Министерство образования и науки Российской Федерации  
Федеральное государственное автономное образовательное учреждение  
высшего профессионального образования  
«Нижегородский государственный университет им. Н.И. Лобачевского»

**Институт информационных технологий, математики и механики**

Лабораторная работа

“ Хранение графических объектов.

Динамические структуры данных”

**Выполнил:** студент группы 0826-1

Власов А.С.

**Научный руководитель:**

Барышева И.В.

Нижний Новгород

2016

**Содержание**

1. Введение 3
2. Постановка задачи 3
3. Руководство пользователя 4
4. Руководство программиста 6
5. Заключение 11
6. Приложение 12

**1.**Введение  
 Под графической информацией мы понимаем всю совокупность информации, которая нанесена на самые различные носители – бумагу, пленку, кальку, картон. Для изобразительной деятельности не существует строгого перечня элементов, а, следовательно, их список невозможен. В связи с такой ситуацией возникает глобальная задача – выяснить, каким образом можно превратить любые изображения в цифровые коды, с которыми только и могут работать компьютерные устройства и как хранить полученную информацию. Следует обратить внимание на различие подходов человека и электронного устройства к изображению. Компьютерная графика появилась достаточно давно – уже в 1960-ых годах существовали полноценные графические системы, которые могут создавать различные объекты любого размера, формы и цвета.

Одной из важных задач в создании динамических структур и представлении на ЭВМ сложных математических моделей является разработка структуры хранения геометрических объектов. Данная структура должна хранить информационное описание объекта (координаты, радиус, размер) и содержать операции для динамической обработки (визуализация, скрытие, перемещение). Задача реализации такой структуры нетрудно решается для базовых геометрических объектов – точки, окружности или прямоугольника. Трудности возникают для более сложных объектов. Одним из первых и наиболее простых и естественных подходов к решению проблемы удобного и эффективного конструирования сложных объектов является метод, опирающийся на то, что сложные фигуры строятся, как правило, из базовых примитивов. В данной работе будет рассмотрена одна из структур хранения реализующая данную идею – сплетения.

Сплетение - связь элементов, основанная на сплетении указателей. Каждый элемент сплетения может содержать информацию о количестве полей с указателями и формате поля данных. Плексы (сплетения) используются для представления различных семейств связей между индивидуумами и владельцами, отражают производственные, отраслевые связи и т.п.

Плекс(многосвязный список) - это нелинейная структура данных, объединяющая такие понятия, дерево, граф и списковые структуры. У каждого элемента имеется несколько полей с указателями на другие элементы того же сплетения. В зависимости от характера взаимного расположения элементов в памяти линейные структуры можно разделить на структуры с последовательным распределением элементов в памяти (векторы, строки, массивы, стеки, очереди) и структуры с произвольным связным распределением элементов в памяти (односвязные, двусвязные списки). Пример нелинейных структур - многосвязные списки, деревья, графы.

Типичными графами являются схемы авиалиний и схемы метро, а на географических картах - изображение железных или автомобильных дорог. Выбранные точки графа называются его вершинами, а соединяющие их линии - ребрами.

АВ

В

СА

ВС

С

А

В

Принцип плекса – это хранение сложной фигуры, которую можно нарисовать “не отрывая карандаша”. Повторяющаяся точка на чертеже должна быть представлена одним и тем же объектом (не следует допускать множественности представления одного и того же значения). В плексе повторение происходит максимум у одной точки и не является серьезным недостатком.

Многосвязная структура обладает следующими свойствами:

1) на каждый элемент (узел, вершину) может быть произвольное количество ссылок;

2) каждый элемент может иметь связь с любым количеством других элементов;

3) каждая связка (ребро, дуга) может иметь направление и вес.

**2.** Постановка задачи

В рамках лабораторной работы ставится задача создания программы, которая будет рисовать геометрические объекты с помощью элемента линия, скрывать их, а также перемещать. Для удобства будем использовать графический редактор Windows. Нам потребуется структура хранения геометрических объектов – плекс. Для этого требуется реализовать абстрактный базовый класс TBase, а также наследуемые от него TPoint, TLine, TPlex представляющий точку, линию и плекс соответственно Классы должны содержать основные методы обработки геометрических объектов: отображение фигуры, скрытие, перемещение и выделение.

**3.** Структура проекта

При использовании данной структуры данных нам потребуется следующий алгоритм обхода фигуры:

1. Линия первый раз

- Запись в стек Cur

- Уход влево

1. Линия второй раз

- Запись в стек Cur

- Уход вправо

1. Линия третий раз

- Left из стека точек с изъятием

- Right из стека точек без изъятия

- Рисование линии(Left, Right)

- Cur из стека линий

1. Точка

- Запись Cur в стек точек

- Cur из стека линий

**Class TBase** (абстрактный) - для представления свойств, являющихся общими для всех графических объектов

**Поля:**

TypeFigure Type - Тип фигуры;

string Name – Имя;

bool Visible – Видимость;

int Сolor – Цвет;

int Width – Толщина (линии, точки);

int Region – Облась попадания;

int Rate – Рейтинг (в плексе)

**Методы:**

GetType(),GetVisability(), GetWidth(), GetRate() – для получения соответствующих значений полей класса;

SetColor(int \_col),SetWidth(int \_width), SetVisability() – для установки значений соответствующих полей класса.

**Виртуальные методы:**

Void IncRate() – инкрементирует рейтинг (до 3);

Void Draw(Graphics ^g) – отрисовывает объект на форме;

Bool IsInArea(int \_x, int \_y) – проверяет, попали ли координаты в какую-либо фигуру.

**Class TPoint** (наследник от TBase) - для представления свойств для геометрического объекта точка

**Поля:**

int x,y – координаты;

**Методы:**

Get\_x(),Get\_y() - получение значений координат;

MovePoint(int dx = 0, int dy = 0) - двигает точку на заданные промежутки;

Draw(Graphics ^g) - нарисовать точку;

DecRate() – уменьшает рейтинг на 1 (если rate > 0);

IsInArea(int \_x, int \_y) – попали ли координаты в точку;

Pack() - запаковать параметры точки в строку.

**Class TLine** (наследник от TBase) - для представления свойств для геометрического объекта линия

**Поля:**

TBase \*left,\*right – левая и правая точки.

**Методы:**

Inverse() – переварачивает линию(меняет местами крайние точки);

Draw(Graphics ^g) – нарисовать;

IncRate();

bool IsInArea(int \_x, int \_y) – см. Класс TPoint;

SetLeft(),SetRight() – задает левую и правую точки соответственно;

\*TBase GetLeft(),\*TBase GetRight() – получение левой и правой точки линии соответственно;

Pack() - запаковать параметры точки в строку.

**Class Plex - для создания самого плекса**

**Поля:**

\*TBase start - Начальный элемент плекса.

**Методы:**

TPoint\* SearchPoint(const string name) – поиск точки в плексе по имени;

TLine\* SearchLineContainingPoint(const string name) – поиск линии, содержащей данную точку;

void InsertLine(TLine \*temp, TLine \*res) – добавить линию temp к концу существующей res;

bool AddLine(TLine \*temp) – добавить линию к плексу;

TPoint\* CheckPoint(const int \_x, const int \_y) – поиск точки по координатам (если нет - нулевой указатель);

TLine\* CheckLine(const int \_x, const int \_y) – поиск линии, близкой к заданной точке;

void Draw(Graphics ^g) Нарисовать плекс;

void Move(const int dx, const int dy) Передвинуть плекс;

TBase\* getStart()Получить начало плекса;

void MovePoint(const string name, const int dx, const int dy) Передвинуть одну точку;

void Save(const string filename) Сохранить плекс в файл;

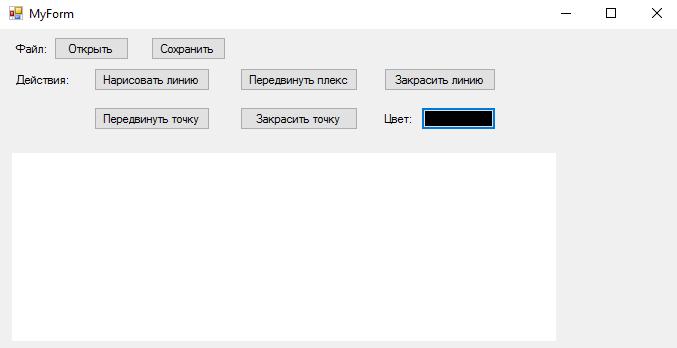
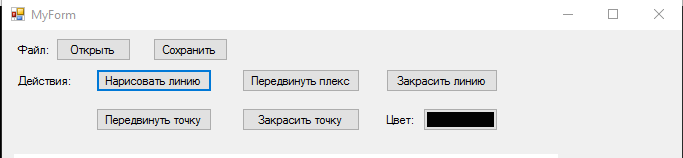
void Load(const string filename) Загрузить плекс из файла;

bool isNotEmpty()Существует ли уже плекс;

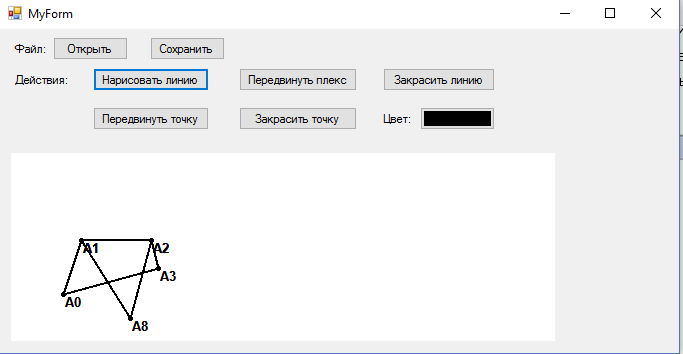
string getLastPoint()Получить последнее имя точки

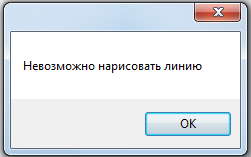
**Class TStack - написанный ранее класс, реализующий стек**

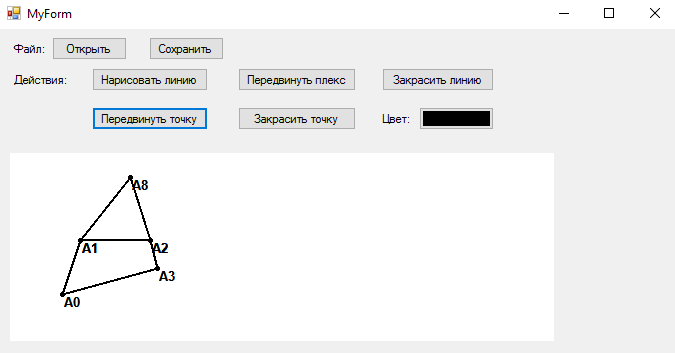
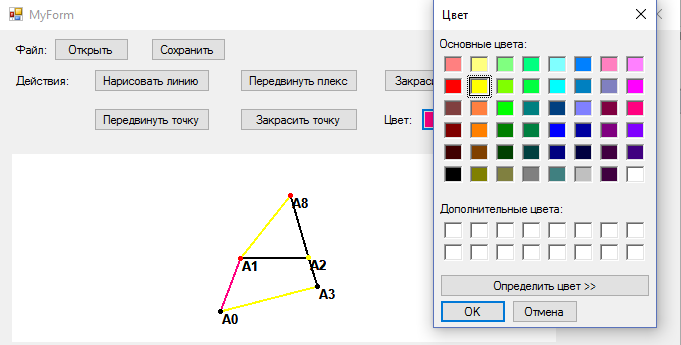
**4.** Руководство пользователя  
Для демонстрации работы с плексами над полиномами была создана форма. Приведем инструкцию по работе с ней:

1. Запустить программу. Появится главное окно.
2. Для начала работы выберете из предложенного списка необходимую операцию.
3. При нажатии кнопки “Нарисовать линию”, появляется возможность рисования линий на форме. Для этого нужно нажать левой кнопкой мыши в первой точке линии, провести по экрану и отпустить во второй точке линии. Между точками появится линия.

Замечание 1! При работе с чертежом, нельзя заходить за границы белого прямоугольника.



Замечание 2! При попытке нарисовать линию вне уже имеющегося рисунка возникнет предупреждение:

1. При нажатии кнопки “Передвинуть точку”, появляется возможность перемещать точки. Для этого нужно левой кнопкой мыши нажать на точку, которую вы собираетесь передвинуть, перевести курсор в точку, в которую будет перемещена выбранная вами точка, и отпустить кнопку мыши. Аналогично, при нажатии кнопки “Передвинуть Плекс”, можно перемещать весь рисунок.
2. При нажатии кнопок “Закрасить точку” и “Закрасить линию”, можно соответственно изменить цвет точки и линии. Предварительно нужно выбрать цвет в специальном окне, а затем нажать на точку или линию, которую вы хотите закрасить.
3. Для открытия ранее созданного рисунка нужно нажать на кнопку “Открыть”. Для Сохранения текущего рисунка необходимо нажать на кнопку “Сохранить”.

**4.** Заключение

В данной работе была изучена и реализована на примере простейшего графического редактора структура хранения геометрических объектов - плекс. Было рассмотрено два алгоритма обхода плекса: простой в реализации и понимании, но не производительный рекурсивный алгоритм, и труднопонимаемый, но быстродейственный итеративный алгоритм. При решении поставленных задач был изучен и использован механизм динамической идентификации типа данных, опыт работы с которым будет полезен в дальнейшем.

**5.** Приложение

*Код класса Stack:*

#pragma once

#include <iostream>

#include <stdio.h>

using namespace std;

template <class T>

class TStack

{

private:

T\* mas;

int MaxSize;//

int pos

public:

TStack(int \_MaxSize=100)

{

MaxSize=\_MaxSize;

mas=new T [MaxSize];

pos=-1;

};

void SetPos(int \_pos){pos=\_pos;};

~TStack();

int Full();

int Empty();

void Push(T el);

T Pop();

T Top();

void Clear();

};

template <class T>

TStack<T>::~TStack()

{

delete[] mas;

mas=NULL;

}

template <class T>

int TStack<T>::Full()

{

return (pos==MaxSize-1);

}

template <class T>

int TStack<T>::Empty()

{

return (pos==-1);

}

template <class T>

void TStack<T>::Push(T el)

{

mas[pos+1]=el; pos++;

}

template <class T>

T TStack<T>::Pop()

{

return mas[pos--];

}

template <class T>

T TStack<T>::Top()

{

return mas[pos];

}

template <class T>

void TStack<T>::Clear()

{

pos=-1;

}

#pragma once

#include <string>

using namespace System;

using namespace System::Drawing;

using namespace std;

enum TypeFigure { \_Point, \_Line, \_Rectangle, \_Circle };

class TBase

{

protected:

TypeFigure type;

string name;

bool visible;

int color;

int width;

int region;

int rate;

public:

TBase(string \_name, TypeFigure \_type, bool \_visible = true,

Color \_color = Color::Black, int \_width = 5, int \_rate = 1)

{

name = \_name;

type = \_type;

visible = \_visible;

color = \_color.ToArgb();

width = \_width;

region = 2 \* width;

rate = \_rate;

}

virtual ~TBase()

{

}

string getName() { return name; }

TypeFigure getType() { return type; }

bool getVisibility() { return visible; }

void setVisibility() { visible = !visible; }

Color getColor() { return Color::FromArgb(color); }

void setColor(Color \_color) { color = \_color.ToArgb(); }

int getWidth() { return width; }

void setWidth(const int \_width) { width = \_width; region = 2 \* width; }

int getRate() { return rate; }

virtual void incRate() = 0;

virtual void Draw(Graphics^ g) = 0;

virtual bool isInArea(const int \_x, const int \_y) = 0;

};

void StrToArray(string Pst, string PW[], int &Pk)

{

string t;

int l;

t = Pst + "|";

Pk = 0;

while (t.length()>0)

{

l = t.find("|");

if (l>0)

PW[Pk++] = t.substr(0, l);

t.erase(0, l + 1);

}

}

#pragma once

#include <string>

#include "TBase.h"

#include "TPoint.h"

using namespace System;

using namespace System::Drawing;

using namespace std;

class TLine : public TBase

{

private:

TBase \*left;

TBase \*right;

public:

TLine(TBase \*\_left, TBase \*\_right) : TBase(\_left->getName() + "," + \_right->getName(),

TypeFigure::\_Line)

{

left = \_left;

right = \_right;

width = 2;

}

TLine(const TLine &temp) : TBase(temp.name, temp.type, temp.visible,

Color::FromArgb(temp.color), temp.width, temp.rate)

{

region = temp.region;

left = temp.left;

right = temp.right;

}

TLine(const string temp, TBase \*\_left, TBase \*\_right) : TBase("", TypeFigure::\_Line)

{

string W[7];

int k = 0;

StrToArray(temp, W, k);

name = W[1];

visible = (bool)atoi(W[2].c\_str());

color = atoi(W[3].c\_str());

width = atoi(W[4].c\_str());

left = \_left;

right = \_right;

}

~TLine()

{

if (left != nullptr)

if (left->getType() == TypeFigure::\_Point)

if (left->getRate() == 1)

delete left;

else

((TPoint\*)left)->decRate();

if (right != nullptr)

if (right->getType() == TypeFigure::\_Point)

if (right->getRate() == 1)

delete right;

else

((TPoint\*)right)->decRate();

}

TLine &operator=(const TLine &temp)

{

left = temp.left;

right = temp.right;

name = temp.name;

visible = temp.visible;

color = temp.color;

width = temp.width;

region = temp.region;

rate = temp.rate;

return \*this;

}

TBase\* getLeft()

{

return left;

}

TBase\* getRight()

{

return right;

}

void setLeft(TBase\* temp)

{

left = temp;

}

void setRight(TBase\* temp)

{

right = temp;

}

void Inverse()

{

TBase \*temp = left;

left = right;

right = temp;

name = left->getName() + "," + right->getName();

}

void incRate()

{

if (rate < 3)

rate++;

else

rate = 1;

}

void Draw(Graphics^ g)

{

if ((left->getType() == TypeFigure::\_Point) && (right->getType() == TypeFigure::\_Point))

{

g->DrawLine(%Pen(Color::FromArgb(color), (float)width), ((TPoint\*)left)->getX(),

((TPoint\*)left)->getY(), ((TPoint\*)right)->getX(), ((TPoint\*)right)->getY());

left->Draw(g);

right->Draw(g);

}

}

bool isInArea(const int \_x, const int \_y)

{

float abs1 = (float)Math::Abs((((TPoint\*)left)->getX() - \_x) \* (((TPoint\*)right)->getY() - \_y)

- (((TPoint\*)right)->getX() - \_x) \* (((TPoint\*)left)->getY() - \_y));

float length = (float)Math::Sqrt((((TPoint\*)right)->getX() - ((TPoint\*)left)->getX()) \* (((TPoint\*)right)->getX() - ((TPoint\*)left)->getX())

+ (((TPoint\*)right)->getY() - ((TPoint\*)left)->getY()) \* (((TPoint\*)right)->getY() - ((TPoint\*)left)->getY()));

float OH = abs1 / length;

int AOAB = ((\_x - ((TPoint\*)left)->getX()) \* (((TPoint\*)right)->getX() - ((TPoint\*)left)->getX())

+ (\_y - ((TPoint\*)left)->getY()) \* (((TPoint\*)right)->getY() - ((TPoint\*)left)->getY()));

int BOBA = ((\_x - ((TPoint\*)right)->getX()) \* (((TPoint\*)left)->getX() - ((TPoint\*)right)->getX())

+ (\_y - ((TPoint\*)right)->getY()) \* (((TPoint\*)left)->getY() - ((TPoint\*)right)->getY()));

if ((OH <= (float)region) && (AOAB \* BOBA >= 0))

return true;

else

return false;

}

string Pack()

{

string res = "\_Line";

res += "|" + name + "|" +

to\_string(visible) + "|" +

to\_string(color) + "|" +

to\_string(width) + "|" +

to\_string(rate);

return res;

}

};

#pragma once

#include <string>

#include <fstream>

#include "TBase.h"

#include "TPoint.h"

#include "TLine.h"

#include "TStack.h"

using namespace System;

using namespace System::Drawing;

using namespace std;

class TPlex

{

private:

TBase \*start;

public:

TPlex(TBase\* \_start = nullptr)

{

start = \_start;

}

TPlex(const TPlex &temp)

{

}

~TPlex()

{

TStack<TBase\*> LineStack;

LineStack.Put(nullptr);

TBase \*current = start;

while (current != nullptr)

{

if (current->getType() == TypeFigure::\_Point)

{

current = LineStack.Get();

current->incRate();

}

else

{

switch (current->getRate())

{

case 1:

LineStack.Put(current);

current = ((TLine\*)current)->getLeft();

break;

case 2:

LineStack.Put(current);

current = ((TLine\*)current)->getRight();

break;

case 3:

delete current;

current = LineStack.Get();

if (current != nullptr)

current->incRate();

break;

}

}

}

}

TPlex &operator=(const TPlex temp)

{

if (start != nullptr)

delete start;

TPlex \*tmp = new TPlex(temp);

start = tmp->start;

}

private:

TPoint\* SearchPoint(const string name)

{

TPoint\* result = 0;

TBase\* current = start;

TStack<TBase\*> LineStack;

LineStack.Put(nullptr);

while ((current != nullptr) && (result == 0))

if (current->getType() == TypeFigure::\_Point)

{

if (current->getName() == name)

result = (TPoint\*)current;

else

{

current = LineStack.Get();

current->incRate();

}

}

else

switch (current->getRate())

{

case 1:

LineStack.Put(current);

current = ((TLine\*)current)->getLeft();

break;

case 2:

LineStack.Put(current);

current = ((TLine\*)current)->getRight();

break;

case 3:

current->incRate();

current = LineStack.Get();

current->incRate();

break;

}

while (LineStack.Show() != nullptr)

{

current = LineStack.Get();

while (current->getRate() != 1)

current->incRate();

}

return result;

}

TLine\* SearchLineContainingPoint(const string name)

{

TLine\* result = 0;

TBase\* current = start;

TStack<TBase\*> LineStack;

LineStack.Put(nullptr);

while ((current != nullptr) && (result == nullptr))

if (current->getType() == TypeFigure::\_Point)

{

if (current->getName() == name)

result = (TLine\*)LineStack.Show();

else

{

current = LineStack.Get();

current->incRate();

}

}

else

switch (current->getRate())

{

case 1:

LineStack.Put(current);

current = ((TLine\*)current)->getLeft();

break;

case 2:

LineStack.Put(current);

current = ((TLine\*)current)->getRight();

break;

case 3:

current->incRate();

current = LineStack.Get();

if (current != nullptr)

current->incRate();

break;

}

while (LineStack.Show() != nullptr)

{

current = LineStack.Get();

while (current->getRate() != 1)

current->incRate();

}

return result;

}

void InsertLine(TLine \*temp, TLine \*res)

{

if (res->getName().find(temp->getLeft()->getName()) == 0)

{

temp->setLeft(res->getLeft());

res->setLeft(temp);

}

else

{

temp->setLeft(res->getRight());

res->setRight(temp);

}

}

public:

TPoint\* CheckPoint(const int \_x, const int \_y)

{

TPoint \*result = nullptr;

TStack<TBase\*> LineStack;

LineStack.Put(nullptr);

TBase \*current = start;

while (current != nullptr)

if (current->getType() == TypeFigure::\_Point)

{

if (current->isInArea(\_x, \_y))

result = (TPoint\*)current;

current = LineStack.Get();

current->incRate();

}

else

switch (current->getRate())

{

case 1:

LineStack.Put(current);

current = ((TLine\*)current)->getLeft();

break;

case 2:

LineStack.Put(current);

current = ((TLine\*)current)->getRight();

break;

case 3:

current->incRate();

current = LineStack.Get();

if (current != nullptr)

current->incRate();

break;

}

return result;

}

TLine\* CheckLine(const int \_x, const int \_y)

{

TStack<TBase\*> LineStack;

LineStack.Put(nullptr);

TStack<TBase\*> PointStack;

TLine \*result = nullptr;

TBase \*current = start;

while (current != nullptr)

if (current->getType() == TypeFigure::\_Point)

{

PointStack.Put(current);

current = LineStack.Get();

current->incRate();

}

else

switch (current->getRate())

{

case 1:

LineStack.Put(current);

current = ((TLine\*)current)->getLeft();

break;

case 2:

LineStack.Put(current);

current = ((TLine\*)current)->getRight();

break;

case 3:

TLine \*temp = new TLine(\*((TLine\*)current));

temp->setRight(PointStack.Get());

temp->setLeft(PointStack.Show());

if (temp->isInArea(\_x, \_y))

result = (TLine\*)current;

temp->setLeft(nullptr);

temp->setRight(nullptr);

delete temp;

current->incRate();

current = LineStack.Get();

if (current != nullptr)

current->incRate();

break;

}

return result;

}

bool AddLine(TLine \*temp)

{

if (start == nullptr)

{

start = temp;

return true;

}

TLine \*Left = SearchLineContainingPoint(temp->getLeft()->getName()),

\*Right = SearchLineContainingPoint(temp->getRight()->getName());

if (Left == Right)

return false;

if (Left != nullptr)

{

InsertLine(temp, Left);

if (Right != nullptr)

{

if (Right->getName().find(temp->getRight()->getName()) == 0)

{

temp->setRight(Right->getLeft());

Right->getLeft()->incRate();

}

else

{

temp->setRight(Right->getRight());

Right->getRight()->incRate();

}

}

}

else

if (Right != nullptr)

{

temp->Inverse();

InsertLine(temp, Right);

}

return true;

}

void Draw(Graphics ^g)

{

TStack<TBase\*> LineStack;

LineStack.Put(nullptr);

TStack<TBase\*> PointStack;

TBase \*current = start;

while (current != nullptr)

if (current->getType() == TypeFigure::\_Point)

{

PointStack.Put(current);

current = LineStack.Get();

current->incRate();

}

else

switch (current->getRate())

{

case 1:

LineStack.Put(current);

current = ((TLine\*)current)->getLeft();

break;

case 2:

LineStack.Put(current);

current = ((TLine\*)current)->getRight();

break;

case 3:

TLine \*temp = new TLine(\*((TLine\*)current));

temp->setRight(PointStack.Get());

temp->setLeft(PointStack.Show());

temp->Draw(g);

temp->setLeft(nullptr);

temp->setRight(nullptr);

delete temp;

current->incRate();

current = LineStack.Get();

if (current != nullptr)

current->incRate();

break;

}

}

void Move(const int dx, const int dy)

{

TStack<TBase\*> LineStack;

LineStack.Put(nullptr);

TBase \*current = start;

while (current != nullptr)

if (current->getType() == TypeFigure::\_Point)

{

((TPoint\*)current)->Move(dx / current->getRate(), dy / current->getRate());

current = LineStack.Get();

current->incRate();

}

else

switch (current->getRate())

{

case 1:

LineStack.Put(current);

current = ((TLine\*)current)->getLeft();

break;

case 2:

LineStack.Put(current);

current = ((TLine\*)current)->getRight();

break;

case 3:

current->incRate();

current = LineStack.Get();

if (current != nullptr)

current->incRate();

break;

}

}

void Move(TBase \*temp, const int dx, const int dy)

{

if (temp->getType() != TypeFigure::\_Point)

Move(((TLine\*)temp)->getLeft(), dx, dy);

else

{

((TPoint\*)temp)->Move(dx / temp->getRate(), dy / temp->getRate());

return;

}

Move(((TLine\*)temp)->getRight(), dx, dy);

}

TBase\* getStart()

{

return start;

}

void MovePoint(const string name, const int dx, const int dy)

{

TPoint \*tmp = SearchPoint(name);

if (tmp != nullptr)

tmp->Move(dx, dy);

}

void Save(const string filename)

{

ofstream fout(filename);

TStack<TBase\*> LineStack;

LineStack.Put(nullptr);

TBase \*current = start;

while (current != nullptr)

if (current->getType() == TypeFigure::\_Point)

{

fout << ((TPoint\*)current)->Pack() << endl;

current = LineStack.Get();

current->incRate();

}

else

switch (current->getRate())

{

case 1:

LineStack.Put(current);

current = ((TLine\*)current)->getLeft();

break;

case 2:

LineStack.Put(current);

current = ((TLine\*)current)->getRight();

break;

case 3:

current->incRate();

fout << ((TLine\*)current)->Pack() << endl;

current = LineStack.Get();

if (current != nullptr)

current->incRate();

break;

}

fout.close();

}

void Load(const string filename)

{

if (start != nullptr)

this->~TPlex();

ifstream fin(filename);

string str;

TStack<TBase\*> stack;

TPoint \*point;

TLine \*line;

getline(fin, str);

while (str != "")

{

if (str.find("\_Point") == 0)

{

point = new TPoint(str);

stack.Put(point);

}

else

{

line = new TLine(str, nullptr, nullptr);

line->setRight(stack.Get());

line->setLeft(stack.Get());

stack.Put(line);

}

getline(fin, str);

}

start = stack.Get();

fin.close();

}

bool isNotEmpty()

{

return (start == nullptr) ? false : true;

}

string getLastPoint()

{

TStack<TBase\*> LineStack;

LineStack.Put(nullptr);

string result = "";

TBase \*current = start;

while (current != nullptr)

if (current->getType() == TypeFigure::\_Point)

{

if (current->getName() > result)

result = current->getName();

current = LineStack.Get();

current->incRate();

}

else

switch (current->getRate())

{

case 1:

LineStack.Put(current);

current = ((TLine\*)current)->getLeft();

break;

case 2:

LineStack.Put(current);

current = ((TLine\*)current)->getRight();

break;

case 3:

current->incRate();

current = LineStack.Get();

if (current != nullptr)

current->incRate();

break;

}

return result;

}

};

#pragma once

#include <string>

#include "TBase.h"

using namespace System;

using namespace System::Drawing;

using namespace std;

class TPoint : public TBase

{

private:

int x;

int y;

public:

TPoint(string \_name, int \_x, int \_y) : TBase(\_name, TypeFigure::\_Point)

{

x = \_x;

y = \_y;

}

TPoint(const TPoint &temp) : TBase(temp.name, temp.type, temp.visible,

Color::FromArgb(temp.color), temp.width, temp.rate)

{

region = temp.region;

x = temp.x;

y = temp.y;

}

TPoint(const string temp) : TBase("", TypeFigure::\_Point)

{

string W[9];

int k = 0;

StrToArray(temp, W, k);

name = W[1];

visible = (bool)atoi(W[2].c\_str());

color = atoi(W[3].c\_str());

width = atoi(W[4].c\_str());

rate = atoi(W[5].c\_str());

x = atoi(W[6].c\_str());

y = atoi(W[7].c\_str());

}

~TPoint()

{

}

TPoint &operator=(const TPoint %temp)

{

x = temp.x;

y = temp.y;

name = temp.name;

visible = temp.visible;

color = temp.color;

width = temp.width;

region = temp.region;

rate = temp.rate;

return \*this;

}

int getX() { return x; }

int getY() { return y; }

void Move(int dx, int dy)

{

x += dx;

y += dy;

}

void incRate()

{

rate++;

}

bool isInArea(const int \_x, const int \_y)

{

return ((x - \_x) \* (x - \_x) + (y - \_y) \* (y - \_y) <= region \* region);

}

void Draw(Graphics^ g)

{

g->FillEllipse(%SolidBrush(Color::FromArgb(color)), (float)x - (float)width / 2.0f,

(float)y - (float)width / 2.0f, (float)width, (float)width);

g->DrawString(%String(name.c\_str()), %Font("Arial", 10, FontStyle::Bold), %SolidBrush(Color::Black), (float)x, (float)y);

}

void decRate()

{

if (rate > 1)

rate--;

}

string Pack()

{

string res = "\_Point";

res += "|" + name + "|" +

to\_string(visible) + "|" +

to\_string(color) + "|" +

to\_string(width) + "|" +

to\_string(rate) + "|" +

to\_string(x) + "|" +

to\_string(y);

return res;

}

};

class PointName

{

private:

string lastname;

public:

PointName()

{

lastname = "A0";

}

string Get()

{

string temp = lastname;

int i;

for (i = lastname.length() - 1; (i >= 0) && (lastname[i] == 'Z'); i--);

if (i < 0)

{

lastname.insert(0, "A");

i = 1;

}

else

lastname[i++]++;

while (i < lastname.length())

lastname[i++] = 'A';

return temp;

}

void SetLastName(const string temp)

{

lastname = temp;

}

};

#pragma once

#include <string>

#include <msclr/marshal\_cppstd.h>

#include "TPlex.h"

namespace Ai\_Plex {

using namespace System;

using namespace System::ComponentModel;

using namespace System::Collections;

using namespace System::Windows::Forms;

using namespace System::Data;

using namespace System::Drawing;

/// <summary>

/// Сводка для MyForm

/// </summary>

public ref class MyForm : public System::Windows::Forms::Form

{

public:

MyForm(void)

{

InitializeComponent();

//

//TODO: добавьте код конструктора

//

}

protected:

/// <summary>

/// Освободить все используемые ресурсы.

/// </summary>

~MyForm()

{

if (components)

{

delete components;

}

}

public: System::Windows::Forms::PictureBox^ pictureBox1;

protected:

protected:

private: System::Windows::Forms::Button^ button1;

private: System::Windows::Forms::Button^ button2;

private: System::Windows::Forms::Button^ button3;

private: System::Windows::Forms::Label^ label1;

private: System::Windows::Forms::Label^ label2;

private: System::Windows::Forms::Button^ button4;

private: System::Windows::Forms::Button^ button5;

private: System::Windows::Forms::Button^ button6;

private: System::Windows::Forms::Button^ button7;

private: System::Windows::Forms::ColorDialog^ colorDialog1;

private: System::Windows::Forms::OpenFileDialog^ openFileDialog1;

private: System::Windows::Forms::SaveFileDialog^ saveFileDialog1;

private: System::Windows::Forms::Label^ label3;

private: System::Windows::Forms::Button^ button8;

private:

/// <summary>

/// Обязательная переменная конструктора.

/// </summary>

System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code

/// <summary>

/// Требуемый метод для поддержки конструктора — не изменяйте

/// содержимое этого метода с помощью редактора кода.

/// </summary>

void InitializeComponent(void)

{

this->pictureBox1 = (gcnew System::Windows::Forms::PictureBox());

this->button1 = (gcnew System::Windows::Forms::Button());

this->button2 = (gcnew System::Windows::Forms::Button());

this->button3 = (gcnew System::Windows::Forms::Button());

this->label1 = (gcnew System::Windows::Forms::Label());

this->label2 = (gcnew System::Windows::Forms::Label());

this->button4 = (gcnew System::Windows::Forms::Button());

this->button5 = (gcnew System::Windows::Forms::Button());

this->button6 = (gcnew System::Windows::Forms::Button());

this->button7 = (gcnew System::Windows::Forms::Button());

this->colorDialog1 = (gcnew System::Windows::Forms::ColorDialog());

this->openFileDialog1 = (gcnew System::Windows::Forms::OpenFileDialog());

this->saveFileDialog1 = (gcnew System::Windows::Forms::SaveFileDialog());

this->label3 = (gcnew System::Windows::Forms::Label());

this->button8 = (gcnew System::Windows::Forms::Button());

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->pictureBox1))->BeginInit();

this->SuspendLayout();

//

// pictureBox1

//

this->pictureBox1->Anchor = static\_cast<System::Windows::Forms::AnchorStyles>((((System::Windows::Forms::AnchorStyles::Top | System::Windows::Forms::AnchorStyles::Bottom)

| System::Windows::Forms::AnchorStyles::Left)

| System::Windows::Forms::AnchorStyles::Right));

this->pictureBox1->BackColor = System::Drawing::SystemColors::ButtonHighlight;

this->pictureBox1->Location = System::Drawing::Point(12, 124);

this->pictureBox1->Name = L"pictureBox1";

this->pictureBox1->Size = System::Drawing::Size(544, 188);

this->pictureBox1->TabIndex = 0;

this->pictureBox1->TabStop = false;

this->pictureBox1->SizeChanged += gcnew System::EventHandler(this, &MyForm::pictureBox1\_SizeChanged);

this->pictureBox1->MouseDown += gcnew System::Windows::Forms::MouseEventHandler(this, &MyForm::pictureBox1\_MouseDown);

this->pictureBox1->MouseUp += gcnew System::Windows::Forms::MouseEventHandler(this, &MyForm::pictureBox1\_MouseUp);

//

// button1

//

this->button1->Location = System::Drawing::Point(421, 78);

this->button1->Name = L"button1";

this->button1->Size = System::Drawing::Size(75, 23);

this->button1->TabIndex = 1;

this->button1->UseVisualStyleBackColor = true;

this->button1->Click += gcnew System::EventHandler(this, &MyForm::button1\_Click);

//

// button2

//

this->button2->Location = System::Drawing::Point(54, 8);

this->button2->Name = L"button2";

this->button2->Size = System::Drawing::Size(75, 23);

this->button2->TabIndex = 2;

this->button2->Text = L"Открыть";

this->button2->UseVisualStyleBackColor = true;

this->button2->Click += gcnew System::EventHandler(this, &MyForm::button2\_Click);

//

// button3

//

this->button3->Location = System::Drawing::Point(94, 39);

this->button3->Name = L"button3";

this->button3->Size = System::Drawing::Size(116, 23);

this->button3->TabIndex = 3;

this->button3->Text = L"Нарисовать линию";

this->button3->UseVisualStyleBackColor = true;

this->button3->Click += gcnew System::EventHandler(this, &MyForm::button3\_Click);

//

// label1

//

this->label1->AutoSize = true;

this->label1->Location = System::Drawing::Point(12, 13);

this->label1->Name = L"label1";

this->label1->Size = System::Drawing::Size(39, 13);

this->label1->TabIndex = 4;

this->label1->Text = L"Файл:";

//

// label2

//

this->label2->AutoSize = true;

this->label2->Location = System::Drawing::Point(13, 44);

this->label2->Name = L"label2";

this->label2->Size = System::Drawing::Size(60, 13);

this->label2->TabIndex = 5;

this->label2->Text = L"Действия:";

//

// button4

//

this->button4->Location = System::Drawing::Point(94, 78);

this->button4->Name = L"button4";

this->button4->Size = System::Drawing::Size(116, 23);

this->button4->TabIndex = 6;

this->button4->Text = L"Передвинуть точку";

this->button4->UseVisualStyleBackColor = true;

this->button4->Click += gcnew System::EventHandler(this, &MyForm::button4\_Click);

//

// button5

//

this->button5->Location = System::Drawing::Point(240, 39);

this->button5->Name = L"button5";

this->button5->Size = System::Drawing::Size(118, 23);

this->button5->TabIndex = 7;

this->button5->Text = L"Передвинуть плекс";

this->button5->UseVisualStyleBackColor = true;

this->button5->Click += gcnew System::EventHandler(this, &MyForm::button5\_Click);

//

// button6

//

this->button6->Location = System::Drawing::Point(240, 78);

this->button6->Name = L"button6";

this->button6->Size = System::Drawing::Size(118, 23);

this->button6->TabIndex = 8;

this->button6->Text = L"Закрасить точку";

this->button6->UseVisualStyleBackColor = true;

this->button6->Click += gcnew System::EventHandler(this, &MyForm::button6\_Click);

//

// button7

//

this->button7->Location = System::Drawing::Point(384, 39);

this->button7->Name = L"button7";

this->button7->Size = System::Drawing::Size(112, 23);

this->button7->TabIndex = 9;

this->button7->Text = L"Закрасить линию";

this->button7->UseVisualStyleBackColor = true;

this->button7->Click += gcnew System::EventHandler(this, &MyForm::button7\_Click);

//

// openFileDialog1

//

this->openFileDialog1->DefaultExt = L"plex";

this->openFileDialog1->FileName = L"openFileDialog1";

this->openFileDialog1->Filter = L"Plex files|\*.plex";

this->openFileDialog1->Title = L"Открыть плекс";

//

// saveFileDialog1

//

this->saveFileDialog1->DefaultExt = L"plex";

this->saveFileDialog1->FileName = L"Plex1";

this->saveFileDialog1->Filter = L"Plex files|\*.plex";

this->saveFileDialog1->Title = L"Сохранить плекс";

this->saveFileDialog1->FileOk += gcnew System::ComponentModel::CancelEventHandler(this, &MyForm::saveFileDialog1\_FileOk);

//

// label3

//

this->label3->AutoSize = true;

this->label3->Location = System::Drawing::Point(381, 83);

this->label3->Name = L"label3";

this->label3->Size = System::Drawing::Size(35, 13);

this->label3->TabIndex = 10;

this->label3->Text = L"Цвет:";

//

// button8

//

this->button8->Location = System::Drawing::Point(151, 8);

this->button8->Name = L"button8";

this->button8->Size = System::Drawing::Size(75, 23);

this->button8->TabIndex = 11;

this->button8->Text = L"Сохранить";

this->button8->UseVisualStyleBackColor = true;

this->button8->Click += gcnew System::EventHandler(this, &MyForm::button8\_Click);

//

// MyForm

//

this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);

this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;

this->ClientSize = System::Drawing::Size(680, 324);

this->Controls->Add(this->button8);

this->Controls->Add(this->label3);

this->Controls->Add(this->button7);

this->Controls->Add(this->button6);

this->Controls->Add(this->button5);

this->Controls->Add(this->button4);

this->Controls->Add(this->label2);

this->Controls->Add(this->label1);

this->Controls->Add(this->button3);

this->Controls->Add(this->button2);

this->Controls->Add(this->button1);

this->Controls->Add(this->pictureBox1);

this->Name = L"MyForm";

this->Text = L"MyForm";

this->FormClosing += gcnew System::Windows::Forms::FormClosingEventHandler(this, &MyForm::MyForm\_FormClosing);

this->Load += gcnew System::EventHandler(this, &MyForm::MyForm\_Load);

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->pictureBox1))->EndInit();

this->ResumeLayout(false);

this->PerformLayout();

}

#pragma endregion

int flag;

TPlex \*plex;

PointName \*pointname;

Graphics^ g;

TPoint\* Left;

TPoint\* Right;

private: System::Void button5\_Click(System::Object^ sender, System::EventArgs^ e) {

flag = 2;

}

private: System::Void pictureBox1\_MouseDown(System::Object^ sender, System::Windows::Forms::MouseEventArgs^ e) {

Left = nullptr;

Right = nullptr;

int x = e->X;

int y = e->Y;

if (flag == 0) {

if (plex->isNotEmpty())

Left = plex->CheckPoint(x, y);

if (Left == nullptr)

Left = new TPoint(pointname->Get(), x, y);

}

if (flag == 1) {

if (plex->isNotEmpty())

Left = plex->CheckPoint(x, y);

}

if (flag == 2) {

Left = new TPoint("temp", x, y);

}

if (flag == 3) {

if (plex->isNotEmpty())

Left = plex->CheckPoint(x, y);

}

}

private: System::Void pictureBox1\_MouseUp(System::Object^ sender, System::Windows::Forms::MouseEventArgs^ e) {

int x = e->X;

int y = e->Y;

TLine \*line;

if (flag == 0) {

if (plex->isNotEmpty())

Right = plex->CheckPoint(x, y);

if (Left != nullptr)

if (Left->isInArea(x, y))

Right = Left;

if (Left != Right)

{

if (Right == nullptr)

Right = new TPoint(pointname->Get(), x, y);

line = new TLine(Left, Right);

if (!plex->AddLine(line))

{

MessageBox::Show("Невозможно нарисовать линию");

delete line;

}

}

}

if (flag == 1) {

if (Left != nullptr)

if (!Left->isInArea(x, y))

Left->Move(x - Left->getX(), y - Left->getY());

}

if (flag == 2) {

if (plex->isNotEmpty())

plex->Move(x - Left->getX(), y - Left->getY());

delete Left;

}

if (flag == 3) {

if (Left != nullptr)

if (Left->isInArea(x, y))

Left->setColor(button1->BackColor);

}

if (flag == 4) {

TLine \*line = nullptr;

if (plex->isNotEmpty())

line = plex->CheckLine(x, y);

if (line != nullptr)

line->setColor(button1->BackColor);

}

g->Clear(Color::White);

if (plex->isNotEmpty())

plex->Draw(g);

}

private: System::Void button3\_Click(System::Object^ sender, System::EventArgs^ e) {

flag = 0;

}

private: System::Void pictureBox1\_SizeChanged(System::Object^ sender, System::EventArgs^ e) {

g = pictureBox1->CreateGraphics();

g->Clear(Color::White);

if (plex->isNotEmpty())

plex->Draw(g);

}

private: System::Void button4\_Click(System::Object^ sender, System::EventArgs^ e) {

flag = 1;

}

private: System::Void MyForm\_Load(System::Object^ sender, System::EventArgs^ e) {

plex = new TPlex();

flag = 0;

button1->BackColor = colorDialog1->Color;

pointname = new PointName();

g = pictureBox1->CreateGraphics();

}

private: System::Void MyForm\_FormClosing(System::Object^ sender, System::Windows::Forms::FormClosingEventArgs^ e) {

delete plex;

}

private: System::Void saveFileDialog1\_FileOk(System::Object^ sender, System::ComponentModel::CancelEventArgs^ e) {

}

private: System::Void button6\_Click(System::Object^ sender, System::EventArgs^ e) {

flag = 3;

}

private: System::Void button1\_Click(System::Object^ sender, System::EventArgs^ e) {

if (colorDialog1->ShowDialog() == System::Windows::Forms::DialogResult::OK)

{

button1->BackColor = colorDialog1->Color;

}

}

private: System::Void button7\_Click(System::Object^ sender, System::EventArgs^ e) {

flag = 4;

}

private: System::Void button2\_Click(System::Object^ sender, System::EventArgs^ e) {

if (openFileDialog1->ShowDialog() == System::Windows::Forms::DialogResult::OK)

{

String^ filename = openFileDialog1->FileName;

plex->Load(msclr::interop::marshal\_as<std::string>(filename));

g->Clear(Color::White);

plex->Draw(g);

pointname->SetLastName(plex->getLastPoint());

pointname->Get();

}

}

private: System::Void button8\_Click(System::Object^ sender, System::EventArgs^ e) {

if (saveFileDialog1->ShowDialog() == System::Windows::Forms::DialogResult::OK)

{

String^ filename = saveFileDialog1->FileName;

plex->Save(msclr::interop::marshal\_as<std::string>(filename));

}

}

};

}