МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего образования

**«Нижегородский государственный университет им. Н.И. Лобачевского»**

**Национальный исследовательский университет**

**Институт информационных технологий, математики и механики**

**Кафедра математического обеспечения и суперкомпьютерных технологий**

**ЛАБОРАТОРНАЯ РАБОТА №12**

**«Структура хранения геометрических объектов.**

**Плексы»**

**Выполнил:** студент группы 381706-2

Жбанова Надежда Сергеевна

\_\_\_\_\_\_\_\_\_\_\_ Подпись

**Руководитель:**

Ассистент кафедры МОСТ

Лебедев Илья Геннадьевич

\_\_\_\_\_\_\_\_\_\_\_ Подпись

Нижний Новгород

2019

Содержание

[1.Введение 3](#_Toc9863400)

[2. Цели и задачи 4](#_Toc9863401)

[2.1. Используемые инструменты 4](#_Toc9863402)

[3. Руководство пользователя 5](#_Toc9863403)

[4. Руководство программиста 7](#_Toc9863404)

[4.1. Описание структуры программы 7](#_Toc9863405)

[4.2. Описание функций и процедур, их алгоритмов 8](#_Toc9863406)

[5. Заключение 11](#_Toc9863407)

[6. Литература 12](#_Toc9863408)

[7. Приложения 13](#_Toc9863409)

[7.1. Приложение 1:Класс TExсeption 13](#_Toc9863410)

[7.2. Приложение 2:Класс TPlex 13](#_Toc9863411)

[7.3. Приложение 3:Класс TCircle 16](#_Toc9863412)

[7.4. Приложение 4:Класс TLine 17](#_Toc9863413)

[7.5. Приложение 5:Класс TPoint 20](#_Toc9863414)

[7.6. Приложение 6:Код программы тестирования 21](#_Toc9863415)

[7.7. Приложение 7:Тесты для класса 22](#_Toc9863416)

# 1.Введение

В данной лабораторной работе будет реализована структура хранения и обработки сложных геометрических объектов на основе простейших, таких как линия и точка. Эта актуальная во всех смыслах задача будет решаться с использованием класса Плекс, так как потребуется хранить геометрический объект с неизвестным количеством углов или перегибов.

Эта структура данных будет хранить указатель либо на какую-либо точку ломаной линии, либо указатель на другой плекс, и таким образом, динамически сможет добавлять новые элементы в уже существующий объект класс. Получается бинарное дерево, для обхода которого будем использовать рекурсивный метод с динамическим преобразованием типов.

# 2. Цели и задачи

В рамках лабораторной работы ставится задача эффективной реализации структуры хранения нетипичных геометрических объектов и редактировании чертежей образуемых из ограниченного набора различных геометрических элементов (точек, линий).

В процессе выполнения лабораторной работы требуется использовать систему контроля версий [Git](https://git-scm.com/book/ru/v2) и фрэймворк для разработки автоматических тестов [Google Test](https://github.com/google/googletest).

Выполнение работы предполагает решение следующих задач:

1. Реализация классов простейших геометрических объектов TCircle, TLine и TPoint.
2. Реализация простейших операций с плексами, таких как создание плекса, добавление линии и вывод плекса на экран.
3. Реализация класса для обработки исключений– TException, которые могут возникнуть при выполнении различных операций, согласно заданному интерфейсу.
4. Обеспечение работоспособности тестов и примера использования.
5. Модификация примера использования в тестовое приложение.

## 2.1. Используемые инструменты

* Система контроля версий [Git](https://git-scm.com/book/ru/v2). Рекомендуется использовать один из следующих клиентов на выбор студента:
  + [Git](https://git-scm.com/downloads)
  + [GitHub Desktop](https://desktop.github.com/)
* Фреймворк для написания автоматических тестов [Google Test](https://github.com/google/googletest).
* Среда разработки Microsoft Visual Studio (2008 или старше).
* Опционально. Утилита [CMake](http://www.cmake.org/) для генерации проектов по сборке исходных кодов. Может быть использована для генерации решения для среды разработки, отличной от Microsoft Visual Studio 2008 или 2010.

# 3. Руководство пользователя

Запускаем программу sample\_tplex.cpp (Рис.1,2). Пользователю предлагается ввести координаты нескольких точек будущего плекса.

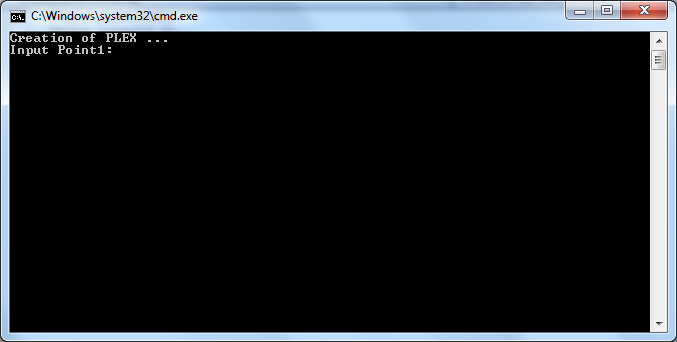


Рис.1. Пользовательский ввод.

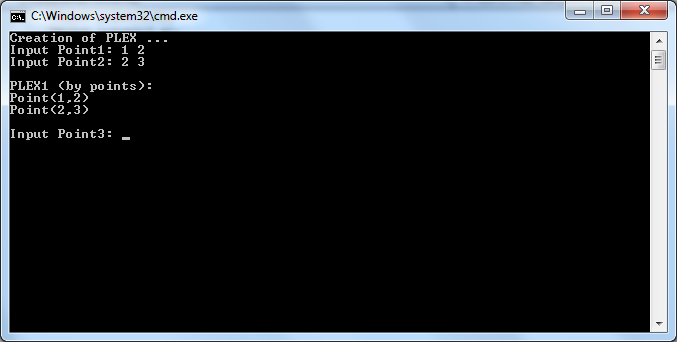


Рис.2. Пользовательский ввод.

Программа выведет созданные Плексы№1 и №2, а также создаст копию Плекса№ 2. (Рис.3):

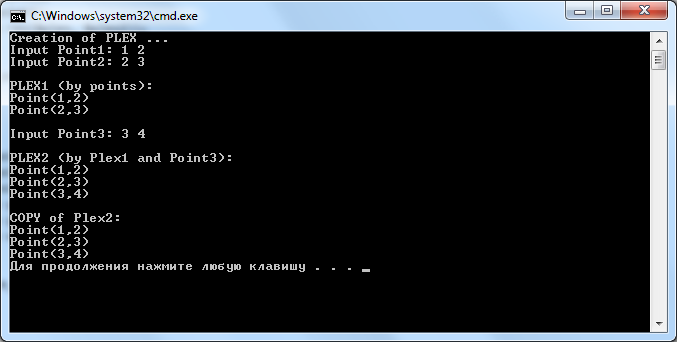


Рис.3.Пример работы простых операций с плексами.

Алгоритм можно повторить несколько раз.

# 4. Руководство программиста

Разработка системы вычисления проводились в среде “Microsoft Visual Studio 2010”.

В данной работе будет использовано несколько классов:

* Класс “Точка” (TPoint).
* Класс “Линия” (TLine).
* Класс “Круг” (TCircle).
* Класс “Плекс” (TPlex).
* Класс исключения (TExсeption).
* Базовый класс Стека.

## 4.1. Описание структуры программы

Модульная структура программы:

1. tplex.h – модуль с классом TPlex, в котором определен интерфейс класса Плекс и реализация его методов.
2. tcircle.h - модуль с классом TCircle, в котором определен интерфейс класса Круг и реализация его методов.
3. tline.h - модуль с классом TLine, в котором определен интерфейс класса Линия и реализация его методов.
4. tpoint.h - модуль с классом TPoint, в котором определен интерфейс класса Точка и реализация его методов.
5. exсeption.h – модуль с классом исключения TExсeption.
6. sample\_tplex.cpp– модуль программы тестирования, с которым работает пользователь.
7. test\_main.cpp, test\_tplex.cpp – модуль с функциями тестирования для созданных классов. Содержат 5 тестов для класса TPlex, 10 – для класса TCircle, 10 - для класса TLine и 10 - для класса TPoint.

## 4.2. Описание функций и процедур, их алгоритмов

Рассмотрим реализацию методов класса TPlex:

class TPlex: public TPoint

TPoint \*left - указатель на левую ветку плекса.

TPoint \*right - указатель на правую ветку плекса.

1. virtual TPoint\* Show(TPlex\* p) – (защищенный метод).

2. TPlex() – конструктор по умолчанию.

3. TPlex(TPoint\* a, TPoint\* b) – конструктор с параметрами. Принимает два указателя на точки.

4. TPlex(const TPlex& A) – конструктор копирования. Принимает ссылку на объект класса TPlex.

5. ~TPlex() – деструктор.

6. TPoint GetR() – метод, возвращающий указатель на правую точку.

7. TPoint GetL() - метод, возвращающий указатель на левую точку.

8. void Del() – метод, удаляющий плекс.

9. TPlex& operator +=(TLine& A) – перегрузка оператора “+=”. Принимает ссылку на объект класса линии и добавляет ее при условии, что одна из точек прямой уже принадлежит плексу.

10. virtual void Show() – метод, который выводит все точки плекса на экран.

Рассмотрим реализацию методов класса TCircle:

class TCircle

TPoint\* center – указатель на точку центра круга.

double radius – значение радиуса круга.

1. TCircle() - конструктор по умолчанию.

2. TCircle(const TPoint\* A, double rad) - конструктор с параметрами.

Принимает указатель на точку – центр и радиус круга.

3. TCircle(const int \_x, const int \_y, const double R) - конструктор с параметрами.

Принимает значения координат точки и радиус.

4. TCircle(const TCircle& A) - конструктор копирования. Принимает ссылку на объект класса TCircle.

5. ~TCircle() - деструктор.

6. void SetX(const int param) - метод, устанавливающий координату центра x.

7. void SetY(const int param) - метод, устанавливающий координату центра y.

8. void SetRadius(const double rad) - метод, устанавливающий радиус круга.

9. double GetX() - метод, возвращающий координату центра x.

10. double GetY() - метод, возвращающий координату центра y.

11. double GetRadius() - метод, возвращающий радиус круга.

12. void Show() - метод вывода параметров круга на экран.

Рассмотрим реализацию методов класса TLine:

class TLine

TPoint\* A – указатель на первую точку.

TPoint\* B - указатель на вторую точку.

1. TLine(void) - конструктор по умолчанию.

2. TLine(TPoint\* \_A, TPoint\* \_B) - конструктор с параметрами. Принимает два указателя на точки.

3. TLine(const TLine& line) - конструктор копирования. Принимает ссылку на объект класса TLine.

4. ~TLine() - деструктор.

5. void SetPointA(TPoint\* \_A) – метод, устанавливающий указатель на первую точку.

6. void SetPointB(TPoint\* \_B) - метод, устанавливающий указатель на вторую точку.

7. TPoint\* GetPointA() – метод, возвращающий указатель на первую точку.

8. TPoint\* GetPointB() - метод, возвращающий указатель на вторую точку.

9. int GetX1() - метод, возвращающий координату x первой точки.

10. int GetY1() - метод, возвращающий координату y первой точки.

11. int GetX2() - метод, возвращающий координату x второй точки.

12. int GetY2() - метод, возвращающий координату y второй точки.

13. void SetX1(int x1) - метод, устанавливающий координату x первой точки.

14. void SetY1(int y1) - метод, устанавливающий координату y первой точки.

15. void SetX2(int x2) - метод, устанавливающий координату x второй точки.

16. void SetY2(int y2) - метод, устанавливающий координату y второй точки.

17. double GetLength() const – метод, возвращающий длину отрезка.

18. bool operator==(const TLine& line) – перегрузка оператора сравнения для отрезков.

19. bool operator!=(const TLine& line) - перегрузка оператора сравнения для отрезков.

20. virtual void Show() – метод вывода координат отрезка на экран.

21. friend std::ostream &operator<<(std::ostream &ostr, TLine line) – дружественный метод, перегрузка оператора потокового вывода линии.

Рассмотрим реализацию методов класса TPoint:

class TPoint

int x, y – координаты точки.

1. TPoint(int \_x = 0, int \_y = 0) - конструктор класса, принимающий координаты точки. По умолчанию обе координаты устанавливаются в ноль.

2. TPoint(const TPoint& A) - конструктор копирования. Принимает ссылку на объект класса TPoint.

3. ~TPoint() - деструктор.

4. void SetX(int \_x) - метод, устанавливающий координату x.

5. void SetY(int \_y) - метод, устанавливающий координату y.

6. int GetX() - метод, возвращающий координату x.

7. int GetY() - метод, возвращающий координату y.

8. bool operator==(const TPoint& A) - перегрузка оператора сравнения для точек.

9. bool operator!=(const TPoint& A) - перегрузка оператора сравнения для точек.

10. virtual void Show() - метод вывода координат точки на экран.

11. friend std::ostream &operator<<(std::ostream &ostr, TPoint point) - дружественный метод, перегрузка оператора потокового вывода точки.

# 

# 5. Заключение

В результате лабораторной работы был разработан класс Плекса и классы основных простейших фигур Точки, Линии и Круга, а также освоены такие инструменты разработки программного обеспечения, как система контроля версий [Git](https://git-scm.com/book/ru/v2) и фрэймворк для разработки автоматических тестов [Google Test](https://github.com/google/googletest).

Созданный классы были успешно протестированы с использованием Google Tests.

# 6. Литература

1. Лабораторный практикум: Учебно-методическое пособие / Мееров И.Б. [и др.] – Нижний Новгород: Нижегородский госуниверситет , 2017. – 105с.
2. Тестирование с использованием Google Test

(http://neerc.ifmo.ru/wiki/index.php?title=Тестирование\_с\_использованием\_Google\_Test#.D0.A4.D1.83.D0.BD.D0.BA.D1.86.D0.B8.D1.8F\_main.28.29)

1. Касперски К. Техника оптимизации программ. Эффективное использование памяти. — СПб.: БХВ-Петербург, 2003. — 464 с.: ил.

# 7. Приложения

## 7.1. Приложение 1:Класс TExсeption

|  |
| --- |
| **exception.h** |
| #pragma once  #include <iostream>  #include <string>  class TException  {  private:  std::string str;  public:  TException(std::string \_str);  void Show();  };  TException::TException(std::string \_str) : str(\_str) {}  void TException::Show()  {  std::cout << "\nWarning! \nMessage: " << str << std::endl;  } |

## 7.2. Приложение 2:Класс TPlex

|  |
| --- |
| **tplex.h** |
| #pragma once  #include <iostream>  #include <stack>  #include "exception.h"  #include "tpoint.h"  #include "tline.h"  class TPlex: public TPoint  {  protected:  TPoint \*left;  TPoint \*right;  virtual TPoint\* Show(TPlex\* p); //not used  public:  TPlex();  TPlex(TPoint\* a, TPoint\* b);  TPlex(const TPlex& A);  ~TPlex();  TPoint GetR();  TPoint GetL();  void Del();  TPlex& operator +=(TLine& A);  virtual void Show();  };  //----------------------------------------------------------------------  TPlex::TPlex()  {  right = NULL;  left = NULL;  }  //----------------------------------------------------------------------  TPlex::TPlex(TPoint\* A, TPoint\* B)  {  TPlex\* memory = dynamic\_cast<TPlex\*>(A);  TPlex\* memory2 = dynamic\_cast<TPlex\*>(B);  if (memory != NULL)  left = new TPlex(\*memory);  else  left = new TPoint(\*A);  if (memory2 != NULL)  right = new TPlex(\*memory2);  else  right = new TPoint(\*B);  }  //----------------------------------------------------------------------  TPlex::TPlex(const TPlex& plex)  {  TPlex\* memory = dynamic\_cast<TPlex\*>(plex.left);  TPlex\* memory2 = dynamic\_cast<TPlex\*>(plex.right);  if (memory != NULL)  left = new TPlex(\*memory);  else  left = new TPoint(\*plex.left);  if (memory2 != NULL)  right = new TPlex(\*memory2);  else  right = new TPoint(\*plex.right);  }  //----------------------------------------------------------------------  TPlex::~TPlex()  {  Del();  }  //----------------------------------------------------------------------  TPoint TPlex::GetR()  {  return \*right;  }  //----------------------------------------------------------------------  TPoint TPlex::GetL()  {  return \*left;  }  //----------------------------------------------------------------------  void TPlex::Del()  {  TPlex\* tmp1 = dynamic\_cast<TPlex\*>(left);  TPlex\* tmp2 = dynamic\_cast<TPlex\*>(right);  if (tmp1 != NULL)  tmp1->Del(); //recursion  else  delete left;  if (tmp2 != NULL)  tmp2->Del(); //recursion  else  delete right;  }  //----------------------------------------------------------------------  TPlex& TPlex::operator +=(TLine& A)  {  if (right == NULL && left == NULL)  {  left = A.GetPointA();  right = A.GetPointB();  }  else  {  std::stack<TPoint\*> memory;  std::stack<TPoint\*> memory2;  memory.push(left);  memory.push(right);  memory2.push(this);  memory2.push(this);  bool flag = true;  while (flag == true)  {  TPoint\* tmp = memory.top();  TPlex\* pl = dynamic\_cast<TPlex\*>(memory2.top());  if (\*tmp != \*A.GetPointA() && \*tmp != \*A.GetPointB())  {  TPlex\* px = dynamic\_cast<TPlex\*>(tmp);  if (px != NULL)  {  memory.push(px->left);  memory.push(px->right);  memory2.push(px);  memory2.push(px);  }  }  else if (\*tmp == \*A.GetPointA())  {  flag = false;  if (\*tmp == \*pl->left)  {  pl->left = new TPlex(A.GetPointB(), A.GetPointA());  }  else  {  pl->right = new TPlex(A.GetPointB(), A.GetPointA());  }  }  else if (\*tmp == \*A.GetPointB())  {  flag = false;  if (\*tmp == \*pl->left)  pl->left = new TPlex(A.GetPointA(), A.GetPointB());  else  pl->right = new TPlex(A.GetPointA(), A.GetPointB());  }  }  }  return \*this;  }  //----------------------------------------------------------------------  TPoint\* TPlex::Show(TPlex\* p)  {  TPlex\* lp = dynamic\_cast<TPlex\*>(p -> left);  TPlex\* rp = dynamic\_cast<TPlex\*>(p -> right);  TPoint\* tmp\_l;  TPoint\* tmp\_r;  if(lp == NULL)  {  tmp\_l = p -> left;  }  else  tmp\_r = Show(lp); //recursion  if(rp == NULL)  {  tmp\_r = p -> right;  }  else  tmp\_r = Show(rp); //recursion  std::cout << "Line: " << \*tmp\_l << \* tmp\_r <<std::endl;  return tmp\_r;  }  //----------------------------------------------------------------------  void TPlex::Show()  {  TPlex\* memory = dynamic\_cast<TPlex\*>(left);  TPlex\* memory2 = dynamic\_cast<TPlex\*>(right);  if (memory != NULL)  memory->Show();  else  left->Show();  if (memory2 != NULL)  memory2->Show();  else  right->Show();  } |

## 7.3. Приложение 3:Класс TCircle

|  |
| --- |
| **tcircle.h** |
| #pragma once  #include "tpoint.h"  class TCircle  {  protected:  TPoint\* center;  double radius;  public:  TCircle();  TCircle(const TPoint\* A, double rad);  TCircle(const int \_x, const int \_y, const double R);  TCircle(const TCircle& A);  ~TCircle();  void SetX(const int param);  void SetY(const int param);  void SetRadius(const double rad);  double GetX() { return center->GetX(); }  double GetY() { return center->GetY(); }  double GetRadius() { return radius; }  void Show();  };  //----------------------------------------------------------------------  TCircle::TCircle() : radius(0)  {  center = new TPoint(0,0);  }  //----------------------------------------------------------------------  TCircle::TCircle(const TPoint\* A, double R)  {  if (R < 0)  throw TException("Error");  center = new TPoint(\*A);  radius = R;  }  //----------------------------------------------------------------------  TCircle::TCircle(const int \_x, const int \_y, const double R)  {  if (R < 0)  throw TException("Error");  center = new TPoint(\_x, \_y);  radius = R;  }  //----------------------------------------------------------------------  TCircle::TCircle(const TCircle& A)  {  center = new TPoint(\*A.center);  radius = A.radius;  }  //----------------------------------------------------------------------  TCircle::~TCircle()  {  delete center;  radius = 0;  }  //----------------------------------------------------------------------  void TCircle::SetX(const int param)  {  center->SetX(param);  }  //----------------------------------------------------------------------  void TCircle::SetY(const int param)  {  center->SetY(param);  }  //----------------------------------------------------------------------  void TCircle::SetRadius(const double R)  {  if (R < 0)  throw TException("Error");  radius = R;  }  //----------------------------------------------------------------------  void TCircle::Show()  {  std::cout << "Center( ";  center->Show();  std::cout << " )";  std::cout << "Radius ( " << radius << " )" << std::endl;  } |

## 7.4. Приложение 4:Класс TLine

|  |
| --- |
| **tline.h** |
| #pragma once  #include <cmath>  #include <iostream>  #include "tpoint.h"  class TLine  {  protected:  TPoint\* A; //first point  TPoint\* B; //second point  public:  TLine(void);  TLine(TPoint\* \_A, TPoint\* \_B);  TLine(const TLine& line);  ~TLine();  void SetPointA(TPoint\* \_A);  void SetPointB(TPoint\* \_B);  TPoint\* GetPointA();  TPoint\* GetPointB();  int GetX1();  int GetY1();  int GetX2();  int GetY2();  void SetX1(int x1);  void SetY1(int y1);  void SetX2(int x2);  void SetY2(int y2);  double GetLength() const;  bool operator==(const TLine& line);  bool operator!=(const TLine& line);  virtual void Show();  friend std::ostream &operator<<(std::ostream &ostr, TLine line);  };  //----------------------------------------------------------------------  TLine::TLine(void)  {  A = new TPoint;  B = new TPoint;  }  //----------------------------------------------------------------------  TLine::TLine(TPoint\* \_A, TPoint\* \_B)  {  A = new TPoint(\*\_A);  B = new TPoint(\*\_B);  }  //----------------------------------------------------------------------  TLine::TLine(const TLine& line)  {  A = new TPoint(\*line.A);  B = new TPoint(\*line.B);  }  //----------------------------------------------------------------------  TLine::~TLine()  {  delete A;  delete B;  }  //----------------------------------------------------------------------  void TLine::SetPointA(TPoint\* \_A)  {  delete A;  A = new TPoint(\*\_A);  }  //----------------------------------------------------------------------  void TLine::SetPointB(TPoint\* \_B)  {  delete B;  B = new TPoint(\*\_B);  }  //----------------------------------------------------------------------  TPoint\* TLine::GetPointA()  {  return A;  }  //----------------------------------------------------------------------  TPoint\* TLine::GetPointB()  {  return B;  }  //----------------------------------------------------------------------  int TLine::GetX1()  {  return A->GetX();  }  //----------------------------------------------------------------------  int TLine::GetY1()  {  return A->GetY();  }  //----------------------------------------------------------------------  int TLine::GetX2()  {  return B->GetX();  }  //----------------------------------------------------------------------  int TLine::GetY2()  {  return B->GetY();  }  //----------------------------------------------------------------------  void TLine::SetX1(int x1)  {  A->SetX(x1);  }  //----------------------------------------------------------------------  void TLine::SetY1(int y1)  {  A->SetY(y1);  }  //----------------------------------------------------------------------  void TLine::SetX2(int x2)  {  B->SetX(x2);  }  //----------------------------------------------------------------------  void TLine::SetY2(int y2)  {  B->SetY(y2);  }  //----------------------------------------------------------------------  double TLine::GetLength() const  {  //Length = √(X²+Y²) = √((X2-X1)²+(Y2-Y1)²)  double first = B->GetX() - A->GetX();  double second = B->GetY()-A->GetY();  double res = sqrt(pow(first,2)+pow(second,2));  return res;  }  //----------------------------------------------------------------------  bool TLine::operator==(const TLine& line)  {  if(line.GetLength() == this->GetLength())  return true;  else  return false;  }  //----------------------------------------------------------------------  bool TLine::operator!=(const TLine& line)  {  if(line.GetLength() != this->GetLength())  return true;  else  return false;  }  //----------------------------------------------------------------------  void TLine::Show()  {  std::cout <<"Line("<< \*A <<","<< \*B <<")";  }  //----------------------------------------------------------------------  std::ostream &operator<<(std::ostream &ostr, TLine line)  {  ostr << "Line(" << \*line.A << "," << \*line.B <<")";  return ostr;  } |

## 7.5. Приложение 5:Класс TPoint

|  |
| --- |
| **tpoint.h** |
| #pragma once  #include <iostream>  class TPoint  {  protected:  int x, y;  public:  TPoint(int \_x = 0, int \_y = 0);  TPoint(const TPoint& A);  ~TPoint();  void SetX(int \_x);  void SetY(int \_y);  int GetX();  int GetY();  bool operator==(const TPoint& A);  bool operator!=(const TPoint& A);  virtual void Show();  friend std::ostream &operator<<(std::ostream &ostr, TPoint point);  };  //----------------------------------------------------------------------  TPoint::TPoint(int \_x, int \_y)  {  x = \_x;  y = \_y;  }  //----------------------------------------------------------------------  TPoint::TPoint(const TPoint& A)  {  x = A.x;  y = A.y;  }  //----------------------------------------------------------------------  TPoint::~TPoint()  {  x = 0;  y = 0;  }  //----------------------------------------------------------------------  void TPoint::SetX(int \_x)  {  x = \_x;  }  //----------------------------------------------------------------------  void TPoint::SetY(int \_y)  {  y = \_y;  }  //----------------------------------------------------------------------  int TPoint::GetX()  {  return x;  }  //----------------------------------------------------------------------  int TPoint::GetY()  {  return y;  }  //----------------------------------------------------------------------  bool TPoint::operator==(const TPoint& A)  {  if ((x == A.x) && (y == A.y))  return true;  else  return false;  }  //----------------------------------------------------------------------  bool TPoint::operator!=(const TPoint& A)  {  if ((x == A.x) && (y == A.y))  return false;  else  return true;  }  //----------------------------------------------------------------------  void TPoint::Show()  {  std::cout << "Point(" << x << ","<< y <<")\n";  }  //----------------------------------------------------------------------  std::ostream &operator<<(std::ostream &ostr, TPoint point)  {  ostr << "Point(" << point.x << ", " << point.y <<")\n";  return ostr;  } |

## 7.6. Приложение 6:Код программы тестирования

|  |
| --- |
| **sample\_tplex.cpp** |
| } #include <iostream>  #include <locale.h>  #include "tplex.h"  int main()  {  //EXAMPLE  std::cout << "Creation of PLEX ...";  int x1, x2, x3, y1, y2, y3;  std::cout << "\nInput Point1: ";  std::cin >> x1 >> y1;  TPoint P(x1, y1);  std::cout << "Input Point2: ";  std::cin >> x2 >> y2;  TPoint P2(x2, y2);  std::cout << "\nPLEX1 (by points):\n";  TPlex plex(&P, &P2);  plex.Show();  std::cout << "\nInput Point3: ";  std::cin >> x3 >> y3;  TPoint P3(x3, y3);  std::cout << "\nPLEX2 (by Plex1 and Point3):\n";  TPlex plex2(&plex, &P3);  plex2.Show();  std::cout << "\nCOPY of Plex2:\n";  TPlex plex3(plex2);  plex3.Show();  return 0;  } |

## 7.7. Приложение 7:Тесты для класса

|  |
| --- |
| **test\_main.cpp** |
| #include <gtest.h>  int main(int argc, char \*\*argv)  {  ::testing::InitGoogleTest(&argc, argv);  return RUN\_ALL\_TESTS();  } |

|  |
| --- |
| **test\_tplex.cpp** |
| #include "tplex.h"  #include <gtest.h>  //TESTS FOR CLASS TPoint  TEST(TPoint, can\_create\_point\_without\_parametrs)  {  ASSERT\_NO\_THROW(TPoint point);  }  TEST(TPoint, point\_was\_created\_without\_parametrs\_has\_nulls)  {  TPoint point;  EXPECT\_EQ(point.GetX(), 0);  EXPECT\_EQ(point.GetY(), 0);  }  TEST(TPoint, can\_create\_point\_with\_parametrs)  {  ASSERT\_NO\_THROW(TPoint point(3,5));  }  TEST(TPoint, can\_get\_x\_and\_y)  {  TPoint point(3,5);  EXPECT\_EQ(point.GetX(), 3);  EXPECT\_EQ(point.GetY(), 5);  }  TEST(TPoint, can\_set\_x\_and\_y)  {  TPoint point(3,5);  point.SetX(7);  point.SetY(8);  EXPECT\_EQ(point.GetX(), 7);  EXPECT\_EQ(point.GetY(), 8);  }  TEST(TPoint, can\_create\_copy\_of\_point)  {  TPoint point(3,5);  ASSERT\_NO\_THROW(TPoint point2(point));  TPoint point2(point);  EXPECT\_EQ(point2.GetX(), 3);  EXPECT\_EQ(point2.GetY(), 5);  }  TEST(TPoint, compare\_equal\_points\_return\_true)  {  TPoint point(3,5);  TPoint point2(3,5);  EXPECT\_TRUE(point == point2);  }  TEST(TPoint, compare\_non\_equal\_points\_return\_false)  {  TPoint point(3,5);  TPoint point2(9,1);  EXPECT\_FALSE(point == point2);  }  TEST(TPoint, can\_return\_true\_if\_points\_are\_not\_equil)  {  TPoint point(3,5);  TPoint point2(9,1);  EXPECT\_TRUE(point != point2);  }  TEST(TPoint, can\_return\_false\_if\_points\_are\_equil)  {  TPoint point(3,5);  TPoint point2(3,5);  EXPECT\_FALSE(point != point2);  }  //TESTS FOR CLASS TLine  TEST(TLine, can\_create\_line)  {  ASSERT\_NO\_THROW(TLine line);  }  TEST(TLine, can\_create\_line\_by\_points)  {  TPoint point(3,5);  TPoint point2(9,1);  ASSERT\_NO\_THROW(TLine line(&point, &point2));  }  TEST(TLine, can\_create\_copy\_of\_line)  {  TPoint point(3,5);  TPoint point2(9,1);  TLine line(&point, &point2);  ASSERT\_NO\_THROW(TLine line2(line));  TLine line2(line);  EXPECT\_EQ(line2.GetX1(), 3);  EXPECT\_EQ(line2.GetY1(), 5);  EXPECT\_EQ(line2.GetX2(), 9);  EXPECT\_EQ(line2.GetY2(), 1);  }  TEST(TLine, can\_get\_point\_A\_and\_point\_B)  {  TPoint point(3,5);  TPoint point2(9,1);  TLine line(&point, &point2);  EXPECT\_TRUE(\*line.GetPointA() == point);  EXPECT\_TRUE(\*line.GetPointB() == point2);  }  TEST(TLine, can\_set\_point\_A\_and\_point\_B)  {  TLine line;  TPoint point(3,5);  TPoint point2(9,1);  ASSERT\_NO\_THROW(line.SetPointA(&point));  ASSERT\_NO\_THROW(line.SetPointB(&point2));  line.SetPointA(&point);  line.SetPointB(&point2);  EXPECT\_EQ(line.GetX1(), 3);  EXPECT\_EQ(line.GetY1(), 5);  EXPECT\_EQ(line.GetX2(), 9);  EXPECT\_EQ(line.GetY2(), 1);  }  TEST(TLine, can\_get\_x\_and\_y\_of\_point\_A\_and\_point\_B)  {  TPoint point(3,5);  TPoint point2(9,1);  TLine line(&point, &point2);  EXPECT\_EQ(line.GetX1(), 3);  EXPECT\_EQ(line.GetY1(), 5);  EXPECT\_EQ(line.GetX2(), 9);  EXPECT\_EQ(line.GetY2(), 1);  }  TEST(TLine, can\_set\_x\_and\_y\_of\_point\_A\_and\_point\_B)  {  TLine line;  line.SetX1(3);  line.SetY1(5);  line.SetX2(9);  line.SetY2(1);  EXPECT\_EQ(line.GetX1(), 3);  EXPECT\_EQ(line.GetY1(), 5);  EXPECT\_EQ(line.GetX2(), 9);  EXPECT\_EQ(line.GetY2(), 1);  }  TEST(TLine, can\_get\_length\_of\_line)  {  TPoint point(1,1);  TPoint point2(4,5);  TLine line(&point, &point2);  EXPECT\_EQ(line.GetLength(), 5);  }  TEST(TLine, compare\_equal\_lines)  {  TPoint point(3,5);  TPoint point2(1,5);  TLine line(&point, &point2);  TLine line2(line);  EXPECT\_TRUE(line == line2);  EXPECT\_FALSE(line != line2);  }  TEST(TLine, compare\_non\_equal\_lines)  {  TPoint point(3,5);  TPoint point2(3,6);  TPoint point3(7,1);  TLine line(&point, &point2);  TLine line2(&point2, &point3);  EXPECT\_TRUE(line != line2);  EXPECT\_FALSE(line == line2);  }  //TESTS FOR CLASS TCircle  TEST(TCircle, can\_create\_circle\_by\_three\_ways)  {  ASSERT\_NO\_THROW(TCircle circle1);  TPoint point(1, 1);  ASSERT\_NO\_THROW(TCircle circle2(&point, 1));  ASSERT\_NO\_THROW(TCircle circle3(1, 1, 1));  }  TEST(TCircle, can\_not\_create\_circle\_with\_negative\_radius)  {  TPoint point(1, 1);  ASSERT\_ANY\_THROW(TCircle circle1(&point, -1));  ASSERT\_ANY\_THROW(TCircle circle2(1, 1, -1));  }  TEST(TCircle, can\_create\_copy\_of\_circle)  {  TCircle circle(1, 1, 1);  ASSERT\_NO\_THROW(TCircle circle1(circle));  }  TEST(TCircle, can\_get\_X)  {  TCircle circle(1, 1, 1);  EXPECT\_EQ(circle.GetX(), 1);  }  TEST(TCircle, can\_get\_Y)  {  TCircle circle(1, 1, 1);  EXPECT\_EQ(circle.GetY(), 1);  }  TEST(TCircle, can\_get\_radius)  {  TCircle circle(1, 1, 1);  EXPECT\_EQ(circle.GetRadius(), 1);  }  TEST(TCircle, can\_set\_X)  {  TCircle circle(1, 1, 1);  circle.SetX(2);  EXPECT\_EQ(circle.GetX(), 2);  }  TEST(TCircle, can\_set\_Y)  {  TCircle circle(1, 1, 1);  circle.SetY(2);  EXPECT\_EQ(circle.GetY(), 2);  }  TEST(TCircle, can\_set\_R\_radius)  {  TCircle circle(1, 1, 1);  circle.SetRadius(2);  EXPECT\_EQ(circle.GetRadius(), 2);  }  TEST(TCircle, can\_not\_set\_negative\_radius)  {  TCircle circle(1, 1, 1);  ASSERT\_ANY\_THROW(circle.SetRadius(-1));  }  //TESTS FOR CLASS TPlex  TEST(TPlex, can\_create\_plex)  {  ASSERT\_NO\_THROW(TPlex plex);  }  TEST(TPlex, can\_create\_plex\_by\_points)  {  TPoint point(3,5);  TPoint point2(7,4);  ASSERT\_NO\_THROW(TPlex plex(&point, &point2));  }  TEST(TPlex, can\_create\_plex\_by\_plexes)  {  TPoint point(3,5);  TPoint point2(7,4);  TPoint point3(8,3);  TPlex plex(&point, &point2);  TPlex plex2(&plex, &point3);  ASSERT\_NO\_THROW(TPlex res(&plex, &plex2));  }  TEST(TPlex, can\_create\_copy\_of\_plex)  {  TPoint point(3,5);  TPoint point2(7,4);  TPoint point3(8,3);  TPlex plex(&point, &point2);  TPlex plex2(&plex, &point3);  TPlex win(&plex, &plex2);  ASSERT\_NO\_THROW(TPlex plex2(plex));  ASSERT\_NO\_THROW(TPlex plex3(plex2));  ASSERT\_NO\_THROW(TPlex win2(win));  }  TEST(TPlex, can\_add\_line\_to\_plex)  {  TPoint point(3,5);  TPoint point2(7,4);  TPoint point3(8,3);  TPoint point4(3,3);  TPlex plex(&point, &point2);  TPlex plex1(&plex, &point3);  TLine line;  line.SetPointA(&point4);  line.SetPointB(&point3);  ASSERT\_NO\_THROW(plex1 += line);  } |