МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ   
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КПІ»

Кафедра  
Автоматизованих систем обробки інформації та управління

Звіт  
з додаткової роботи  
з дисципліни  
«Алгоритмізація та програмування»

Виконав:  
Халін О. І.

Студент гр.  
ІС-62, ФІОТ

1 курс, залікова книжка №ІС-6226

# ЗМІСТ

[ЗМІСТ 2](#_Toc471132670)

[ВСТУП 4](#_Toc471132671)

[Тема роботи 4](#_Toc471132672)

[Мета роботи 4](#_Toc471132673)

[Актуальність роботи 4](#_Toc471132674)

[ОСНОВНА ЧАСТИНА 6](#_Toc471132675)

[Теоретична частина з предмету 6](#_Toc471132676)

[Побудова опуклої оболонки для множини з N точок площини 6](#_Toc471132677)

[Перевірка опуклості многокутника 9](#_Toc471132678)

[Перевірка приналежності точки внутрішньої області многокутника 9](#_Toc471132679)

[Коло, що «охоплює» N точок площини 11](#_Toc471132680)

[Теоретична частина з інформатики 12](#_Toc471132681)

[Елементи управління 12](#_Toc471132682)

[Опис роботи 13](#_Toc471132683)

[«Теорія» 13](#_Toc471132684)

[«Практика» 14](#_Toc471132685)

[Посібник користувача 14](#_Toc471132686)

[Панель елементів 14](#_Toc471132687)

[Панель задач 15](#_Toc471132688)

[Код програми 16](#_Toc471132689)

[Клас “Dot” – клас точки 16](#_Toc471132690)

[Клас “Line” – клас прямої 18](#_Toc471132691)

[Клас “Polygon” – клас многокутника 21](#_Toc471132692)

[Клас “Circle” – клас кола 24](#_Toc471132693)

[Клас “MyPoint” – клас точки для задачі «Мінімальне коло навколо точок» 26](#_Toc471132694)

[Клас “MyCircle” – клас точки для задачі «Мінімальне коло навколо точок» 27](#_Toc471132695)

[Клас “Geometry” – клас збору всього до купи 28](#_Toc471132696)

[Код “Form” 35](#_Toc471132697)

[Клас “Program” 49](#_Toc471132698)

[Блок-схема 50](#_Toc471132699)

[Практичне застосування програми 50](#_Toc471132700)

[Програмно-апаратні вимоги 50](#_Toc471132701)

[Комплектація програми 51](#_Toc471132702)

[Використані програмні засоби 51](#_Toc471132703)

[ВИСНОВКИ 52](#_Toc471132704)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 53](#_Toc471132705)

[Література 53](#_Toc471132706)

[Інтернет-джерела 53](#_Toc471132707)

# ВСТУП

## Тема роботи

Тема роботи була обрана з декількох причин. По-перше, через зацікавленість у математиці та в точних науках взагалі. По-друге, через бажання зробити корисну програму, яка б стала в нагоді вчителям, які хочуть наочно продемонструвати засоби вирішення задач обчислювальної геометрії на площині, та учням, які вивчають тему самостійно та прагнуть її краще опанувати, також для олімпіадників при підготовці до олімпіад високого рівня. Врешті-решт була обрана дуже цікава, хоч і складна тема: «Обчислювальна геометрія на площині».

## Мета роботи

Основною метою роботи було дослідити методи вирішення задач обчислювальної геометрії та змоделювати деякі з них. Для досягнення цієї мети було поставлено наступні задачі:

1. Побудова опуклої оболонки для множини з N точок площини;
2. Перевірка опуклості многокутника;
3. Перевірка приналежності точки внутрішньої області многокутника;
4. Коло, що «охоплює» N точок площини.

## Актуальність роботи

Я вважаю, що тема «Обчислювальна геометрія на площині» є досить актуальною темою в наші дні, тому що:

* Ця тема є досить перспективною в програмуванні і має велику кількість алгоритмів які корисно запрограмувати, щоб в подальшому легше було зрозуміти дану тему;
* Для одинадцятикласників ця тема є новою, вона не вивчається в основному шкільному курсі, тож вона є не легкою в розумінні , чим не менш цікава;
* Переважна частина літератури з даного питання була випущена в останні 15-20 років і більшість з них на англійській мові, тож знайти насправді корисну інформацію для вирішення задач обчислювальної геометрії з метою їх подальшого кодування досить нелегко.

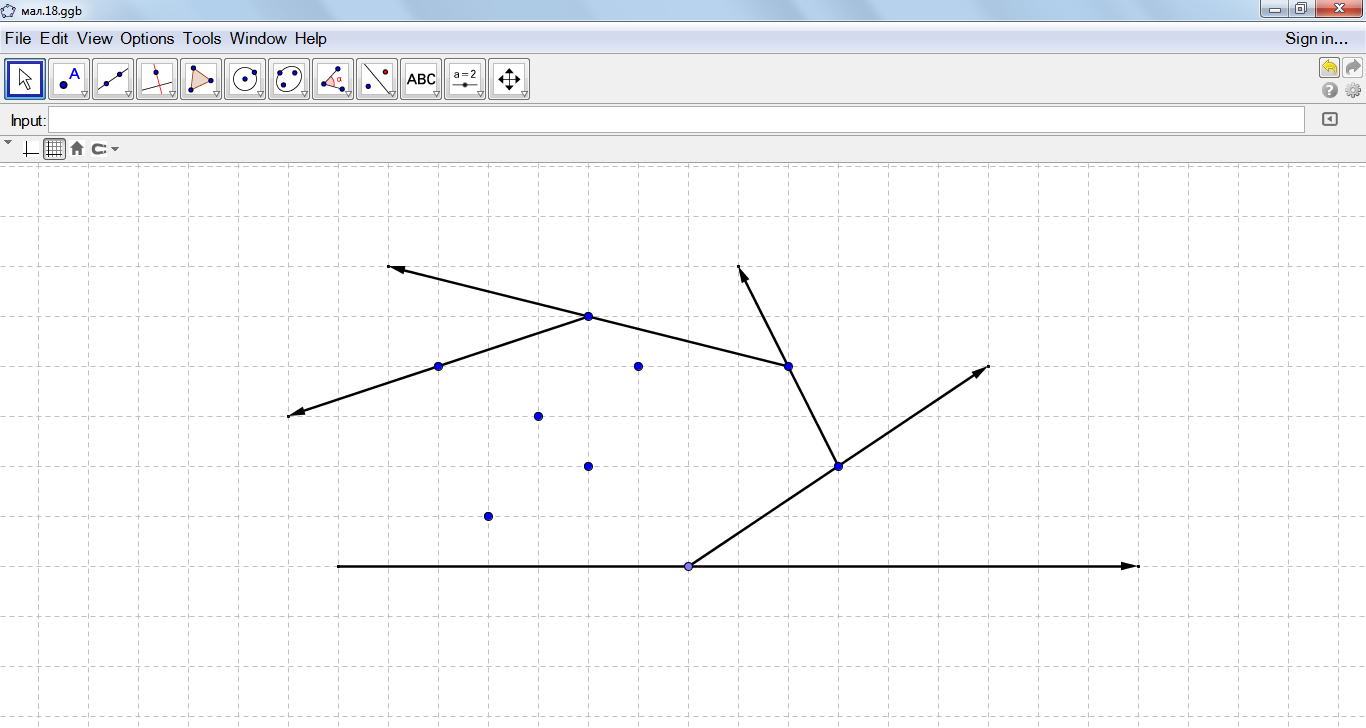
# ОСНОВНА ЧАСТИНА

## Теоретична частина з предмету

### Побудова опуклої оболонки для множини з N точок площини

Опуклою оболонкою деякої заданої безлічі точок називається перетин усіх опуклих великих кількостей, що містять задану множину. Для кінцевої безлічі точок опуклої оболонки завжди буде опуклий многокутник, усі вершини якого є точками початкової великої кількості.

Завдання полягає в тому, щоб для заданої кінцевої безлічі точок знайти вершини опуклої оболонки цієї великої кількості. Перераховуватимемо вершини в порядку обходу проти годинникової стрілки. Для ефективного вирішення цього завдання існує декілька різних алгоритмів. Приведемо найбільш просту реалізацію одного з них - алгоритму Джарвіса. Цей алгоритм іноді називають «завертанням подарунка».

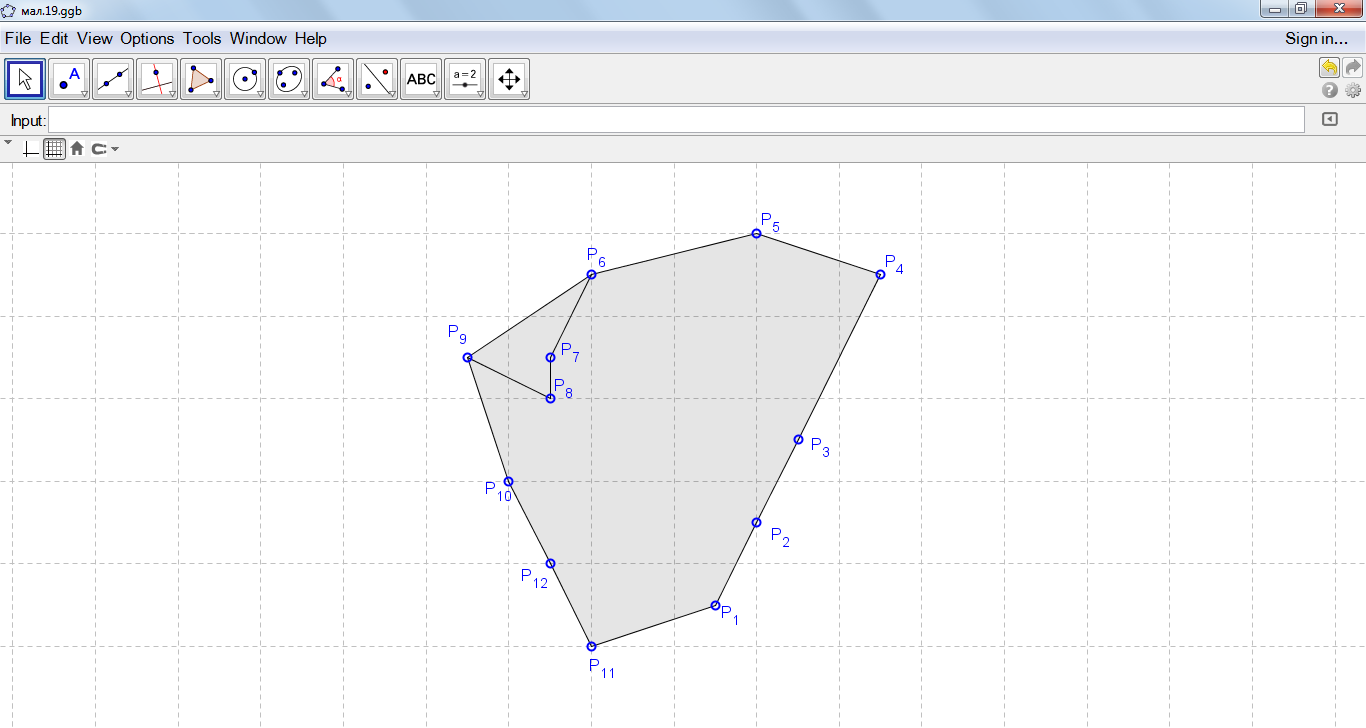


Перерахування точок шуканої межі опуклого многокутника розпочнемо з правої нижньої точки , яка свідомо належить межі опуклої оболонки. Позначимо її координати . Наступною при вказаному порядку обходу буде точка . Вона, очевидно, має ту властивість, що усі інші точки лежать «зліва» від вектору , тобто орієнтований кут між векторами і невід'ємний для будь-якої точки нашої множини. Для кандидата на роль точки перевіряємо виконання умови з усіма точками . Якщо точок, що задовольняють цій умові, декілька, то вершиною шуканого многокутника стане та з них, для якої довжина вектору максимальна.

Поступатимемо так само і далі. Припустимо, вже знайдена а вершина опуклої оболонки. Для наступної точки косі добутки невід’ємні для всіх точок . Якщо таких точок декілька, то вибираємо ту, для якої вектор має найбільшу довжину. Безпосередньо пошук такої точки можна здійснювати так. Спочатку ми можемо вважати наступною, , будь-яку точку. Потім обчислюємо значення , розглядаючи в якості усі інші точки. Якщо для однієї з них вказане вираження менше нуля, вважаємо наступною її і продовжуємо перевірку інших точок (аналогічно алгоритму пошуку мінімального елементу в масиві). Якщо ж значення вираження дорівнює нулю, то порівнюємо квадрати довжин векторів. В результаті за операцій чергова вершина опуклої оболонки буде знайдена. Продовжуючи цю процедуру, ми рано чи пізно повернемося до точки . Це означатиме, що опукла оболонка побудована.

При рішенні цієї задачі у разі спочатку цілочисельних координат ми повністю можемо уникнути застосування речової арифметики, а отже, позбавитися від втрати точності обчислень. Інакше в рішення можуть бути включені «зайві» точки, близькі до межі опуклої оболонки, або не враховані деякі з «потрібних» точок. Складність цього алгоритму складе , де – кількість точок в опуклій оболонці, у гіршому разі рівне .

Існує інший алгоритм рішення цієї задачі (алгоритм Грехема) з обчислювальною складністю , грунтований на попередньому сортуванні точок початкової великої кількості за значенням кута в полярній системі координат з центром в одній з точок опуклої оболонки. Тобто найбільш трудомістким завданням виявляється саме сортування вихідних точок. Сортування точок можна робити по знаку косого добутку , де – будь-яка вершина опуклої оболонки (наприклад, все та ж права нижня точка). У відсортованому масиві точок усі вказані твори мають бути невід'ємні. Точки з різними кутами () розташовуються в порядку збільшення довжин відповідних векторів .



Далі перегляд Грехема використовує стек, в якому зберігаються точки, що є кандидатами в опуклу оболонку. Спочатку в стек поміщається перша з відсортованих точок. Потім - сусідня з нею вершина опуклої оболонки. Якщо на першому з променів точок декілька, то це точка цього променя , найбільш віддалена від . Третя – точка . Нехай у вершині стоку знаходиться точка . Розглянемо наступну в порядку збільшення полярного кута точку початкової множини . Поки ділянкою ламаної не є опуклим, із стоку видаляється чергова точка . Потім поміщається в стік. У момент закінчення перегляду усіх точок в стоку знаходитимуться в точності усі вершини опуклої оболонки. Оскільки будь-яка точка додається в стік рівно один раз, то і віддаляється вона з нього не більше одного разу, тому час перегляду складає .

### Перевірка опуклості многокутника

Опуклість многокутника з вершинами , , …, , перерахованими в порядку його обходу, легко перевірити, якщо вичислити знаки косих добутків , (тут є , a ). У опуклого многокутника знаки усіх вказаних добутків або недодатні, або невід'ємні(тобто знаки ненульових творів співпадають). Якщо ми знаємо напрям обходу, то знак косих добутків для опуклого многокутника визначений: при обході за годинниковою стрілкою усі косі добутки недодатні, а проти годинникової стрілки - невід'ємні.

### Перевірка приналежності точки внутрішньої області многокутника

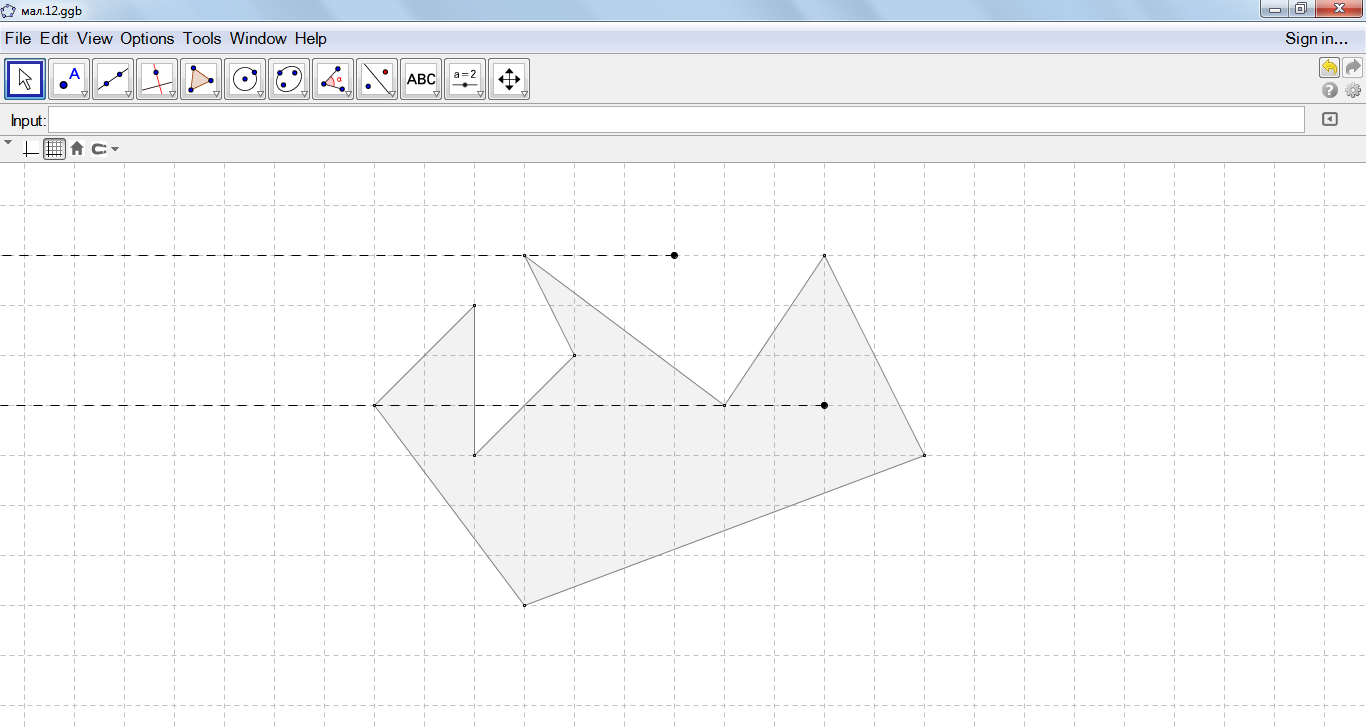
Нехай – деяка точка площини. Вимагається визначити її місцезнаходження відносно замкнутої ламаної, що є межею многокутника. Розглянемо спочатку випадок опуклого многокутника. Нехай задані своїми координатами вершини многокутника , , …, перераховані в порядку його обходу проти годинникової стрілки. При такому обході многокутник лежить ліворуч межі. І, значить, якщо точка лежить усередині многокутника, то орієнтований кут між векторами і негативний. Тому нам досить підрахувати величину косих добутків , ; значення береться по модулю . Якщо усі отримані при цьому значення від’ємні, то точка внутрішня. Якщо ж одне з них дорівнює нулю, а усі інші від’ємні, то належить межі многокутника (переконайтеся, що просто рівність нулю одного зі значень не досить). У протилежному випадку лежить поза нашим многокутником.

Розглянемо тепер довільний многокутник. Проведемо горизонтальний промінь з точки , наприклад, вліво. Оскільки многокутник обмежений, то завжди легко вказати на цьому промені точку , свідомо йому не прилеглу. Далі підрахуємо кількість перетинів відрізку з межею многокутника. Якщо воно дорівнює нулю або парне, то точка лежить поза многокутником, інакше - усередині нього.

Кількість перетинів відрізку з межею ми підрахуємо, розглянувши по черзі перетин відрізку з кожною з ланок ламаної. При цьому можливі наступні особливі випадки:

1. Одна з ланок ламаної прямої цілком міститься усередині відрізку .
2. Ланка ламаної торкається відрізку .
3. лежить на одній з ланок ламаної.

У останньому випадку належить межі многокутника, і в підрахунку загального числа перетинів необхідності немає. Для двох перших випадків поступимо таким чином. У першому випадку припинення ігноруватимемо. У другому - додатково перевіримо, "нижнім" або "верхнім" кінцем ланка ламаної торкається горизонтального відрізку . Якщо точкою дотику є "нижній" кінець ланки, то перетин ігнорується, а якщо "верхній" - то зачитується. З урахуванням цієї угоди торкання відрізку межі многокутника в одних точках ігнорується, а в інших точках вважається двічі. Це не змінить парності числа перетинів, а тільки вона важлива при пошуку відповіді на питання цього завдання. Якщо ж відрізок дійсно перетинає ламану в її вершині, то, за нашою угодою, число перетинів якраз збільшиться на одиницю (перетин з верхнім ребром зарахований не буде, а з нижнім - буде). Наприклад, на кількість перетинів для верхнього з досліджуваних точок буде рівна чотирьом (торкання зараховане двічі), а для нижньої точки - трьом (торкання не враховане, а перетин у вершині ламаній врахований один раз).



### Коло, що «охоплює» N точок площини

Це завдання полягає у відшукуванні координат центру кола мінімально можливого радіусу, усередині якої знаходяться усі задані точки. Іноді цю проблему називають мінімаксним завданням «про культурний центр». У ній вимагається по координатах будинків в місті підібрати місце для будівництва культурного центру так, щоб відстань до максимально віддаленого від нього будинку була мінімальною. Для того, щоб зрозуміти рішення цієї задачі в загальному випадку, розглянемо спочатку «трикутний» варіант: .

Навіть для трьох точок вид рішення залежить від їх взаємного розташування. Нехай точки лежать на одній прямій або утворюють тупокутний трикутник. Тоді шукана точка лежить на середині відрізку, що сполучає найбільш віддалені одна від однієї точки (в середині найбільшої сторони тупокутного трикутника). Насправді, відстань від цієї точки до будь-якої з перших двох зменшити не можна, а третя точка знаходиться на меншій відстані від знайденої точки, отже, вона лежить усередині кола, діаметр якого утворюють дві інші точки. А для гострокутного трикутника рішенням є центр описаного навколо нього кола (зміщення шуканої точки від нього у будь-якому напрямі приведе до збільшення відстані хоч би до однієї з точок). Прямокутний трикутник є «пограничним» для цих двох випадків, тобто для нього шукану точку можна знаходити будь-яким з описаних способів (звичайно, перший спосіб обчислювально більше за просту).

Для довільного N також є два випадки. Якщо знайдуться дві такі точки, що коло, побудоване на відрізку, що сполучає їх, як на діаметрі, містить усі інші точки (тобто для них виконується нерівність , де – центр кола), то це коло - шукане (фактично це випадок «тупокутного трикутника»). Якщо ж такої пари точок не знайшлося, то шукане коло свідомо проходить хоч би через три з вихідних точок. Тому тепер необхідно перебирати усі трійки точок до тих пір, поки не знайдеться така трійка, що коло, що проходить через ці точки, укладатиме усередині себе усі інші точки (випадок «гострокутного трикутника»).

## Теоретична частина з інформатики

Програма створена за допомогою мови програмування C#.

### Елементи управління

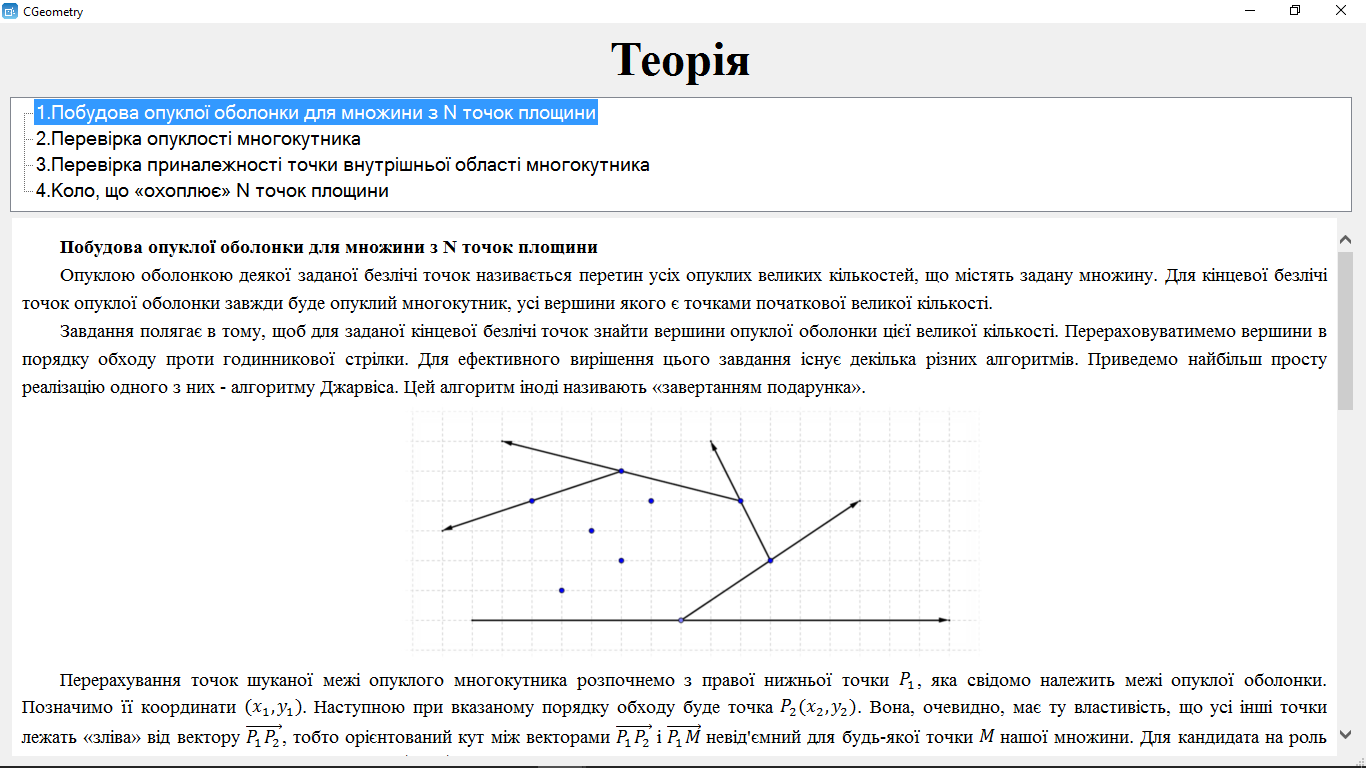
* *Window* – головне вікно, контейнер, в який поміщаються всі інші елементи управління;
* *ToolTip* – компонент, який відображає підказки при наведенні курсору миші на елементи керування;
* *PictureBox* – своєрідне полотно для виконання геометричних побудов;
* *GroopBox* - контейнер, у який можна помістити декілька компонентів;
* *Label* – елемент керування, який слугує для відображення тексту;
* *Button* – кнопка, при натисканні якої відбуваються деякі операції.
* *RichTextBox* – зчитує текстовий файл у форматі *.rtf;*
* *Click –* відбувається після кліку миші по елементу керування;
* *DoubleClick –* відбувається після подвійного кліку миші по елементу керування;
* *MouseDown/MouseUp –* відбувається при затисканій/відтисканій кнопці миші;
* *MouseMove –* відбувається при русі курсора по елементу керування.

## Опис роботи

Сайт: http://cgeometry.co.nf/.

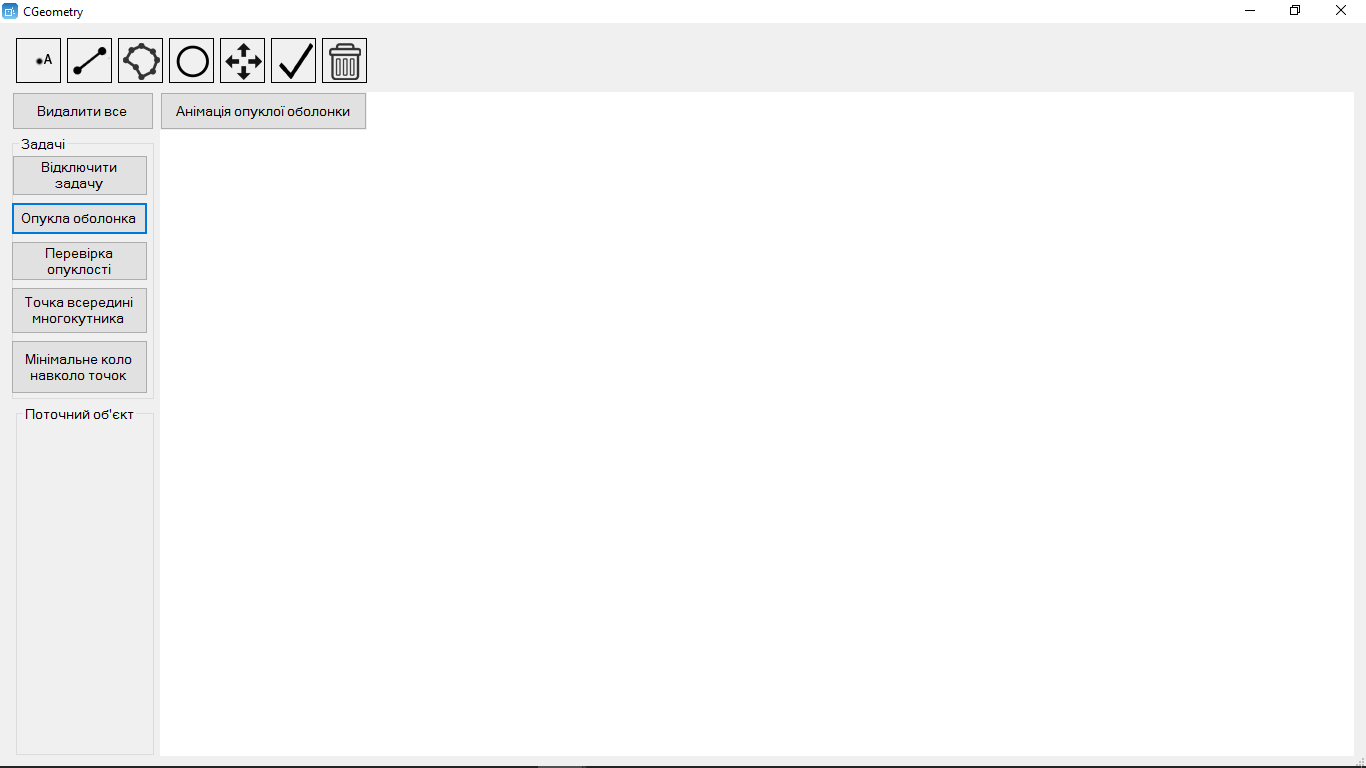
Сама робота складається з двох частин: «Теорія» та «Практика».

### «Теорія»



В частині «Теорія» знаходиться вся теорія з предмету яку я використовував при програмуванні алгоритмів модулювання задач обчислювальної геометрії.

### «Практика»



В частині «Практика» знаходяться елементарні побудови та змодульовані задачі обчислювальної геометрії.

## Посібник користувача

### Панель елементів

* Точка: Щоб побудувати точки необхідно в панелі елементів обрати режим "Точка" і розставити точки в потрібних місцях на полотні. Точки нумеруватимуться цілими числами;
* Відрізок: Щоб побудувати відрізок необхідно в панелі елементів обрати режим "Відрізок" і розставити дві точки в потрібних місцях на полотні. Точки нумеруватимуться літерами латинського алфавіту і цілими числами(A1, B1, ...). У панелі "Поточний об'єкт" буде вказуватися довжина відрізка;
* Многокутник: Щоб побудувати багатокутник потрібно в панелі елементів вибрати режим "Багатокутник" і розставити вершини багатокутника в потрібних місцях на полотні. Щоб завершити побудову багатокутника потрібно натиснути на кнопку "Завершити" в панелі елементів або двічі клікнути по полотну. Вершини багатокутника нумеруватимуться літерами латинського алфавіту і цілими числами(A1, B1, ...). У вікні "Поточний об'єкт" будуть вказуватися довжини сторін багатокутника і його опуклість;
* Коло: Щоб побудувати коло потрібно в панелі елементів вибрати режим "Коло" і розставити дві точки в потрібних місцях на полотні (перша - центр кола, друга - точка кола). У вікні "Поточний об'єкт" будуть вказуватися радіус кола і координати його центру;
* Перемістити: Щоб перемістити об'єкт потрібно в панелі елементів вибрати режим "Перемістити", навести курсор на об'єкт і перемістити його в потрібне місце, затиснувши кнопку миші;
* Закінчити: Щоб закінчити побудову об'єкта потрібно в панелі елементів натиснути кнопку "Закінчити";
* Видалити: Щоб видалити об'єкт потрібно в панелі елементів вибрати режим "Видалити", обрати об'єкт, який потрібно видалити і клікнути на нього.

### Панель задач

* Опукла оболонка: Щоб побудувати опуклу оболонку потрібно в панелі задач вибрати режим "Опукла оболонка" і розставити точки в потрібних місцях на полотні інструментом "Точка". Точки нумеруватимуться цілими числами. Оболонка буде будуватися автоматично. Після побудови опуклої оболонки слід в панеле завдань натиснути на кнопку "Відключити задачу";
* Перевірка опуклості: Щоб перевірити оболонку на опуклість потрібно в панелі задач вибрати режим "Перевірити на опуклість" і клікнути на багатокутник який потрібно перевірити на опуклість;
* Перевірка приналежності точки многокутнику: Щоб перевірити приналежність точки многокутнику потрібно в панелі задач вибрати режим "Точка всередині многокутника" і клікнути на місце розташування точки яку потрібно перевірити;
* Коло, що «охоплює» N точок площини: Щоб побудувати коло мінімального радіуса, що "охоплює" N точок площини потрібно в панелі задач обрати режим "Мінімальне коло навколо точок" і розставити точки в потрібних місцях на полотні. Коло будується автоматично. Після побудови кола слід в панеле завдань натиснути на кнопку "Відключити задачу".

## Код програми

### Клас “Dot” – клас точки

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CGeometry

{

// Класс "Dot" - класс точки

class Dot

{

int x, y; // Координаты точки (x; y)

public bool current = false; // Наведена ли мышь на текущую точку

private int size = 8; // Размер

public Line of\_line = null; // Обратная связь, через точку искать линию

// Для задачи выпуклой оболочки, находит только отдельные точки

public bool stable = true; // Можно ли "присоидениться" к даной точке

// Координата x точки

// Координата y точки

// size - размер на холсте

// stable - стабильна ли (не удалится ли ВНЕЗАПНО),

// нестабильные - точки-середины сторон многоугольника(удаляются при завершении построения многоугольника

public Dot(int x, int y, int size = 8, bool stable = true, Line line = null) //Создание самой точки

{

this.x = x;

this.y = y;

this.size = size;

this.stable = stable;

this.of\_line = line;

}

public Point Position // Позиции точки, типа Point

{

get // Если берем значение из параметра Position

{

return new Point(this.x, this.y);

}

set // Если присваиваем значение к параметру Position

{

this.x = value.X;

this.y = value.Y;

}

}

public void draw(Graphics gr, string name = null) // Рисование точки

{

SolidBrush brush; // Создаем новую кисть

if (current) // Если наведено

brush = new SolidBrush(Color.Green);

else

brush = new SolidBrush(Color.Black);

gr.FillEllipse(brush, this.Position.X - size / 2, this.Position.Y - size / 2, size, size); // Рисуется точка

if (name != null) //Если у точки есть имя - рисуем

{

gr.DrawString(name, new Font("arial", 10), Brushes.Black, this.Position);

}

}

private bool \_\_is\_near(Point point)

{

return Math.Abs(this.Position.X - 4 - point.X) < 10 && Math.Abs(this.Position.Y - 4 - point.Y) < 10;

// Считается что курсор около точки

}

public bool is\_near(Dot dot)

{

return \_\_is\_near(dot.Position);

}

public bool is\_near(Point point)

{

return \_\_is\_near(point);

}

// Вернёт копию точки

// Когда строятся паралельные и перепендикулярные прямые, копии

public Dot Copy()

{

return new Dot(this.Position.X, this.Position.Y, this.size);

}

// Информация о объекте для вывода на форму записываются параметры в ObjInfo

public string ObjInfo

{

get

{

string data = "";

data += (Form1.lang ? "Координати: " : "Координаты: ") + this.Position.ToString() + '\n'.ToString();

return data.ToString();

}

}

}

}

### Клас “Line” – клас прямої

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CGeometry

{

// Класс "Line" - класс прямой

class Line

{

public Dot[] dots = new Dot[2]; // Создается массив из 2-х точек

public Dot middle = new Dot(0, 0, 5); // Середина отрезка построеного из двух точек

private int current\_dot = 0; // Индекс масива

public bool readyLine = false; // Заполн ли массив точек (готова ли линия)

public bool current = false; // Наведена ли мышь на текущую линию

public Line()

{

}

public void AddPoint(Point point) // Добавление точки при клике на полотно

{

Dot dot = new Dot(point.X, point.Y); // Создание новой точки

dots[current\_dot] = dot; // Добавление точки в массив точек

if (current\_dot == 1)

{

readyLine = true;

}

current\_dot = Math.Min(current\_dot + 1, 1); // Чтобы точек не было больше 2-х

}

// Отрисовывает прямую.

public void draw(Graphics gr, int index = 0)

{

// Создает новый массив точек и присваивает в него точки из массива dots

Point[] points = new Point[dots.Length];

for (int i = 0; i < dots.Length; i++)

{

if (dots[i] != null) // Если есть координаты точки

points[i] = dots[i].Position;

}

// Если прямая готова

if (readyLine)

{

if(!current) // Если не выделена

gr.DrawLine(Pens.Brown, points[0], points[1]);

else

gr.DrawLine(Pens.DarkOrange, points[0], points[1]);

update\_middle(); // Метод обновления середины отрезка, ниже

}

for (int i = 0; i < dots.Length; i++) // Рисуются точки

{

if (dots[i] != null) // Если точки существуют

if (index == 0) // Не учитаывем приставку "0" (А0, Б0, ...)

dots[i].draw(gr, name: ((char)('A' + i)).ToString());

else

dots[i].draw(gr, name: ((char)('A' + i)).ToString() + index.ToString());

}

}

// Рисует середины прямых

public void drawMiddles(Graphics gr)

{

if(readyLine)

middle.draw(gr);

}

// Вычисление расстояния от произвольной точки до текущей прямой

private int FindDistanceToSegment(PointF pt)

{

Point p1, p2; // Точки отрезка

Point closest; // Ближайшая точка на отрезке до произвольной точки

p1 = dots[0].Position;

p2 = dots[1].Position;

float dx = p2.X - p1.X; // Растояние Х между точками прямой

float dy = p2.Y - p1.Y; // Растояние У между точками прямой

if ((dx == 0) && (dy == 0))

{

closest = p1;

dx = pt.X - p1.X;

dy = pt.Y - p1.Y;

return (int)Math.Sqrt(dx \* dx + dy \* dy); // Возвращает растояние

}

// Расчитывает значение t, которое является минимальным расстоянием от произвольной точки до прямой

float t = ((pt.X - p1.X) \* dx + (pt.Y - p1.Y) \* dy) / (dx \* dx + dy \* dy);

// Рассматриваем если конец этой прямой

// конец отрезка или его середина

if (t < 0) // Если t ближе к первой точке

{

closest = new Point(p1.X, p1.Y);

dx = pt.X - p1.X;

dy = pt.Y - p1.Y;

}

else if (t > 1) // Если t ближе ко второй точке

{

closest = new Point(p2.X, p2.Y);

dx = pt.X - p2.X;

dy = pt.Y - p2.Y;

}

else

{

closest = new Point((int)(p1.X + t \* dx), (int)(p1.Y + t \* dy));

dx = pt.X - closest.X;

dy = pt.Y - closest.Y;

}

return (int)Math.Sqrt(dx \* dx + dy \* dy);

}

// Попадает ли мышка в область прямой

// target\_point - точка для проверки

public bool PointIsInLine(Point target\_point)

{

return FindDistanceToSegment(target\_point) < 5; //5 - "погрешность" наведения мышки в пикселях, в две стороны

}

// Вернёт копию прямой

// Когда строятся паралельные и перепендикулярные прямые, копии

public Line Copy()

{

Line line = new Line();

line.dots[0] = this.dots[0].Copy();

line.dots[1] = this.dots[1].Copy();

line.readyLine = this.readyLine;

return line;

}

//Обновление середины

public void update\_middle()

{

middle.Position = new Point((dots[0].Position.X + dots[1].Position.X) / 2, (dots[0].Position.Y + dots[1].Position.Y) / 2);

}

// Сдвигает прямую на расстояние offset(на вектор offset)

// offset - точка на которой находится мишь при передвижении

public void SetPosition(Point offset)

{

this.dots[0].Position = new Point(this.dots[0].Position.X - offset.X, this.dots[0].Position.Y - offset.Y);

this.dots[1].Position = new Point(this.dots[1].Position.X - offset.X, this.dots[1].Position.Y - offset.Y);

}

// Длина отрезка

public int Lenght

{

get

{

return (int)Math.Sqrt((this.dots[0].Position.X - this.dots[1].Position.X) \* (this.dots[0].Position.X - this.dots[1].Position.X) + (this.dots[0].Position.Y - this.dots[1].Position.Y) \* (this.dots[0].Position.Y - this.dots[1].Position.Y));

}

}

// Информация об объекте для вывода на форму записываются параметры в ObjInfo

public string ObjInfo

{

get

{

string data = "";

data += (Form1.lang ? "Довжина: " : "Длина: ")+ " " + this.Lenght.ToString() + '\n'.ToString();

return data.ToString();

}

}

}

}

### Клас “Polygon” – клас многокутника

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Drawing.Drawing2D;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CGeometry

{

// Класс "Polygon" - класс многоугольника

class Polygon

{

public List<Dot> dots = new List<Dot>(); // Новый список(коллекция) точек

public List<Dot> middles = new List<Dot>(); // Новый список(коллекция) точек-середин

public bool current = false; // Наведена ли мышь на текущий многоугольник

public Polygon()

{

}

public void AddPoint(Point point) // Добавление точки в список

{

Dot dot = new Dot(point.X, point.Y);

dots.Add(dot);

}

public void draw(Graphics gr, int index = 0)// Рисует полигон

{

if (dots.Count >= 3)

{

// Рисуем полигон

Point[] points = new Point[dots.Count];

for (int i = 0; i < dots.Count; i++) // Преобразуем "Dot" в позицию

points[i] = dots[i].Position;

if(!current) // Если не выделена

gr.FillPolygon(new SolidBrush(Color.FromArgb(30, 255, 0, 0)), points);

else

gr.FillPolygon(new SolidBrush(Color.FromArgb(40, 255, 10, 10)), points);

gr.DrawPolygon(Pens.Brown, points); // Контур полигона

// Пересчитываем середины сторон

middles.Clear();

for (int i = 0; i < dots.Count - 1; i++) // Запоминаем две соседные точки, вычисляем середину

middles.Add(new Dot((dots[i].Position.X + dots[i + 1].Position.X) / 2, (dots[i].Position.Y + dots[i + 1].Position.Y) / 2, 4, stable: false));

// Учитываем первую и последнюю точки

middles.Add(new Dot((dots[0].Position.X + dots[dots.Count - 1].Position.X) / 2, (dots[0].Position.Y + dots[dots.Count - 1].Position.Y) / 2, 4, stable: false));

}

foreach (Dot dot in dots) // Рисуем точки

if(index == 0) // Не учитаывем приставку "0" (А0, Б0, ...)

dot.draw(gr, name:((char)('A' + dots.IndexOf(dot))).ToString());

else

dot.draw(gr, name: ((char)('A' + dots.IndexOf(dot))).ToString() + index.ToString());

}

// Рисует середины прямых

public void drawMiddles(Graphics gr)

{

foreach (Dot dot in middles)

dot.draw(gr);

}

// Проверка вхождения точки в полигон

// polygon - точки полигона

// target\_point - точка для проверки

// true - если точка входит в полигон

public bool PointIsInPolygon(Point target\_point)

{

// Создает GraphicsPath содержащий полигон.

GraphicsPath path = new GraphicsPath();

Point[] points = new Point[dots.Count]; //Вершины полигона

for (int i = 0; i < dots.Count; i++)

points[i] = dots[i].Position;

path.AddPolygon(points);

return path.IsVisible(target\_point);

}

// Сдвигает прямую на расстояние offset(на вектор offset)

// offset - точка на которой находится мишь при передвижении

public void SetPosition(Point offset)

{

foreach(Dot dot in this.dots)

dot.Position = new Point(dot.Position.X - offset.X, dot.Position.Y - offset.Y);

}

// Возвращает расстояние между точками i и i + 1

public int Lenght(Dot a, Dot b)

{

return (int)Math.Sqrt((a.Position.X - b.Position.X) \* (a.Position.X - b.Position.X) + (a.Position.Y - b.Position.Y) \* (a.Position.Y - b.Position.Y));

}

// Вернёт копию многоугольника

public Polygon Copy()

{

Polygon other = new Polygon();

foreach (Dot dot in dots)

other.dots.Add(dot.Copy());

return other;

}

//

//

//

// Вычисление выпуклости многоугольника

//

//

//

// Возвращает векторное произведение AB x BC.

// Векторное произведение - это вектор который перпендикулярен AB

// и BC он имеет длину |AB| \* |BC| \* Sin(theta) и

// направление по правилу by the right-hand rule.

// For two vectors in the X-Y plane, the result is a

// vector with X and Y components 0 so the Z component

// gives the vector's length and direction.

public static float CrossProductLength(float Ax, float Ay, float Bx, float By, float Cx, float Cy)

{

// Считает координаты вектора.

float BAx = Ax - Bx;

float BAy = Ay - By;

float BCx = Cx - Bx;

float BCy = Cy - By;

// Calculate the Z coordinate of the cross product.

return (BAx \* BCy - BAy \* BCx);

}

// Выпуклость

// true - если выпуклый

public bool PolygonIsConvex()

{

// Для каждого набора из трех соседних точек A, B, C,

// находим векторное произведение AB · BC. Если

// знак всех этих произведений в порядке обхода одинаков, углы

// все положительные или отрецательные (зависит

// от направления обхода многоугольника) то полигон выпуклый.

bool got\_negative = false;

bool got\_positive = false;

int num\_points = this.dots.Count;

int B, C;

for (int A = 0; A < num\_points; A++)

{

B = (A + 1) % num\_points;

C = (B + 1) % num\_points;

float cross\_product = CrossProductLength(dots[A].Position.X, dots[A].Position.Y, dots[B].Position.X, dots[B].Position.Y, dots[C].Position.X, dots[C].Position.Y);

if (cross\_product < 0)

{

got\_negative = true;

}

else if (cross\_product > 0)

{

got\_positive = true;

}

if (got\_negative && got\_positive)

return false;

}

// If we got this far, the polygon is convex.

return true;

}

// Информация о объекте для вывода на форму записываются параметры в ObjInfo

public string ObjInfo

{

get

{

string data = "";

data += (Form1.lang ? "Довжини сторін:" : "Длины сторон:") + "\n" ;

for(int i = 0; i < dots.Count - 1; i++) {

data += (Form1.lang ? "Довжина " : "Длина ") + ((char)('A' + i)).ToString() + ((char)('A' + i + 1)).ToString() + ": " + this.Lenght(dots[i], dots[i + 1]).ToString() + '\n'.ToString();

}

data += (Form1.lang ? "Довжина " : "Длина ") + ('A').ToString() + ((char)('A' + dots.Count - 1)).ToString() + ": " + this.Lenght(dots[0], dots[dots.Count - 1]).ToString() + '\n'.ToString();

data += (Form1.lang ? "Опуклий: " : "Выпуклый: ") + ( this.PolygonIsConvex() ? (Form1.lang ? "так." : "да.") : (Form1.lang ? "ні." : "нет."));

return data.ToString();

}

}

}

}

### Клас “Circle” – клас кола

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CGeometry

{

// Класс "Circle" - класс окружности(панель элементов)

class Circle

{

public int radius // Радиус окружности

{

get

{

return (int)Math.Sqrt((center.Position.X - point2.Position.X) \* (center.Position.X - point2.Position.X) + (center.Position.Y - point2.Position.Y) \* (center.Position.Y - point2.Position.Y));

}

}

public Dot center = new Dot(0, 0); // Центр окружности

public Dot point2 = new Dot(0, 0); // Точка на окружности

public SolidBrush brush = new SolidBrush(Color.FromArgb(30, 0, 255, 0)); // Кисть

public bool ready = false; // Построение завершено(т.е. задана ли 2-я точка)

public bool centerDone = false; // Задан ли центр окружности

public bool current = false; // Наведена ли мышь на текущую окружность

public Circle()

{

}

// Отрисовывает круг

public void draw(Graphics gr, int index = 0)

{

if (centerDone) // Если центер существует

{

if (!current) // Если не выделена

{

gr.DrawLine(Pens.Brown, center.Position, point2.Position);

gr.FillEllipse(brush, center.Position.X - radius, center.Position.Y - radius, radius \* 2, radius \* 2);

gr.DrawEllipse(Pens.Green, center.Position.X - radius, center.Position.Y - radius, radius \* 2, radius \* 2);

}

else

{

gr.FillEllipse(new SolidBrush(Color.FromArgb(40, 0, 255, 0)), center.Position.X - radius, center.Position.Y - radius, radius \* 2, radius \* 2);

gr.DrawEllipse(Pens.Green, center.Position.X - radius, center.Position.Y - radius, radius \* 2, radius \* 2);

gr.DrawLine(Pens.DarkOrange, center.Position, point2.Position);

}

center.draw(gr);

point2.draw(gr);

}

}

// Попадает ли мышка в область окружности

// target\_point - точка для проверки

public bool PointIsInCircle(Point target\_point)

{

return ((target\_point.X - center.Position.X) \* (target\_point.X - center.Position.X) + (target\_point.Y - center.Position.Y) \* (target\_point.Y - center.Position.Y) < radius \* radius);

}

// Вернёт копию окружности

public Circle Copy()

{

Circle other = new Circle();

return other;

}

// Сдвигает окружность на расстояние offset(на вектор offset)

// offset - точка на которой находится мишь при передвижении

public void SetPosition(Point offset)

{

this.center.Position = new Point(this.center.Position.X - offset.X, this.center.Position.Y - offset.Y);

this.point2.Position = new Point(this.point2.Position.X - offset.X, this.point2.Position.Y - offset.Y);

}

// Информация о объекте для вывода на форму записываются параметры в ObjInfo

public string ObjInfo

{

get

{

string data = "";

data += (Form1.lang ? "Радіус: " : "Радиус: ") + this.radius.ToString() + "\n";

data += (Form1.lang ? "Координати центра" : "Координаты центра") + this.center.Position.ToString();

return data.ToString();

}

}

}

}

### Клас “MyPoint” – клас точки для задачі «Мінімальне коло навколо точок»

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace SmallestEnclosingCircle

{

public class MyPoint

{

// Координата Х MyPoint

private double x;

// Координата У MyPoint

private double y;

// Construct a MyPoint at the origin

public MyPoint()

{

x = 0;

y = 0;

}

// Конструктор MyPoint для расположения (xVal, yVal)

public MyPoint(double xVal, double yVal)

{

x = xVal;

y = yVal;

}

// Получаем координату Х MyPoint

public double getX()

{

return x;

}

// Получаем координату У MyPoint

public double getY()

{

return y;

}

// Расчитываем расстояние между точками MyPoints

public double distance(MyPoint MyPoint)

{

double dx = x - MyPoint.x;

double dy = y - MyPoint.y;

return Math.Sqrt(dx \* dx + dy \* dy);

}

// Возвращам расположение MyPoint как string

public String toString()

{

return "(" + x + "," + y + ")";

}

}

}

### Клас “MyCircle” – клас точки для задачі «Мінімальне коло навколо точок»

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Drawing;

using System.Windows.Forms;

namespace SmallestEnclosingCircle

{

class MyCircle

{

MyPoint Center; // Центр окружности

double radius; //Радиус окружности

public MyCircle() // Создает пустую окружность

{

this.Center = new MyPoint(0, 0);

radius = 0;

}

public MyCircle(MyPoint Center, double radius) // Создает окружность по центру и радиусу

{

this.Center = Center;

this.radius = radius;

}

public MyCircle(MyPoint Center) // Создает окружность нулевого радиуса

{

this.Center = Center;

this.radius = 0;

}

public MyCircle(MyPoint first, MyPoint second) // Создает окружность по двум точкам

{

Center = new MyPoint((first.getX() + second.getX()) / 2, (first.getY() + second.getY()) / 2);

radius = Center.distance(first);

}

public MyCircle(MyPoint p1, MyPoint p2, MyPoint p3) // Создает окружность по трем точкам(описаная около треугольника)

{

double x = (p3.getX() \* p3.getX() \* (p1.getY() - p2.getY()) + (p1.getX() \* p1.getX() + (p1.getY() - p2.getY()) \* (p1.getY() - p3.getY()))

\* (p2.getY() - p3.getY()) + p2.getX() \* p2.getX() \* (-p1.getY() + p3.getY()))

/ (2 \* (p3.getX() \* (p1.getY() - p2.getY()) + p1.getX() \* (p2.getY() - p3.getY()) + p2.getX() \* (-p1.getY() + p3.getY())));

double y = (p2.getY() + p3.getY()) / 2 - (p3.getX() - p2.getX()) / (p3.getY() - p2.getY()) \* (x - (p2.getX() + p3.getX()) / 2);

Center = new MyPoint(x, y);

radius = Center.distance(p1);

}

public Point getP1() // Возвращает первую точку (центр окружности)

{

return new Point((int)(Center.getX()), (int)Center.getY());

}

public Point getP2() // Возвращает вторую точку (точку окружности)

{

return new Point((int)(Center.getX() + radius), (int)Center.getY()) ;

}

public int belongsToCircle(MyPoint other)

{

if ((Center.distance(other) - radius) < 0.000001) return 0;

if (Center.distance(other) > radius) return -1;

return 1;

}

}

}

### Клас “Geometry” – клас збору всього до купи

using SmallestEnclosingCircle;

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Drawing.Drawing2D;

using System.Linq;

using System.Text;

using System.Threading;

using System.Windows.Forms;

namespace CGeometry

{

// Класс "Geometry"

// В этом классе идет только вызов других классов, подсчет точек, прямых и т.д.

class Geometry

{

public List<Polygon> polygons; // Новый список(коллекция) многоугольников

public List<Line> lines; // Новый список(коллекция) линий

public List<Circle> circles; // Новый список(коллекция) окружностей

public List<Dot> dots\_list; // Новый список(коллекция) точек

public List<Dot> tempdots = new List<Dot>();// Новый список(коллекция) точек пересечения прямых содержащих отрезки

public List<Circle> tempcircles = new List<Circle>();// Окружности для последней задачи

public List<Point> pnts = new List<Point>();// Точки для создания опуклой оболочки

public Graphics gr;

public Geometry(Graphics gr) // Создание пустых объектов

{

polygons = new List<Polygon>();

lines = new List<Line>();

circles = new List<Circle>();

dots\_list = new List<Dot>();

this.gr = gr;

this.gr.SmoothingMode = SmoothingMode.AntiAlias; // Включаем сглаживание

}

public void updategr(Graphics gr) // При смене размера формы пересоздается битмап и обновляеться графикс,

// затем новый графикс передается в эту процедуру и сохраняется

{

this.gr = gr;

this.gr.SmoothingMode = SmoothingMode.AntiAlias; //Включаем сглаживание

}

public void draw() // Рисует все элементы и увеличивает elem\_index

{

int elem\_index = 0; // Добавляем к названию точкек цифры (А1, Б2...)

gr.Clear(Color.White);

DrawLines(pnts); // Для создания опуклой оболочки

foreach (Polygon poly in polygons) // Многоуголники

{

poly.draw(gr, elem\_index);

elem\_index++;

}

foreach (Line line in lines) // Отрезки

{

line.draw(gr, elem\_index);

elem\_index++;

}

foreach (Circle circle in circles) // Окружности

{

circle.draw(gr, elem\_index);

elem\_index++;

}

foreach (Dot dot in dots\_list) // Точки

{

dot.draw(gr, (elem\_index + 1).ToString());

elem\_index++;

}

foreach (Dot dot in tempdots)

{

dot.draw(gr);

elem\_index++;

}

foreach (Circle dot in tempcircles) // Точки для последней задачи про минимальную окружность

{

dot.draw(gr);

}

}

// Возвращает все точки на холсте.

// flag Возвращать ли точки-середины прямых

// unstable Возвращать ли точки-середины многоугольников

public List<Dot> dots(bool flag)

{

List<Dot> dots = new List<Dot>();

foreach (Polygon poly in polygons) // Добавляем все точки из полигонов в лист(коллекцию)

dots.AddRange(poly.dots);

foreach (Line line in lines) // Добавляем все точки из линий в лист(коллекцию)

dots.AddRange(line.dots);

foreach (Circle circle in circles) // Добавляем все точки из кругов

{

dots.Add(circle.center);

dots.Add(circle.point2);

}

foreach (Dot dot in dots\_list) //

dots.Add(dot);

dots.AddRange(tempdots.ToArray());

if (flag) // Если нужны "серединные точки"

{

foreach (Line line in lines) // Добавляем все точки из линий

dots.Add(line.middle);

foreach (Polygon poly in polygons) // Добавляем все точки из полигонов

dots.AddRange(poly.middles);

}

List<Dot> cleared\_dots = new List<Dot>();

for (int i = 0; i < dots.Count; i++) // Очищаем от пустых точк

if (dots[i] != null)

cleared\_dots.Add(dots[i]);

return cleared\_dots;

}

// Отрисовывает середины линий

// flag Определяет, нужно ли рисовать точки-середины прямых.

// Зависит от параметра programMode в форме.

public void DrawMiddles(bool flag)

{

if (!flag)

return;

foreach (Line line in lines)

line.drawMiddles(gr);

foreach (Polygon poly in polygons)

poly.drawMiddles(gr);

}

// Удаление полигона

public void RemovePoly(Polygon poly)

{

this.polygons.Remove(poly);

}

// Удаление круга

public void RemoveCircle(Circle circle)

{

this.circles.Remove(circle);

}

// Удаление линий

public void RemoveLine(Line line)

{

this.lines.Remove(line);

}

//

//

////Поворот одной точки относительно другой

//// pointToRotate - точка, которую нужно перевернуть.

//// centerPoint - точка относительно которой нужно поварачивать pointToRotate.

//// angleInDegrees The rotation angle in degrees.

//public Point RotatePoint(Point pointToRotate, Point centerPoint, double angleInDegrees)

//{

// double angleInRadians = angleInDegrees \* (Math.PI / 180);

// double cosTheta = Math.Cos(angleInRadians);

// double sinTheta = Math.Sin(angleInRadians);

// return new Point

// {

// X = (int)(cosTheta \* (pointToRotate.X - centerPoint.X) - sinTheta \* (pointToRotate.Y - centerPoint.Y) + centerPoint.X),

// Y = (int)(sinTheta \* (pointToRotate.X - centerPoint.X) + cosTheta \* (pointToRotate.Y - centerPoint.Y) + centerPoint.Y)

// };

//}

//

//

// Рисует прямые для задачи выпуклой оболчки

public void DrawLines(List<Point> points)

{

if (points.Count >= 3)

{

gr.DrawLines(Pens.Black, points.ToArray());

gr.DrawLine(Pens.Black, points[0], points.Last());

}

}

// Алгоритм оболочки из интернетов

// http://stackoverflow.com/questions/10020949/gift-wrapping-algorithm

//

// Находит самую левую точку, затем находит самую левую точку из оставшихся...

public List<Point> ConvexHull(List<Point> points)

{

if (points.Count < 3)

{

return new List<Point>();

}

List<Point> hull = new List<Point>();

// Находим самую левую точку

Point vPointOnHull = points.Where(p => p.X == points.Min(min => min.X)).First();

Point vEndpoint;

do

{

hull.Add(vPointOnHull);

vEndpoint = points[0];

for (int i = 1; i < points.Count; i++)

{

if ((vPointOnHull == vEndpoint) || (Orientation(vPointOnHull, vEndpoint, points[i]) == -1))

{

vEndpoint = points[i];

}

}

vPointOnHull = vEndpoint;

}

while (vEndpoint != hull[0]);

return hull;

}

public static int Orientation(Point p1, Point p2, Point p)

{

// Determinant

int Orin = (p2.X - p1.X) \* (p.Y - p1.Y) - (p.X - p1.X) \* (p2.Y - p1.Y);

if (Orin > 0)

return -1; // (\* Orientaion is to the left-hand side \*)

if (Orin < 0)

return 1; // (\* Orientaion is to the right-hand side \*)

return 0; // (\* Orientaion is neutral aka collinear \*)

}

// Возвращает точку пересечения двух прямых

// p1\_1 Первая точка прямой 1

// p1\_2 Вторая точка прямой 1

// p2\_1 Первая точка прямой 2

// p2\_2 Вторая точка прямой 2

// state -1 - если параллельны, 0 - если совпадают, 1 - если пересекаются, -2 - если ошибка

//

public Point GetIntersectionPointOfTwoLines(Line line1, Line line2, out int state)

{

Point p1\_1, p1\_2, p2\_1, p2\_2;

p1\_1 = line1.dots[0].Position;

p1\_2 = line1.dots[1].Position;

p2\_1 = line2.dots[0].Position;

p2\_2 = line2.dots[1].Position;

state = -2;

Point result = new Point();

// Если знаменатель (n) равен нулю, то прямые параллельны.

// Если и числитель (m или w) и знаменатель (n) равны нулю, то прямые совпадают.

// Если нужно найти пересечение отрезков, то нужно лишь проверить, лежат ли ua и ub на промежутке [0,1].

// Если какая-нибудь из этих двух переменных 0 <= ui <= 1, то соответствующий отрезок содержит точку пересечения.

// Если обе переменные приняли значения из [0,1], то точка пересечения прямых лежит внутри обоих отрезков.

float m = ((p2\_2.X - p2\_1.X) \* (p1\_1.Y - p2\_1.Y) - (p2\_2.Y - p2\_1.Y) \* (p1\_1.X - p2\_1.X));

float w = ((p1\_2.X - p1\_1.X) \* (p1\_1.Y - p2\_1.Y) - (p1\_2.Y - p1\_1.Y) \* (p1\_1.X - p2\_1.X));

float n = ((p2\_2.Y - p2\_1.Y) \* (p1\_2.X - p1\_1.X) - (p2\_2.X - p2\_1.X) \* (p1\_2.Y - p1\_1.Y));

float Ua = m / n;

float Ub = w / n;

if ((n == 0) && (m != 0))

{

state = -1; // Прямые параллельны (не имеют пересечения)

}

else if ((m == 0) && (n == 0))

{

state = 0; // Прямые совпадают

}

else

{

// Прямые имеют точку пересечения

result.X = (int)(p1\_1.X + Ua \* (p1\_2.X - p1\_1.X));

result.Y = (int)(p1\_1.Y + Ua \* (p1\_2.Y - p1\_1.Y));

// Проверка попадания в интервал

bool a = result.X >= p1\_1.X;

bool b = result.X <= p1\_1.X;

bool c = result.X >= p2\_1.X;

bool d = result.X <= p2\_1.X;

bool e = result.Y >= p1\_1.Y;

bool f = result.Y <= p1\_1.Y;

bool g = result.Y >= p2\_1.Y;

bool h = result.Y <= p2\_1.Y;

if (((a || b) && (c || d)) && ((e || f) && (g || h)))

{

state = 1; // Прямые имеют точку пересечения

}

}

return result; // Возвращает точку пересечения

}

// Минимальна окружность

public MyCircle findSec(int n, MyPoint[] p, int m, MyPoint[] b, bool log = false)

{

MyCircle sec = new MyCircle();

// Вычисление круга

if (m == 1)

{

sec = new MyCircle(b[0]);

}

// Если находятся две такие точки

else if (m == 2)

{

sec = new MyCircle(b[0], b[1]);

}

// Если находятся три такие точки

else if (m == 3)

{

MyCircle ms = new MyCircle(b[0], b[1], b[2]);

Circle c = new Circle();

c.center.Position = ms.getP1();

c.point2.Position = ms.getP2();

c.centerDone = true;

tempcircles.Add(c);

return ms;

}

// Если все точки MyPoints включены

string points = "";

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

{

// Если MyPoint снаружи окружности

if (sec.belongsToCircle(p[i]) == -1)

{

points = "";

foreach (MyPoint pnt in b)

{

if (pnt != null)

points += pnt.toString() + ", ";

}

// B <--- P[i].

b[m] = new MyPoint(p[i].getX(), p[i].getY());

// Recurse

sec = findSec(i, p, m + 1, b);

}

}

points = "";

foreach (MyPoint pnt in b)

{

if (pnt != null)

points += pnt.toString() + ", ";

}

return sec;

}

}

}

### Код “Form”

using SmallestEnclosingCircle;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading;

using System.Windows.Forms;

namespace CGeometry

{

public partial class Form1 : Form

{

Geometry geometry; // Элемент класса геометрия

Bitmap data; // Битман (для ПБ)

Dot dragged = null; // Можно ли перемещать

dynamic draggedFigure = null; // Текущий меремещаемый объект

dynamic copied = null; // Текщий копируемый объект

int count = 0;

enum Zadacha // Перечисление задач вычислительной геометрии

{

none, // Нет задач

obolochka\_is\_tochek, // Построение оболочки

convex, // Проверка опуклости

inside, // Точка внутри полигона

minimalCircle // Окружность минимального радиуса

}

private Zadacha \_\_zadacha = Zadacha.none;

Zadacha zadacha

{

get

{

return this.\_\_zadacha;

}

set

{

this.\_\_zadacha = value;

switch (value)

{

case Zadacha.none:

this.inf.Text = "";

this.geometry.pnts = new List<Point>();

break;

case Zadacha.obolochka\_is\_tochek:

this.inf.Text = lang ? "Задача побудови опуклої оболочки з точок. Розставити точки в потрібних місцях на площині інструментом 'Точка'. Оболонка буде будуватися автоматично." : "Задача постоения выпуклой оболочки из точек. Поставьте точки на нужных местах инструментом 'Точка'. Оболочка строится автоматически.";

break;

case Zadacha.convex:

this.inf.Text = lang ? "Перевірка многокутника на опуклість. Натисніть клавішою миші на многокутник який потрібно перевірити на опуклість." : "Проверка многоугольника на выпуклость. Нажмите левой клавишей мыши на многоугольник который вы хотите проверить.";

break;

case Zadacha.inside:

this.inf.Text = lang ? "Перевірка приналежності точки многокутнику. Натисніть клавішою миші на місце розташування точки яку потрібно перевірити." : "Проверка попадания точки внутрь многоугольника. Нажмите левой клавишей мыши на место, которое вы хотите проверить.";

break;

case Zadacha.minimalCircle:

this.inf.Text = lang ? "Задача побудови кола мінімального радіуса, котре охоплює всі точки площини. Розставити точки в потрібних місцях на площині інструментом 'Точка'. Коло будується автоматично." : "Задача постоения минимальной окружности вокруг точек. Поставьте точки на нужных местах инструментом 'Точкa'. Окружность строится автоматически.";

break;

}

}

}

enum Mode // Перечесление элементарных построений

{

drawPoly, // Построение полигона

drawLine, // Построение отрезка

normal, // Обычный режим

dragging, // Перетаскивания точек

deleting, // Удаления фигур

figureDrag, // Перемещение элементов целиком

drawCircle, // Построение окружностей

addDot // Построение точек

}

private Mode \_\_programMode = Mode.normal;

Mode programMode

{

get

{

return this.\_\_programMode;

}

set

{

this.\_\_programMode = value;

switch (value)

{

case Mode.drawPoly:

blank.Cursor = System.Windows.Forms.Cursors.Cross;

this.inf.Text = lang ? "Режим побудови многокутника. Розставте вершини многокутника в потрібних місцях. Щоб завершити побудову многокутника потрібно натиснути на кнопку 'Завершити' в панелі елементів або двічі клікнути на площину." : "Режим построения многоугольника. Расставьте точки в нужных местах. Завершить фигуру - двойной клик или нажание на кнопку 'Закончить'.";

break;

case Mode.addDot:

blank.Cursor = System.Windows.Forms.Cursors.Cross;

this.inf.Text = lang ? "Режим побудови точок. Розставте точки в потрібних місцях." : "Режим построения точек. Расставьте точки в нужных местах.";

break;

case Mode.drawLine:

blank.Cursor = System.Windows.Forms.Cursors.Cross;

this.inf.Text = lang ? "Режим побудови відрізків. Розставте дві точки в потрібних місцях." : "Режим построения отрезков. Расставьте две точки в нужных местах.";

break;

case Mode.drawCircle:

blank.Cursor = System.Windows.Forms.Cursors.Cross;

this.inf.Text = lang ? "Режим побудови кіл. Розставте дві точки в потрібних місцях (перша - центр кола, друга - точка кола)." : "Режим построения окружности. Розставьте две точки в нужных местах (первая - центр окружности, вторая - точка окружности).";

break;

case Mode.normal:

blank.Cursor = System.Windows.Forms.Cursors.Arrow;

this.inf.Text = "";

break;

case Mode.deleting:

this.inf.Text = lang ? "Режим видалення. Виберіть елемент, котрий необхідно видалити і натисніть на нього." : "Режим удаления. Выберите элемент, который вы хотите удалить и нажмите на него мышкой.";

this.current\_obj\_info.Text = "";

blank.Cursor = System.Windows.Forms.Cursors.Hand;

break;

case Mode.figureDrag:

this.inf.Text = lang ? "Режим переміщення фігур. Наведіть курсор на об'єкт і перемістить його в потрібне місце, затиснувши кнопку миші" : "Режим перемещения фигур. Наведите мышь на нужную фигуру и переместите её, зажав кнопку мыши.";

blank.Cursor = System.Windows.Forms.Cursors.Hand;

break;

}

// Удаляем линии, которые пользователь недорисовал

List<Line> toDelete = new List<Line>();

foreach (Line line in geometry.lines)

if (!line.readyLine)

toDelete.Add(line);

foreach (Line line in toDelete)

geometry.lines.Remove(line);

// Аналогичто с полигонами

List<Polygon> toDeletePoly = new List<Polygon>();

foreach (Polygon poly in geometry.polygons)

if (poly.dots.Count < 3)

toDeletePoly.Add(poly);

foreach (Polygon poly in toDeletePoly)

geometry.polygons.Remove(poly);

}

}

public Form1()

{

InitializeComponent();

this.BringToFront();

this.Focus();

}

// Изменить размер холста при изменении размера формы

private void Form1\_Resize(object sender, EventArgs e)

{

data = new Bitmap(Math.Max(10, blank.Width), Math.Max(10, blank.Height)); // Чтобы не выкидывало ошибку

geometry.updategr(Graphics.FromImage(data));

}

// Перерисовка программы

public void update()

{

geometry.tempdots.Clear();

for (int i = 0; i < geometry.lines.Count; i++)

{

for (int j = i + 1; j < geometry.lines.Count; j++)

{

Line line1 = geometry.lines[i];

Line line2 = geometry.lines[j];

if (line1.readyLine && line2.readyLine)

{

int state;

Point pnt = geometry.GetIntersectionPointOfTwoLines(line1, line2, out state);

if (state == 1)

{

geometry.tempdots.Add(new Dot(pnt.X, pnt.Y, size: 1, stable: false)); // Размер!!

}

}

}

}

geometry.draw();

geometry.DrawMiddles(programMode != Mode.normal);

blank.Image = data;

}

private void Form1\_Load(object sender, EventArgs e)

{

data = new Bitmap(blank.Width, blank.Height);

geometry = new Geometry(Graphics.FromImage(data));

}

// Двойной клик по форме

private void blank\_MouseDoubleClick(object sender, MouseEventArgs e)

{

switch (programMode)

{

//Две точки добавится одна удалиться и полигон построится

case Mode.drawPoly:

this.geometry.polygons.Last().dots.Remove(this.geometry.polygons.Last().dots.Last()); //hack for double dots

break;

case Mode.normal:

break;

}

this.programMode = Mode.normal;

this.blank.Cursor = System.Windows.Forms.Cursors.Arrow;

}

// Клик мышкой по белому полю

private void blank\_MouseClick(object sender, MouseEventArgs e)

{

Point position = e.Location;

foreach (Dot dot in geometry.dots(true)) // Проверяем каждую точку, нет ли возможности "примагнититься"

if (dot.is\_near(e.Location))

position = dot.Position;

switch (programMode) // Определяем текущий режим программы

{

case Mode.drawPoly:

geometry.polygons.Last().AddPoint(position);

break;

case Mode.drawLine:

geometry.lines.Last().AddPoint(position);

if (geometry.lines.Last().readyLine)

geometry.lines.Add(new Line());

break;

case Mode.normal:

foreach (Dot dot in geometry.dots(false))

if (dot.is\_near(e.Location)) // Если точка близко к курсору

{

dragged = dot; // То при клике она выбирается

programMode = Mode.dragging; // Меняем режим программы на "перемещение точки"

break;

}

if (Control.alt\_modifier)

{

foreach (Polygon poly in geometry.polygons) //проверяем аналогично полигоны

if (poly.current)

{

this.copied = poly.Copy();

this.copied.SetPosition(new Point(-60, 0));

geometry.polygons.Add(this.copied);

this.programMode = Mode.figureDrag;

break;

}

foreach (Line line in geometry.lines) //проверяем, нет ли под мышкой линий

if (line.current)

{

this.copied = line.Copy(); //если есть - копируем

this.copied.SetPosition(new Point(-60, 0));

geometry.lines.Add(this.copied);

this.programMode = Mode.figureDrag;

break;

}

foreach (Circle circle in geometry.circles) //проверяем, нет ли под мышкой circle

if (circle.current)

{

this.copied = circle.Copy(); //если есть - копируем

this.copied.SetPosition(new Point(-60, 0));

geometry.circles.Add(this.copied);

this.programMode = Mode.figureDrag;

break;

}

}

break;

case Mode.deleting:

foreach (Polygon poly in geometry.polygons)

if (poly.current)

{

geometry.RemovePoly(poly);

break;

}

foreach (Line line in geometry.lines)

if (line.current)

{

geometry.RemoveLine(line);

break;

}

foreach (Circle circle in geometry.circles)

if (circle.current)

{

geometry.RemoveCircle(circle);

break;

}

foreach(Dot dot in geometry.dots\_list)

{

if (dot.current)

{

geometry.dots\_list.Remove(dot);

break;

}

}

this.current\_obj\_info.Text = "";

break;

case Mode.figureDrag:

foreach (Line line in geometry.lines) // Проверяем линии

if (line.current)

{

this.draggedFigure = line;

break;

}

if (this.draggedFigure == null) // Если не нашлось линий под курсором - проверяем многоугольники

foreach (Polygon poly in geometry.polygons)

if (poly.current)

{

this.draggedFigure = poly;

break;

}

if (this.draggedFigure == null) // Если не нашлось полигона под курсором - проверяем круги

foreach (Circle circle in geometry.circles)

if (circle.current)

{

this.draggedFigure = circle;

break;

}

break;

case Mode.drawCircle:

this.geometry.circles.Last().center.Position = position;

this.geometry.circles.Last().centerDone = true;

break;

case Mode.addDot:

this.geometry.dots\_list.Add(new Dot(position.X, position.Y, 6, true));

this.programMode = Mode.addDot;

break;

}

checkZadacha(position, true);

update();

}

public void checkZadacha(Point pnt, bool click = false)

{

switch (zadacha)

{

case Zadacha.obolochka\_is\_tochek:

List<Point> points = new List<Point>();

foreach (Dot dot in geometry.dots\_list)

{

points.Add(dot.Position);

}

geometry.pnts = this.geometry.ConvexHull(points);

break;

case Zadacha.convex:

if (click)

foreach (Polygon poly in geometry.polygons)

if (poly.current)

{

MessageBox.Show((lang ? "Обраний многокутник " : "Выбранный многоугольник ") + (poly.PolygonIsConvex() ? (lang ? "опуклий." : "выпуклый.") : (lang ? "неопуклий." : "невыпуклый.")));

this.zadacha = Zadacha.none;

break;

}

break;

case Zadacha.inside:

if (click) {

bool o = true;

foreach (Polygon poly in geometry.polygons)

if (poly.PointIsInPolygon(pnt))

{

MessageBox.Show((lang ? "Точка знаходиться всередині " : "Точка находится внутри ") + (poly.PolygonIsConvex() ? (lang ? "опуклого" : "выпуклого") : (lang ? "неопуклого" : "невыпуклого")) + (lang ? " многокутника." : " многоугольника."));

this.zadacha = Zadacha.none;

o = false;

break;

}

if (o)

MessageBox.Show(lang ? "Точка знаходить зовні многокутника." : "Точка находится снаружи многоугольника.");

Dot t = new Dot(pnt.X, pnt.Y, stable:false);

geometry.dots\_list.Add(t);

this.zadacha = Zadacha.none;

}

break;

case Zadacha.minimalCircle:

List<MyPoint> p = new List<MyPoint>();

foreach (Dot dot in geometry.dots\_list)

{

p.Add(new MyPoint(dot.Position.X, dot.Position.Y));

}

int n = p.Count(); // Number of MyPoints input by the user

MyPoint[] b = new MyPoint[3]; // MyPoints on the boundary of the circle

MyCircle sec; // Smallest Enclosing Circle

sec = geometry.findSec(n, p.ToArray(), 0, b, click);

Circle c = new Circle();

c.center.Position = sec.getP1();

c.point2.Position = sec.getP2();

c.brush = new SolidBrush(Color.FromArgb(30, 0, 0, 255));

c.centerDone = true;

geometry.tempcircles.Clear();

geometry.tempcircles.Add(c);

break;

}

}

Point last\_location = new Point(0, 0);

// Движение мышкой по полю

private void Form1\_MouseMove(object sender, MouseEventArgs e)

{

// На какое растояние передвинулась точка

Point delta = new Point(last\_location.X - e.Location.X, last\_location.Y - e.Location.Y);

Point position = e.Location;

switch (programMode)

{

case Mode.figureDrag:

if (this.draggedFigure != null)

this.draggedFigure.SetPosition(delta);

goto case Mode.normal;

case Mode.normal:

case Mode.deleting:

foreach (Dot dot in geometry.dots(false)) {

dot.current = dot.is\_near(position);

}

foreach (Polygon poly in geometry.polygons)

{

poly.current = poly.PointIsInPolygon(e.Location);

if (poly.current)

this.current\_obj\_info.Text = poly.ObjInfo;

}

foreach (Line line in geometry.lines) {

line.current = line.PointIsInLine(e.Location);

if (line.current)

this.current\_obj\_info.Text = line.ObjInfo;

}

foreach (Circle cirlce in geometry.circles)

{

cirlce.current = cirlce.PointIsInCircle(e.Location);

if (cirlce.current)

this.current\_obj\_info.Text = cirlce.ObjInfo;

}

break;

case Mode.dragging:

if (dragged != null)

{

foreach (Dot dot in geometry.dots(true))

if (dot.is\_near(position) && dot != dragged)//если рядом с мішкой есть точка...

{

position = dot.Position; //"примагничиваемся" к точке

break; //выходим из цикла после нахождения точки

}

dragged.Position = position;

}

break;

case Mode.drawCircle:

this.geometry.circles.Last().point2.Position = position;

goto case Mode.normal;

}

checkZadacha(position);

update();

last\_location = e.Location;

}

private void blank\_MouseUp(object sender, MouseEventArgs e)

{

Point position = e.Location;

switch (programMode)

{

case Mode.dragging:

foreach (Dot dot in geometry.dots(true)) // Проверяем каждую точку, нет ли возможности "примагнититься"

if (dot.is\_near(e.Location) && dot != dragged) // Если есть точка рядом - меняем координаты

{

position = dot.Position;

break;

}

else

dragged.Position = position; // Иначе просто примагничиваем

dragged = null;

programMode = Mode.normal;

break;

case Mode.figureDrag:

this.draggedFigure = null;

break;

case Mode.drawCircle:

this.geometry.circles.Last().point2.Position = position;

geometry.circles.Add(new Circle());

break;

}

}

private void draw\_line\_button\_Click(object sender, EventArgs e)

{

this.current\_obj\_info.Text = "";

this.programMode = Mode.drawLine;

geometry.lines.Add(new Line());

}

private void add\_polygon\_Click(object sender, EventArgs e)

{

this.current\_obj\_info.Text = "";

this.programMode = Mode.drawPoly;

Polygon poly = new Polygon();

geometry.polygons.Add(poly);

update();

}

private void endFigure\_Click(object sender, EventArgs e)

{

this.programMode = Mode.normal;

}

private void deleteToolStripMenuItem\_Click(object sender, EventArgs e)

{

this.current\_obj\_info.Text = "";

this.programMode = Mode.deleting;

}

private void dragFigures\_Click(object sender, EventArgs e)

{

this.current\_obj\_info.Text = "";

this.programMode = Mode.figureDrag;

}

private void addcircle\_Click(object sender, EventArgs e)

{

this.current\_obj\_info.Text = "";

this.programMode = Mode.drawCircle;

this.geometry.circles.Add(new Circle());

}

private void addDot\_Click(object sender, EventArgs e)

{

this.current\_obj\_info.Text = "";

this.programMode = Mode.addDot;

}

private void zadacha4\_Click(object sender, EventArgs e)

{

count = 1;

geometry.pnts.Clear();

this.programMode = Mode.addDot;

this.zadacha = Zadacha.obolochka\_is\_tochek;

List<Point> points = new List<Point>();

foreach (Dot dot in geometry.dots\_list)

{

points.Add(dot.Position);

}

geometry.pnts = this.geometry.ConvexHull(points);

update();

this.lines.Enabled = false;

this.polylines.Enabled = false;

this.addcircle.Enabled = false;

this.addDot.Enabled = true;

}

private void button1\_Click(object sender, EventArgs e)

{

this.zadacha = Zadacha.none;

if (count == 1)

{

List<Point> points = new List<Point>();

foreach (Dot dot in geometry.dots\_list)

{

points.Add(dot.Position);

}

geometry.pnts = this.geometry.ConvexHull(points);

update();

}

this.lines.Enabled = true;

this.polylines.Enabled = true;

this.addcircle.Enabled = true;

this.addDot.Enabled = true;

}

// Oчистить всё

public void clearAll()

{

this.geometry.lines.Clear();

this.geometry.dots\_list.Clear();

this.geometry.circles.Clear();

this.geometry.polygons.Clear();

this.geometry.tempcircles.Clear();

this.geometry.tempdots.Clear();

this.geometry.pnts.Clear();

this.current\_obj\_info.Text = "";

update();

}

private void button2\_Click(object sender, EventArgs e)

{

clearAll();

this.programMode = Mode.normal;

}

private void button7\_Click(object sender, EventArgs e)

{

this.programMode = Mode.normal;

this.zadacha = Zadacha.convex;

}

private void button8\_Click(object sender, EventArgs e)

{

this.programMode = Mode.normal;

this.zadacha = Zadacha.inside;

}

private void button9\_Click(object sender, EventArgs e)

{

clearAll();

this.zadacha = Zadacha.minimalCircle;

this.lines.Enabled = false;

this.polylines.Enabled = false;

this.addcircle.Enabled = false;

this.addDot.Enabled = true;

}

private void button9\_Click\_1(object sender, EventArgs e)

{

count = 0;

clearAll();

this.programMode = Mode.addDot;

this.zadacha = Zadacha.minimalCircle;

this.lines.Enabled = false;

this.polylines.Enabled = false;

this.addcircle.Enabled = false;

this.addDot.Enabled = true;

}

// Алгоритм оболочки из интернетов

//http://stackoverflow.com/questions/10020949/gift-wrapping-algorithm

public List<Point> ConvexHull(List<Point> points)

{

if (points.Count < 3)

{

return new List<Point>();

}

List<Point> hull = new List<Point>();

// get leftmost point

Point vPointOnHull = points.Where(p => p.X == points.Min(min => min.X)).First();

Point vEndpoint;

do

{

hull.Add(vPointOnHull);

vEndpoint = points[0];

geometry.gr.DrawLine(Pens.Red, vPointOnHull, vEndpoint);

for (int i = 1; i < points.Count; i++)

{

if ((vPointOnHull == vEndpoint) || (Geometry.Orientation(vPointOnHull, vEndpoint, points[i]) == -1))

{

vEndpoint = points[i];

}

if (hull.IndexOf(points[i]) == -1)

{

geometry.gr.DrawLine(Pens.Red, vPointOnHull, points[i]);

if (hull.Count >= 2)

geometry.gr.DrawLines(Pens.Black, hull.ToArray());

this.blank.Refresh();

Thread.Sleep(200);

}

}

geometry.draw();

if (hull.Count >= 3)

{

geometry.gr.DrawLines(Pens.Black, hull.ToArray());

geometry.gr.DrawLine(Pens.Black, hull[0], hull.Last());

this.blank.Refresh();

Thread.Sleep(200);

}

geometry.draw();

vPointOnHull = vEndpoint;

}

while (vEndpoint != hull[0]);

return hull;

}

private void button6\_Click(object sender, EventArgs e)

{

count = 1;

button6.Enabled = false;

geometry.pnts.Clear();

update();

this.programMode = Mode.normal;

this.zadacha = Zadacha.obolochka\_is\_tochek;

List<Point> points = new List<Point>();

foreach (Dot dot in geometry.dots\_list)

{

points.Add(dot.Position);

}

ConvexHull(points);

geometry.pnts = this.geometry.ConvexHull(points);

update();

this.lines.Enabled = false;

this.polylines.Enabled = false;

this.addcircle.Enabled = false;

this.addDot.Enabled = true;

button6.Enabled = true;

}

private void checkBox1\_CheckedChanged(object sender, EventArgs e)

{

if (checkBox1.Checked)

groupBox2.Visible = true;

else

groupBox2.Visible = false;

}

private void Form1\_KeyDown(object sender, KeyEventArgs e)

{

Control.alt\_modifier = e.Alt;

}

private void Form1\_KeyUp(object sender, KeyEventArgs e)

{

Control.alt\_modifier = e.Alt;

}

public static bool lang = true;

private void comboBox1\_SelectedIndexChanged(object sender, EventArgs e)

{

switch (comboBox1.Text)

{

case "Русский":

lang = false;

button2.Text = "Удалить все";

groupBox1.Text = "Задачи";

button1.Text = "Отключить задачу";

zadacha4.Text = "Выпуклая оболочка";

button7.Text = "Проверка выпуклости";

button8.Text = "Точка внутри многоугольника";

button9.Text = "Минимальная окружность около точек";

checkBox1.Text = "Показать длины в пикселях";

groupBox2.Text = "Текущий объект";

button6.Text = "Анимация выпуклой оболочки";

toolTip1.Active = false;

toolTip2.Active = true;

break;

case "Українська":

lang = true;

button2.Text = "Видалити все";

groupBox1.Text = "Задачі";

button1.Text = "Відключити задачу";

zadacha4.Text = "Опукла оболочка";

button7.Text = "Перевірка опуклості";

button8.Text = "Точка всередині многокутника";

button9.Text = "Мінімальне коло навколо точок";

checkBox1.Text = "Показати довжини сторін в пікселях";

groupBox2.Text = "Поточний об'єкт";

button6.Text = "Анімація опуклої оболонки";

toolTip2.Active = false;

toolTip1.Active = true;

break;

}

}

}

}

### Клас “Program”

using System;

using System.Collections.Generic;

using System.Linq;

using System.Windows.Forms;

namespace CGeometry

{

static class Program

{

// Главная точка входа для приложения.

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new Form1());

}

}

}

## Блок-схема

## Практичне застосування програми

Дану програму можна використовувати:

* на уроках математики, для освоєння матеріалу;
* на уроках програмування;
* при самостійному освоєнні матеріалу;
* при підготовці до олімпіад з інформатики високого рівня;
* як посібник.

## Програмно-апаратні вимоги

Для повноцінної роботи програми необхідно задовольнити наступні вимоги:

Системні вимоги:

* NET Framework 4.5 або пізніше;
* Windows 8.1 або пізніші версії.

Апаратні вимоги:

* IntelCeleronDual-Coreprocessor або вище;
* Колонки (не обов’язково);
* монітор з роздільною здатністю 1366 х 768;
* 20 MB вільного місця на жорсткому диску.

## Комплектація програми

Щоб відкриту програму на своєму комп’ютері необхідно скачати собі папку «CGeometry» та запустити файл «CGeometry.exe»

1. CGeometry.exe – файл запуску програми.
2. Файли \*.jpg, \*.png, \*.gif, – це картинки, що використовуються програмою.
3. Файли \*.htm – це файли, що містять текстові дані для програми.
4. Документація.doc – файл з документацією до роботи;
5. ReadMe.txt – файл, необхідний для ознайомлення з програмою.

## Використані програмні засоби

* Visual Studio 2013-2015;
* Microsoft Word 2013;
* Gimp 2;
* Adobe Photoshop CS4
* Inkscape
* Paint.Net
* Sublime Text 3;
* Koala;
* Mozilla Firefox.

# ВИСНОВКИ

Задля виконання цієї роботи було досліджено відбудова 2-D графіки у мові програмування С#. Також, було змодельовано деякі задачі обчислювальної геометрії, було розглянуто їх особливості.

За час роботи над програмою я відкрив для себе багато нових аспектів програмування, а також вдосконалив раніше отримані знання як з програмування.

Також в процесі написання роботи я у великій мірі зрозумів, що основами сучасних прикладних наук є все ж таки фундаментальні науки, зокрема саме математичні знання є основою для програмістів. Створена мною програма дозволяє користувачу краще ознайомитися з темою «Обчислювальна геометрія на площині». Я сподіваюсь, що моя робота в цьому плані може бути корисною.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

## Література

1. Кормен Т., Лейзерсон Ч., Ривести Р. «Алгоритмы. Построение и анализ». М.: МЦНМО, 2000.
2. Шикин Е.В., Боресков А.В., Зайцев А.А. «Начало компьютерной графики». М.: Диалог-МИФИ, 1993.
3. Курант Р., Роббинс Г. «Что такое математика». М.: ОГИЗ, государственное изд-во технико-теоретической литературы, 1947.
4. Окулов С.М. «100 задач по информатике». Киров: Изд-во ВГПУ, 2000.
5. Станкевич А.С. «Решение задач I Всероссийской командной олимпиады по программированию». Информатика № 12/2001.
6. Андреева Е.В. «Геометрические задачи на олимпиадах по информатике». Информатика № 14/2002.

## Інтернет-джерела

1. <http://msdn.microsoft.com/ru-ru/library>
2. <https://ru.wikipedia.org>
3. <http://habrahabr.ru/>
4. <http://www.cyberforum.ru/>
5. <http://stackoverflow.com/>
6. <http://getbootstrap.com/>
7. <http://daneden.github.io/animate.css/>
8. http://fortawesome.github.io/Font-Awesome/icons/