -52

4 54 75 236 35 523 57 23 6

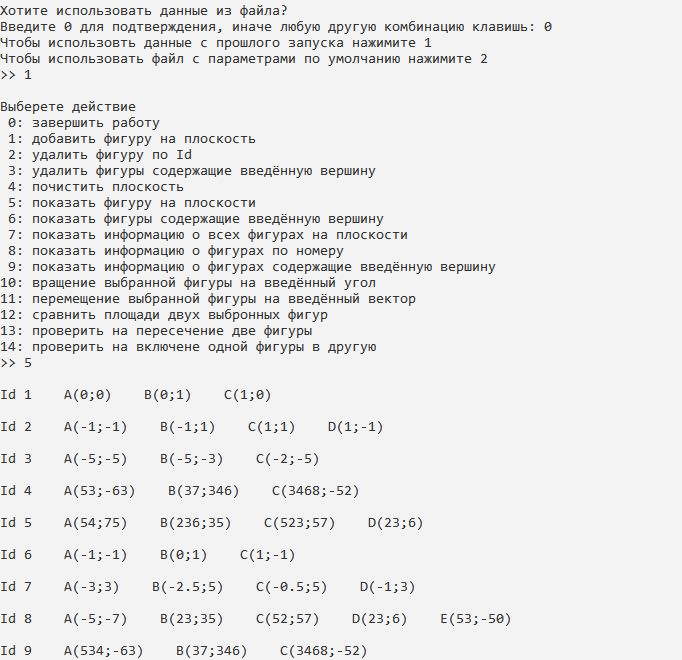
3 -1 -1 0 1 1 -1

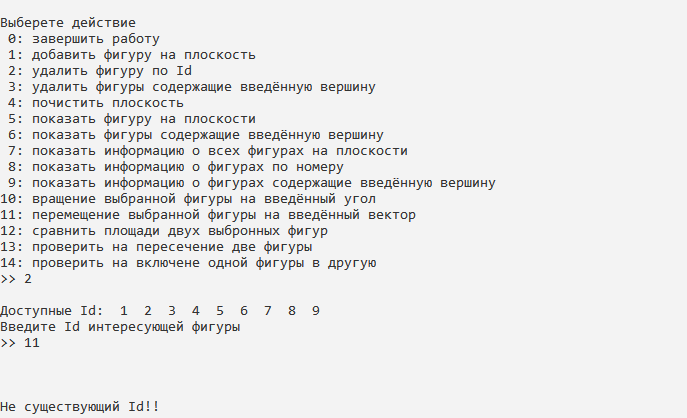
4 -3 3 -2.5 5 -0.5 5 -1 3

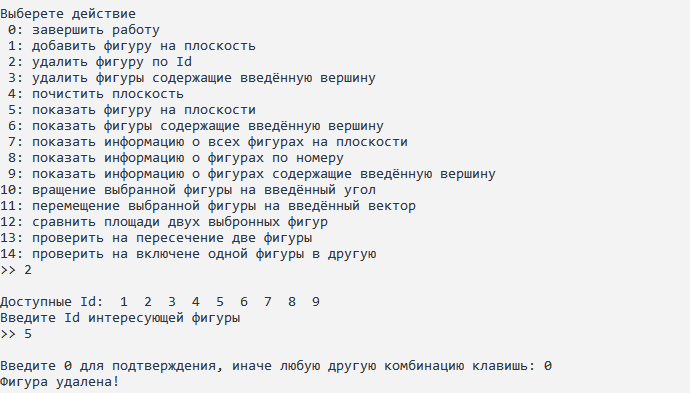
5 -5 -7 23 35 52 57 23 6 53 -50

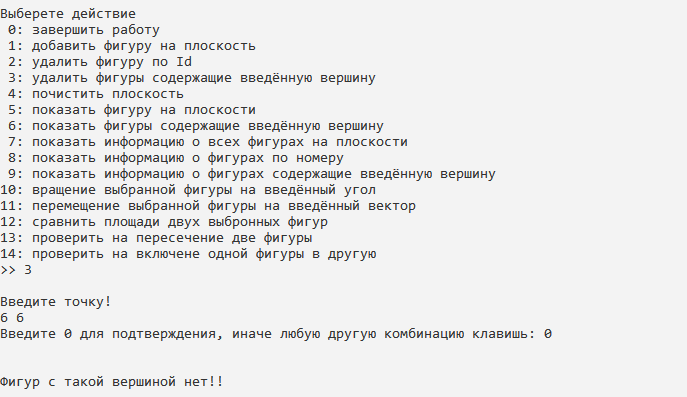
3 534 -63 37 346 3468 -52

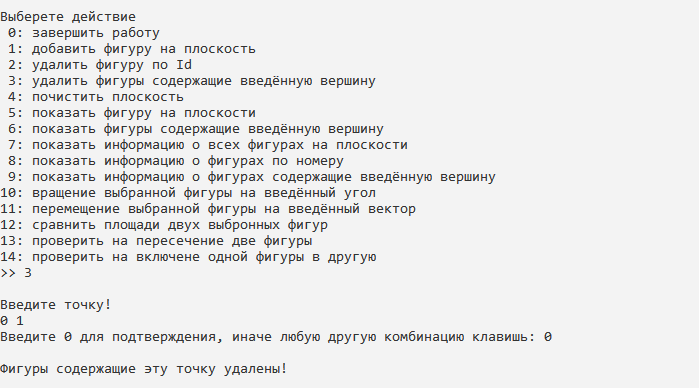
Результаты программы:

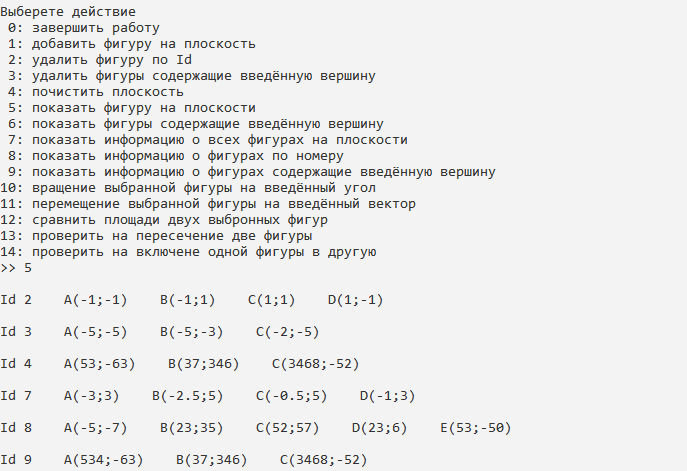


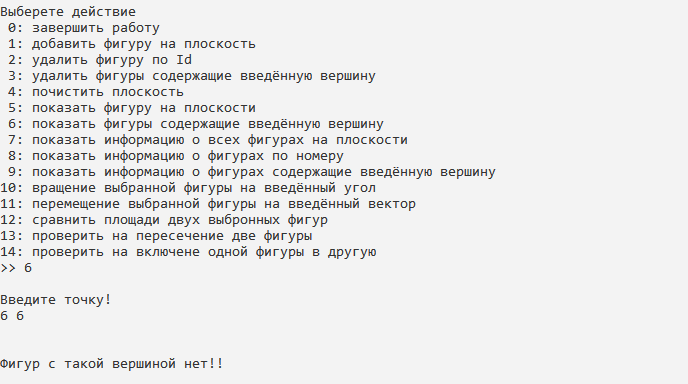


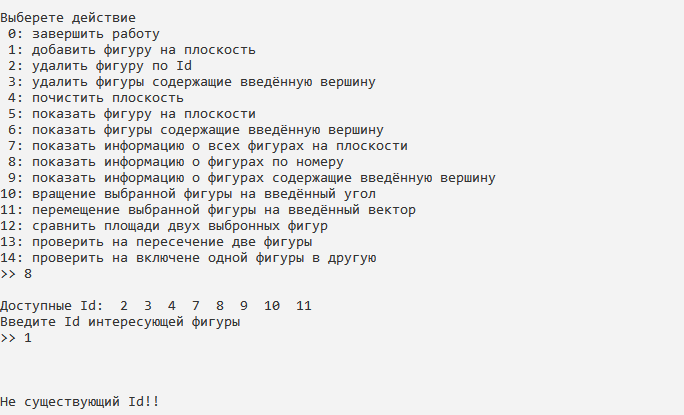


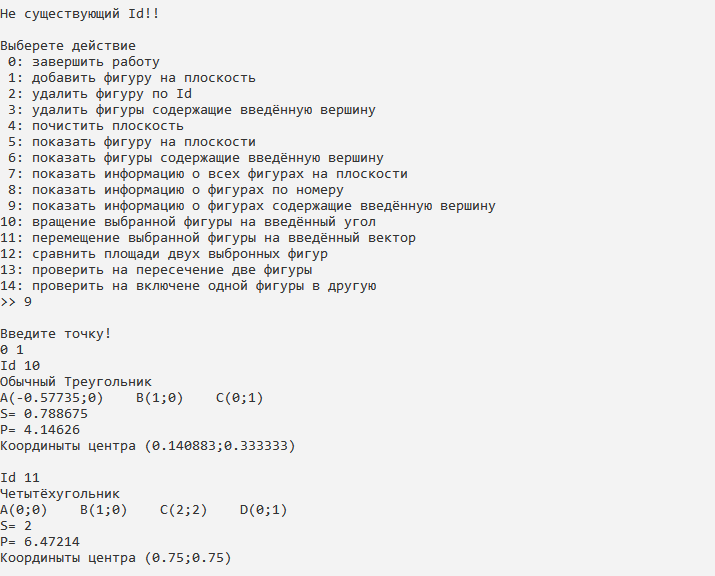


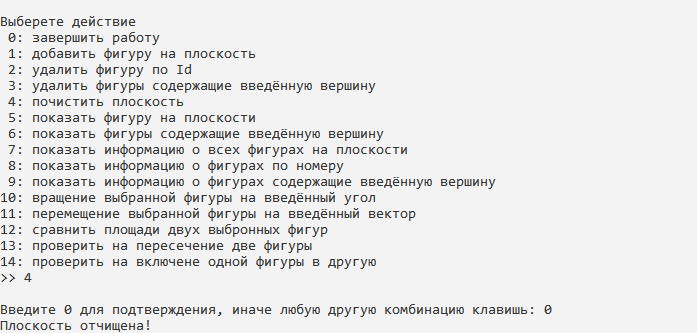


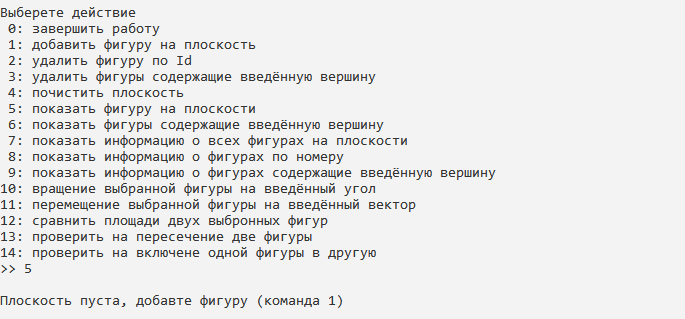


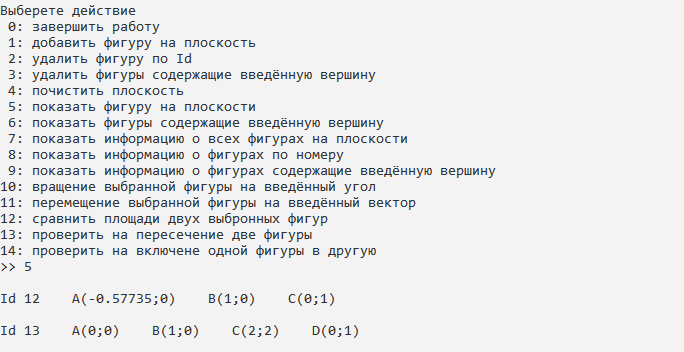


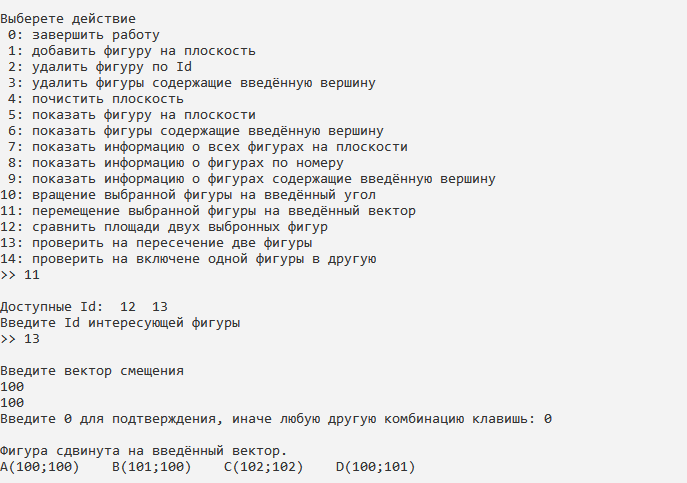


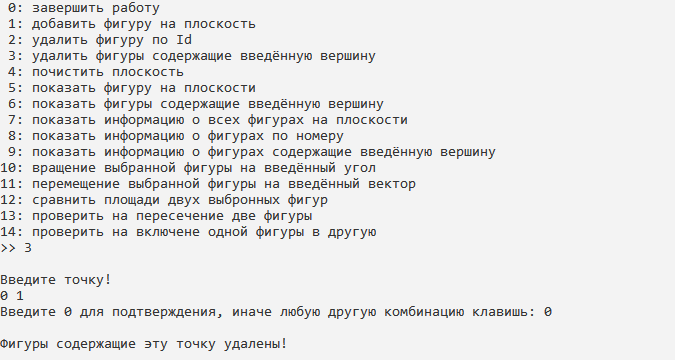


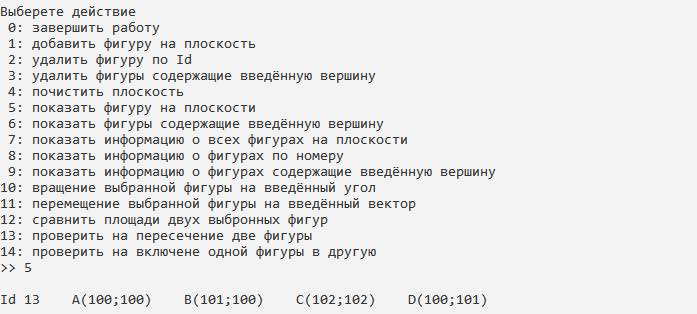


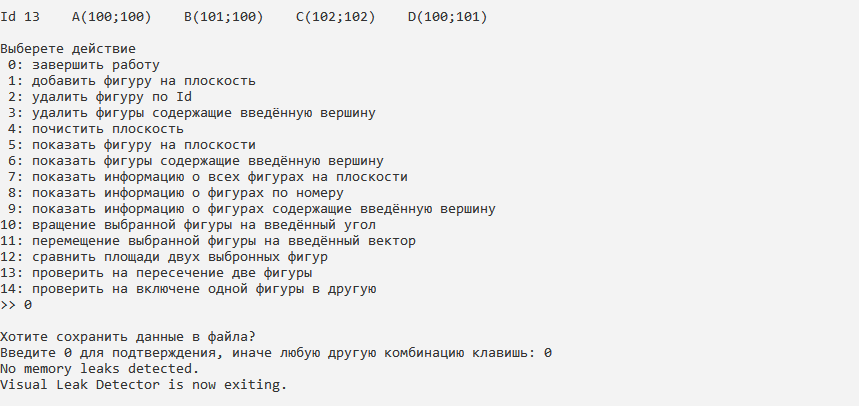














Вывод: был ознакомлен с классами-контейнерами библиотеки STL, выявил преимущества и недостатки использования конкретных контейнеров.