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**

**ЛАБОРАТОРНАЯ РАБОТА №13**

**«Плексы»**

**Выполнил:** студент группы 381903-3

Стеценко Данил Михайлович

**Проверил:**

Ассистент кафедры МОСТ

Лебедев Илья Геннадьевич

Нижний Новгород

2020

Содержание

[1.Введение 3](#_Toc1336360)

[2. Цели и задачи 4](#_Toc1336361)

[2.1. Используемые инструменты 4](#_Toc1336362)

[3. Руководство пользователя 6](#_Toc1336363)

[4. Руководство программиста 7](#_Toc1336364)

[4.1. Описание структуры программы 7](#_Toc1336365)

[4.2. Описание функций и процедур, их алгоритмов 7](#_Toc1336366)

[5. Эксперименты 11](#_Toc1336367)

[6. Заключение 12](#_Toc1336368)

[7. Литература 13](#_Toc1336369)

[8. Приложения 14](#_Toc1336370)

# 1.Введение

Тема лабораторной работы относится к важной области задач обработки графических данных, потребность в анализе которых встречается при рассмотрении многих научно-технических и производственных задач. При построении различных программных систем общего и специального назначения (таких, например, как системы автоматизированного проектирования) могут оказаться необходимыми средства, обеспечивающие хранение графической информации на ЭВМ, возможность редактирования этой информации и использования ее в расчетах, а также получение изображений на экране дисплея или на бумаге.

Лабораторная работа является введением в проблематику структурного (векторного) представления графической информации, когда изображения представляются не в виде растровых данных, а формируются в виде набора графических элементов. Такой подход позволяет существенно повысить эффективность анализа и обработки графических данных.

Дополнительный учебный эффект при выполнении лабораторной работы состоит в изучении способов представления на ЭВМ математических моделей на основе сетевых структур данных.

# 2. Цели и задачи

Для работы с плексами предлагается реализовать:

* представление чертежей в памяти ЭВМ;
* демонстрацию хранимых чертежей на экране дисплея;
* редактирование чертежей (вставку и удаление линий);
* запись (чтение) данных о чертежах в файлах

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).

## 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 main.cpp. Программа выведет 2 точки с их координатами. (Рис.1):

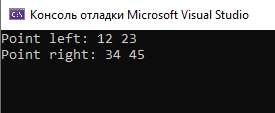


Рис.1.Пример.

# 4. Руководство программиста

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

## 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. TBase.h– модуль с классом TBase
3. TPoint.h– модуль с классом TPoint, в котором определен интерфейс класса точка и реализация его методов.
4. TLine.h– модуль с классом TLine, в котором определен интерфейс класса линия и реализация его методов.
5. test\_text.cpp, test\_main.cpp – модуль с функциями тестирования для созданных классов.

# 5.Эксперименты

Пример построения квадрата:

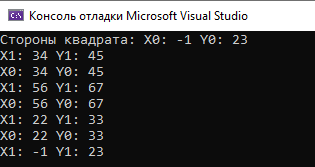


Рисунок.2. Построение квадрата

Пример построения линии:

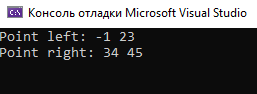


Рисунок.3. Построение линии

# 6. Заключение

В результате лабораторной работы был разработан класс полиномов, 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 класс был протестированы с использованием Google Tests, а также были проведены эксперименты для сравнения теоретической и практической сложности выполнения операций на методе класса.

# 7. 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.

# 8. Приложения

|  |
| --- |
| **TBase.h** |
| #pragma once  #include <iostream>  #include <vector>  class TBase {  public:  TBase() {};  virtual void Print() = 0;  virtual double GetVal(int i) = 0;  virtual void SetVal(double val, int i) = 0;  virtual double& operator[](char\* name) = 0;  virtual double& operator[](int i) = 0;  virtual TBase\* GetChild(int i) = 0;  virtual void SetChild(TBase\* c, int i) = 0;  virtual int GetChildCount() {  return 0;  }  virtual TBase\* PrintAll(TBase\* p) {  return this;  }  virtual TBase\* Clone() = 0;  }; |

|  |
| --- |
| **TPoint.h** |
| #pragma once  #include "TBase.h"  class TBase;  class TPoint : public TBase {  protected:  double x0, x1;  virtual TBase\* GetChild(int i);  virtual void SetChild(TBase\* c, int i);  public:  TPoint(double \_x0 = 0, double \_x1 = 0);  TPoint(const TPoint& p);  virtual void Print();  virtual double GetVal(int i);  virtual void SetVal(double val, int i);  virtual double& operator[](char\* name);  virtual double& operator[](int i);  double GetX0();  double GetX1();  void SetX0(double x);  void SetX1(double x);  virtual TBase\* Clone();  friend std::ostream& operator<<(std::ostream& out, const TPoint& point)  {  out << point.x0 << " " << point.x1;  return out;  }  };  TPoint::TPoint(double \_x0, double \_x1) {  x0 = \_x0;  x1 = \_x1;  }  TPoint::TPoint(const TPoint& p) {  x0 = p.x0;  x1 = p.x1;  }  void TPoint::Print() {  std::cout << "Point X: " << x0 << " " << "Point Y: " << x1 << "\n";  }  double TPoint::GetVal(int i) {  if (i < 0 || i > 1)  throw - 1;  if (i == 0)  return x0;  if (i == 1)  return x1;  }  void TPoint::SetVal(double val, int i) {  if (i < 0 || i > 1)  throw - 1;  if (i == 0)  x0 = val;  if (i == 1)  x1 = val;  }  double& TPoint::operator[](char\* name) {  if (name == "X0")  return x0;  else {  if (name == "X1")  return x1;  else  throw - 1;  }  }  double& TPoint::operator[](int i) {  if (i < 0 || i > 1)  throw - 1;  if (i == 0)  return x0;  if (i == 1)  return x1;  }  TBase\* TPoint::GetChild(int i) {  return 0;  }  void TPoint::SetChild(TBase\* c, int i) {  }  double TPoint::GetX0() {  return x0;  }  double TPoint::GetX1() {  return x1;  }  void TPoint::SetX0(double x) {  x0 = x;  }  void TPoint::SetX1(double x) {  x1 = x;  }  TBase\* TPoint::Clone()  {  return new TPoint(\*this);  } |

|  |
| --- |
| **Tline.h** |
| #pragma once  #include "TBase.h"  #include "TPoint.h"  #include <cstring>  class TLine : public TBase {  protected:  TPoint p1, p2;  virtual void SetChild(TBase\* c, int i);  public:  TLine() {  }  TLine(const TPoint& \_p1, const TPoint& \_p2);  TLine(const TLine& line);  virtual void Print();  TPoint& getFirst();  TPoint& getSecond();  virtual double GetVal(int i);  virtual void SetVal(double val, int i);  virtual double& operator[](char\* name);  virtual double& operator[](int i);  virtual TBase\* GetChild(int i);  virtual TBase\* Clone();  friend std::ostream& operator<<(std::ostream& out, const TLine& line)  {  out << "(" << line.p1 << "; " << line.p2 << ")";  return out;  }  };  TLine::TLine(const TPoint& \_p1, const TPoint& \_p2) {  p1 = \_p1;  p2 = \_p2;  }  TLine::TLine(const TLine& line) {  p1 = line.p1;  p2 = line.p2;  }  void TLine::Print() {  std::cout << "X0: " << p1.GetX0() << " Y0: " << p1.GetX1() << "\nX1: " << p2.GetX0() << " Y1: " << p2.GetX1() << "\n";  }  TPoint& TLine::getFirst() {  return p1;  }  TPoint& TLine::getSecond() {  return p2;  }  double TLine::GetVal(int i) {  if (i < 0 || i > 3)  throw - 1;  if (i == 0)  return p1.GetX0();  if (i == 1)  return p1.GetX1();  if (i == 2)  return p2.GetX0();  if (i == 3)  return p2.GetX1();  }  void TLine::SetVal(double val, int i) {  if (i < 0 || i > 3)  throw - 1;  if (i == 0)  p1.SetX0(val);  if (i == 1)  p1.SetX1(val);  if (i == 2)  p2.SetX0(val);  if (i == 3)  p2.SetX1(val);  }  double& TLine::operator[](char\* name) {  if (strlen(name) != 2)  throw - 1;  if (name[1] < '1' || name[1] > '4')  throw - 1;  if (name[1] - '0' == 1 || name[1] - '0' == 2) {  return p1[name[1] - '0'];  }  if (name[1] - '0' == 2 || name[1] - '0' == 3) {  return p2[name[1] - '0'];  }  }  double& TLine::operator[](int i) {  if (i < 0 || i > 3)  throw - 1;  if (i == 0)  return p1[0];  if (i == 1)  return p1[1];  if (i == 2)  return p2[0];  if (i == 3)  return p2[1];  }  TBase\* TLine::GetChild(int i) {  if (i < 0 || i > 2)  throw - 1;  if (i == 0)  return &p1;  if (i == 1)  return &p2;  }  TBase\* TLine::Clone()  {  return new TLine(\*this);  }  void TLine::SetChild(TBase\* c, int i) {  } |

|  |
| --- |
| **TPlex.h** |
| #pragma once  #include "TBase.h"  #include "TPoint.h"  #include "TLine.h"  #include "TSquare.h"  #include "TCanvas.h"  #include "Stack.h"  #include <string>  #include <iostream>  class TPlex : public TBase {  protected:  TBase\* left;  TBase\* right;  public:  TPlex();  TPlex(TPoint\* a, TPoint\* b);  TPlex(char\* s);  TPlex(const TPlex& p);  virtual void Print();  virtual double GetVal(int i);  virtual void SetVal(double val, int i);  virtual double& operator[](char\* name);  virtual double& operator[](int i);  virtual TBase\* GetChild(int i);  virtual void SetChild(TBase\* c, int i);  virtual TBase\* PrintAll();  TBase\* GetLeft();  TBase\* GetRight();  TPlex\* GetLeftPlex();  TPlex\* GetRightPlex();  void SetLeft(TBase\* x);  void SetRight(TBase\* x);  bool AddLine(TPoint\* a, TPoint\* b);  virtual int GetChildCount()  {  return 2;  }  virtual TBase\* Clone();  friend std::ostream& operator<<(std::ostream& out, const TPlex& \_plex);  };  TPlex::TPlex()  {  left = new TPoint();  right = new TPoint();  }  TPlex::TPlex(char\* s)  {  int count = 1;  int cc = 0;  int i = 0;  int j = 0;  while (s[i] != '\0')  {  if (s[i] == ' ')  count++;  i++;  cc++;  }  if (count % 2 != 0 && count < 4)  {  throw "Error";  }  double\* mass = new double[count];  std::string temp = "";  i = 0;  for (int q = 0; q < cc; q++)  {  if (s[q] != ' ' && s[q] != ',')  temp += s[q];  else if (temp != "")  {  mass[j] = atoi(temp.c\_str());  j++;  temp = "";  }  }  TPoint \*a = new TPoint(mass[0], mass[1]);  TPoint \*b = new TPoint(mass[2], mass[3]);  left = a;  right = b;  for (int q = 4; q < count; q++)  {  TPoint \*temp\_left = new TPoint(mass[q-2], mass[q-1]);  TPoint \*temp\_right = new TPoint(mass[q], mass[q + 1]);  this->AddLine(temp\_left, temp\_right);  q++;  }  }  TPlex::TPlex(TPoint\* a, TPoint\* b)  {  if (a != NULL)  left = a;  if (b != NULL)  right = b;  }  TPlex::TPlex(const TPlex& p)  {  left = p.left->Clone();  right = p.right->Clone();  }  void TPlex::Print()  {  PrintAll();  };  TBase\* TPlex::PrintAll()  {  std::cout << \*this << "\n";  return 0;  }  double TPlex::GetVal(int i) {  throw - 1;  }  void TPlex::SetVal(double val, int i) {  throw - 1;  }  double& TPlex::operator[](char\* name) {  throw - 1;  }  double& TPlex::operator[](int i) {  throw - 1;  }  TBase\* TPlex::GetChild(int i) {  if (i == 0)  return left;  else if (i == 1)  return right;  else  throw - 1;  }  void TPlex::SetChild(TBase\* c, int i)  {  if (i == 0)  left = c;  else if (i == 1)  right = c;  else  throw - 1;  }  TBase\* TPlex::GetLeft()  {  return left;  }  TBase\* TPlex::GetRight()  {  return right;  }  TPlex\* TPlex::GetLeftPlex()  {  return dynamic\_cast<TPlex\*>(left);  }  TPlex\* TPlex::GetRightPlex()  {  return dynamic\_cast<TPlex\*>(right);  }  void TPlex::SetLeft(TBase\* x)  {  left = x;  }  void TPlex::SetRight(TBase\* x)  {  right = x;  }  bool TPlex::AddLine(TPoint\* a, TPoint\* b)  {  TStack<TBase\*> stack(200);  stack.Push(this);  bool result = false;  while (!stack.IsEmpty())  {  TBase\* top\_base = stack.Get();  TBase\* right\_base = top\_base->GetChild(0);  TBase\* left\_base = top\_base->GetChild(1);  if (right\_base->GetChildCount() != 0)  {  stack.Push(right\_base);  }  else if (right\_base == a)  {  TPlex\* new\_plex = new TPlex(a, b);  top\_base->SetChild(new\_plex, 0);  result = true;  break;  }    if (left\_base->GetChildCount() != 0)  {  stack.Push(left\_base);  }  else if (left\_base == b)  {    TPlex\* new\_plex = new TPlex(b, a);  top\_base->SetChild(new\_plex, 1);  result = true;  break;  }  }  return result;  }  TBase\* TPlex::Clone()  {  return new TPlex(\*this);  }  std::ostream& operator<<(std::ostream& out, const TPlex& \_plex)  {  TBase\* base = const\_cast<TPlex\*>(&\_plex);  TPoint\* point\_first = NULL;  TPoint\* point\_second = NULL;  TStack<TBase\*> stack(200);  stack.Push(base);  while (!stack.IsEmpty())  {  base = stack.Get();  while (point\_first == NULL)  {  TBase\* left\_base = base->GetChild(0);  TPlex\* plex = dynamic\_cast<TPlex\*>(left\_base);  if (plex == NULL)  {  point\_first = dynamic\_cast<TPoint\*>(left\_base);  }  else  {  stack.Push(base);  base = left\_base;  }  }  while (point\_second == NULL)  {  TBase\* right\_base = base->GetChild(1);  TPlex\* plex = dynamic\_cast<TPlex\*>(right\_base);  if (plex == NULL)  {    point\_second = dynamic\_cast<TPoint\*>(right\_base);  }  else  {  stack.Push(base);  base = right\_base;  }  }  if (point\_first != NULL && point\_second != NULL)  {  out << "Point left: " << \*point\_first << "\n";  out << "Point right: " << \*point\_second << "\n\n";  TBase\* tmp\_base = point\_second;    if (!stack.IsEmpty())  {  base = stack.Get();    TBase\* base\_left = base->GetChild(0);  TBase\* base\_right = base->GetChild(1);  TPlex\* plex\_left = dynamic\_cast<TPlex\*>(base\_left);  TPlex\* plex\_right = dynamic\_cast<TPlex\*>(base\_right);    if (plex\_left != NULL && plex\_right != NULL)  {    TPlex\* plex\_new = new TPlex(point\_second, 0);    plex\_new->SetChild(base\_right, 1);    stack.Push(plex\_new);    point\_second = NULL;  point\_first = NULL;  }  else  {  if (plex\_left != NULL)  {  point\_first = dynamic\_cast<TPoint\*>(tmp\_base);  point\_second = NULL;  }  else  {  point\_second = dynamic\_cast<TPoint\*>(tmp\_base);  point\_first = NULL;  }  stack.Push(base);  }  }  }  }  return out;  } |