ПРАКТИКУМ
Программирование символьных вычислений
«ГЕОМЕТРИЯ»

Мамиева Мария Алановна
324 группа

Москва
2025

Содержание:

0. Цель работы

Изучить базовые функциональные средства языка программирования Лисп, ориентированного на символьные вычисления и задачи искусственного интеллекта. Освоить основные приемы обработки сложных структур символьных данных, разработать и реализовать лисп-программу для определенного варианта преобразования символьных выражений.

1.Описание варианта

1.1 Изначальный вариант

Геометрия
Точки на плоскости заданы с помощью Лисповского списка: ((x1 y1) (x2 y2)…).
b. По двум наборам из трех точек построить две параболы
i. определить, пересекают ли параболы ось X, если да, то найти эти точки;
ii. определить, пересекаются ли параболы, если да, найти эти точки пересечения;
iii. построить функцию, которая в любой точке X принимает значение, равное среднему арифметическому значений заданных парабол в этой точке.
c. По заданному набору точек на плоскости построить все возможные треугольники
i. найти все равнобедренные и равносторонние треугольники;
ii. найти все прямоугольные треугольники;
iii. найти все подобные треугольники.

1.2 Добавления

В пункт С добавили поиск равных треугольников.

2. Описание кода для пункта B

Основные задачи и их реализация :

2.1 Построение парабол

Входные данные - список из трех точек, вида ((x1 y1) (x2 y2) (x3 y3)).
Вывод: Список из коэффициентов a, b, c, вида (a b c) или текст (если параболу невозможно построить).
Цель : построить парабол.
Логика:
Программа использует метод определителей (Крамера) для нахождения коэффициентов параболы вида:
$y = ax^2 + bx + c$. По трем точкам строится система уравнений:
$y_1 = a \cdot x_1^2 + b \cdot x_1 + c$
$y_2 = a \cdot x_2^2 + b \cdot x_2 + c$

$y_3 = a \cdot x_3^2 + b \cdot x_3 + c$
Коэффициенты вычисляются по формулам:
· a = det1 / full_det
· b = det2 / full_det
· c = det3 / full_det
Проверки корректности:
1. full_det ≠ 0 - точки не лежат на одной вертикали
2. det1 ≠ 0 - дополнительные проверки вырожденности
Если условия не выполняются, возвращается "Невозможно построить параболу!"

Используемые функции :

1. *build_parabola(set) - главная функция построения параболы*
2. *full_det(x1 x2 x3) - главный определитель*
3. *det1(p1 p2 p3) - определитель для коэффициента a*
4. *det2(p1 p2 p3) - определитель для коэффициента b*
5. *det3(p1 p2 p3) - определитель для коэффициента c*

Описание функций:
build_parabola(set) - подаем переменную set - лисповский список для одной параболы.
Проверяет возможность построения параболы:

- Если (full_det x1 x2 x3) = 0 → "Невозможно построить параболу!»
- Если (det1 p1 p2 p3)/(full_det x1 x2 x3) = 0 → "Невозможно построить параболу!»

Вычисляет коэффициенты парабол :

- a = (det1 p1 p2 p3) / (full_det x1 x2 x3)
- b = (det2 p1 p2 p3) / (full_det x1 x2 x3)
- c = (det3 p1 p2 p3) / (full_det x1 x2 x3)

full_det(x1 x2 x3) - подаем координаты x. Ищем определитель по формуле :
full_det = $(x_1 - x_2) \times (x_2 - x_3) \times (x_3 - x_1)$

det1(p1 p2 p3) - подаем три точки. Вычисляет определитель для коэффициента a по формуле :
det1 = $x_1(y_3 - y_2) - x_2(y_1 - y_3) - x_3(y_2 - y_1)$

det2(p1 p2 p3) - подаем три точки. Вычисляет определитель для коэффициента b по формуле :
det2 = $y_1(x_3^2 - x_2^2) - y_2(x_1^2 - x_3^2) - y_3(x_2^2 - x_1^2)$

det3(p1 p2 p3) - подаем три точки. Вычисляет определитель для коэффициента c по формуле :
det3 = $y_1(x_2^2 x_3 - x_3^2 x_2) - y_2(x_3^2 x_1 - x_1^2 x_3) - y_3(x_1^2 x_2 - x_2^2 x_1)$

2.2 Пересечение параболы с осью X

Входные данные: Список коэффициентов параболы вида (a b c)
Вывод:

- Если пересечений нет: "Нет пересечений параболы с осью X!"

• Если одно пересечение: (x 0) - точка пересечения
• Если два пересечения: ((x1 0) (x2 0)) - список из точек пересечения

Цель: Определить, пересекает ли парабола ось X, и найти точки пересечения.

Логика:

Парабола пересекает ось X в точках, где y = 0, т.е. решается квадратное уравнение:
$ax^2 + bx + c = 0$
Алгоритм решения:
1. Вычисляем дискриминант: $D = b^2 - 4ac$
2. Анализируем дискриминант:
   · Если D < 0: нет действительных корней - нет пересечений
   · Если D = 0: один корень - парабола касается оси X
   · Если D > 0: два корня - парабола пересекает ось X в двух точках

Формулы для вычисления корней:
$x_1 = (-b + \sqrt{D}) / (2a)$
$x_2 = (-b - \sqrt{D}) / (2a)$

Используемые функции и их описания:

x_intersect(coef) -Находит точки пересечения с помощью вычисления дискриминанта по формуле:
D = (- (* b b) (* 4 a c)) ; $D = b^2 - 4ac$
Проверка условий:
• Если D < 0: "Нет пересечений параболы с осью X!"
• Если D = 0: (один корень) x = -b/(2a), y = 0
• Если D > 0: (два корня): $x_1 = (-b + \sqrt{D})/(2a)$, $x_2 = (-b - \sqrt{D})/(2a)$

2.3 Определение пересечения двух парабол

Входные данные: Коэффициенты двух парабол вида (a1 b1 c1) и (a2 b2 c2)

Вывод:

• Совпадают: "Параболы совпадают!"
• Не пересекаются: "Параболы не пересекаются!"
• Одно пересечение: (x y)
• Два пересечения: ((x1 y1) (x2 y2))

Цель: Найти точки пересечения двух парабол.

Логика:

Точки пересечения находятся решением системы уравнений двух парабол. Приравниваем уравнения и решаем полученное квадратное уравнение относительно x, затем находим соответствующие значения y.
Алгоритм решения:
1. Вычисляем коэффициенты уравнения:
• $A = a_1 - a_2$
• $B = b_1 - b_2$
• $C = c_1 - c_2$
2. Анализируем полученное уравнение $Ax^2 + Bx + C = 0$
3. В зависимости от коэффициентов определяем количество точек пересечения

Используемые функции:

parabolas_intersect(coef1 coef2) -поиск пересечений
Анализ особых случаев:
- A=0, B=0, C=0: "Параболы совпадают!»
- A=0, B=0:"Параболы не пересекаются!"
- A≠0: квадратное уравнение, два решения, вызываем функцию discrim_check coef

discrim_check(coef A B C) - проверка дискриминанта
Вычисление дискриминанта: $D = b^2 - 4ac$
Анализ дискриминанта:
- D < 0: нет действительных корней
- D = 0: один корень (касание)
- D > 0: два корня (пересечение)

Вычисление координат точек пересечения

calculate_y(coef x) - вычисление y-координаты
Вычисляет значение y для заданного x по формуле параболы: $y = ax^2 + bx + c$

2.4 Построение функции среднего арифметического парабол

Входные данные: Коэффициенты двух парабол вида (a1 b1 c1) и (a2 b2 c2)
Вывод: Уравнение функции в виде списка ("Y=" a_avg "x^2+" b_avg "x+" c_avg) с упрощением при нулевых коэффициентах
Цель: Построить функцию, которая в любой точке X принимает значение, равное среднему арифметическому значений заданных парабол в этой точке.
Логика:
Среднее арифметическое двух парабол $y_1(x) = a_1x^2 + b_1x + c_1$ и $y_2(x) = a_2x^2 + b_2x + c_2$ вычисляется по формуле:
$Y\_avg(x) = (y_1(x) + y_2(x)) / 2 = ((a_1 + a_2)/2)x^2 + ((b_1 + b_2)/2)x + ((c_1 + c_2)/2)$
Алгоритм:
1. Вычисляем средние коэффициенты:
   · $a\_avg = (a_1 + a_2) / 2$
   · $b\_avg = (b_1 + b_2) / 2$
   · $c\_avg = (c_1 + c_2) / 2$
2. Форматируем уравнение с учетом нулевых коэффициентов

Используемая функция:

arith_mean(coef1 coef2) - построение средней функции

Проверка особых случаев упрощения:
- Если a_avg = 0 и b_avg = 0: функция постоянна "Y=" c_avg
- Если a_avg = 0: функция линейна "Y=" b_avg "x+" c_avg
- Общий случай: полное квадратное уравнение "Y=" a_avg "x^2+" b_avg "x+" c_avg)

2.5 Координация работы пункта B

2.5.1 Главная функция координации

parB(l1 l2) - главная функция пункта B
Входные данные: Два списка точек для двух парабол вида ((x1 y1) (x2 y2) (x3 y3))
Вывод: Полный анализ обеих парабол с промежуточными результатами
Логика: Координирует весь процесс анализа двух парабол, последовательно вызывая функции построения и анализа.
Пошаговое описание:

I. Вывод заголовка "Пункт B"
II. Построение и вывод первой параболы («Парабола 1», (build_parabola l1))
III. Построение и вывод второй параболы («Парабола 2», (build_parabola l2))
IV. Передача результатов построения функции анализа(bdop (build_parabola l1) (build_parabola l2))

2.5.2 Функция анализа результатов

bdop(coef1 coef2) - обработчик результатов построения парабол
Входные данные: Коэффициенты двух построенных парабол
Вывод: Полный анализ пересечений и построение средней функции
Логика: Анализирует результаты построения парабол и в зависимости от их корректности выполняет соответствующий анализ.
Пошаговое описание:
Обе параболы построены успешно:
Анализ пересечений с осью X для обеих парабол
Анализ пересечений между параболами
Построение средней функции
Только первая парабола построена успешно: анализ только первой параболы (пересечения с осью X)
Только вторая парабола построена успешно: анализ только второй параболы (пересечения с осью X)

2.6 Исправления

Был практически полностью переписан пункт 2.3 - поиск точкек пересечения парабол.
Ошибка в исходном коде: расчет точек при a1=a2 выдавал ошибочные данные.
Именения:
parabolas_intersect(coef1 coef2) - разбивает случаи с пересекающимися параболами на два : с равными коэффициентами a и с различными . (Для избежания ошибки при делении на ноль)

solve_same_a_case - новая функция
Вход: коэффициенты парабол (два списков)
Выход: точка пересечения

solve_different_a_case - новая функция
Вход: коэффициенты парабол (два списков)
Выход: точки пересечения
Вызывает функцию process_discriminant_result

process_discriminant_result(a b c coef1) - аналог **discrim_check(coef A B C),** но для подсчета точек пересечения вызывает отдельные функции

single_intersection_point(a b coef1) - считает единственную точку пересечения (просто по формуле)

first_intersection_point (a b discriminant coef1) - считает первую точку пересечения из двух - по формуле

second_intersection_point (a b discriminant coef1) - считает вторуюточку пересечения из двух - по формуле

- Функции single_intersection_point first_intersection_point, second_intersection_point были добавлены для удобочитаемости кода

3. Описание кода для пункта C

3.1 Построение всех возможных треугольников

Входные данные: Список точек на плоскости вида ((x1 y1) (x2 y2) (x3 y3) ...)
Вывод: Список всех возможных невырожденных треугольников вида (((x1 y1) (x2 y2) (x3 y3)) …)
Цель: Построить все возможные треугольники из заданного набора точек, исключая вырожденные случаи.
Логика: Программа генерирует все возможные комбинации по 3 точки из заданного набора, затем фильтрует их, оставляя только невырожденные треугольники (с ненулевой площадью).
Алгоритм:
- Удаление дублирующихся точек
- Генерация всех комбинаций по 3 точки
- Проверка каждой комбинации на невырожденность (площадь ≠ 0)
- Возврат списка валидных треугольников

Используемые функции:
build_triangle(l) - главная функция построения треугольников
Вход: список точек
Вызов: удаление дубликатов, генерация комбинаций, проверка треугольников
Вывод: список всех возможных треугольников

rem_dup(l) - удаление дублирующихся точек.
Рекурсивно удаляет повторяющиеся точки из списка.
Использует: find_dup для проверки наличия дубликатов

find_dup(e l) - поиск дубликатов в списке.
Проверяет наличие элемента в списке.
Возвращает: T если найден, nil если нет (аналог member)

build_triangles(l) - построение всех комбинаций треугольников.
Обрабатывает список комбинаций точек.

Для каждой комбинации вызывает проверку валидности.
Объединяет результаты рекурсивно

check_triangles(tri) - проверка треугольника.
Проверяет что треугольник не вырожденный.
Использует: area для вычисления площади. Возвращает: треугольник если площадь ≠ 0, иначе nil.

area(tri) - вычисление площади треугольника.
Использует формулу площади через координаты вершин:
area = $|(x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2)) / 2|$
Если площадь = 0 - треугольник вырожден

free_points(l) - генерация всех комбинаций точек.
Генерирует все комбинации по 3 точки рекурсивно.
Использует: free_dop для фиксации первой точки

free_dop(a l) - вспомогательная для комбинаций.
Фиксирует первую точку a.
Генерирует комбинации с оставшимися точками

free_ddop(a b l) - вспомогательная для комбинаций.
Фиксирует первые две точки a и b.
Добавляет третью точку из оставшегося списка.
Создает треугольник (a b c) для каждой точки c

3.2 Поиск равнобедренных и равносторонних треугольников

Входные данные: Список треугольников вида (((x1 y1) (x2 y2) (x3 y3)) ...)
Вывод:

- Равнобедренные треугольники: список треугольников с двумя равными сторонами
- Равносторонние треугольники: список треугольников со всеми равными сторонами

Цель: Найти все равнобедренные и равносторонние треугольники среди построенных.
Логика: Программа проверяет каждый треугольник на равенство длин сторон с использованием приблизительного сравнения для учета погрешностей вычислений.
Используемые функции:
distance(p1 p2) - вычисление расстояния между точками. Использует формулу расстояния:
$\sqrt{((x_2 - x_1)^2 + (y_2 - y_1)^2)}$
Вычисляет длину стороны треугольника

approx_equal(a b) - проверка приблизительного равенства.
Проверяет равенство чисел с погрешностью 0.001.
Учитывает погрешности вычислений с плавающей точкой.

equilateral(l) - поиск равносторонних треугольников.
Фильтрует список , оставляя только равносторонние.
Использует: equilateral1 для проверки каждого треугольника

equilateral1(l) - проверка равносторонности.
Проверяет что все три стороны равны.
Возвращает: T если треугольник равносторонний

isosceles(l) - поиск равнобедренных треугольников.
Фильтрует список треугольников, оставляя только равнобедренные.
Использует: isosceles1 для проверки каждого треугольника

isosceles1(l) - проверка равнобедренности.
Проверяет что хотя бы две стороны равны.
Возвращает: T если треугольник равнобедренный

3.3 Поиск прямоугольных треугольников

Входные данные: Список треугольников вида (((x1 y1) (x2 y2) (x3 y3)) ...)
Вывод: Список прямоугольных треугольников (треугольников с прямым углом)
Цель: Найти все прямоугольные треугольники среди построенных.
Логика: Программа проверяет каждый треугольник на выполнение теоремы Пифагора - что квадрат одной стороны равен сумме квадратов двух других сторон.
Пошаговый алгоритм проверки:

I. Вычисление длин сторон треугольника
II. Проверка трех вариантов теоремы Пифагора
III. Если выполняется хотя бы одно условие - треугольник прямоугольный

Используемые функции:

right_triangles(l) - поиск прямоугольных треугольников
Фильтрует список треугольников
Оставляет только прямоугольные
Использует: right_triangle_p для проверки каждого треугольника

right_triangle_p(tri) - проверка прямоугольности треугольника
Вычисляет длины всех трех сторон
Передает стороны функции right_triangle_p1 для проверки теоремы Пифагора

right_triangle_p1(a b c) - проверка теоремы Пифагора
Проверяет все три возможные гипотенузы:
- Если $a^2 + b^2 \approx c^2$ (c - гипотенуза)
- Если $a^2 + c^2 \approx b^2$ (b - гипотенуза)
- Если $b^2 + c^2 \approx a^2$ (a - гипотенуза)

Использует: approx_equal для учета погрешностей вычислений

3.4 Поиск подобных треугольников

Входные данные: Список треугольников вида (((x1 y1) (x2 y2) (x3 y3)) ...)
Вывод: Список пар подобных треугольников вида ((t1 t2) (t3 t4) ...)

Цель: Найти все пары подобных треугольников среди построенных.
Логика: Программа проверяет все возможные пары треугольников на подобие, сравнивая отношения длин их сторон после сортировки.
Пошаговый алгоритм проверки подобия:

I. Для каждого треугольника t1:
- Вычислить длины всех трех сторон
- Отсортировать стороны по возрастанию: (s1 s2 s3)

II. Для каждого треугольника t2 (из оставшихся):Вычислить и отсортировать его стороны: (s1' s2' s3')
III. Проверить пропорциональность
IV. Если пропорциональность выполняется - треугольники подобны

Используемые функции:

similar_triangles(l) - поиск всех подобных треугольников
Рекурсивно обрабатывает список треугольников
Для каждого треугольника ищет подобные среди оставшихся
Объединяет результаты с рекурсивным вызовом для хвоста списка

find_similar(t1 l) - поиск подобных для одного треугольника
Ищет все треугольники в списке l, подобные треугольнику t1
Возвращает список пар (t1 t2) для каждой найденной пары

similar_p(t1 t2) - проверка подобия двух треугольников
Вычисляет и сортирует стороны обоих треугольников
Сравнивает отношения сторон через функцию similar_check

similar_check(s1 s2) - проверка пропорциональности сторон
Проверяет что все отношения сторон равны:
Защита от деления на ноль: проверяет что $s2_1 > 0$

sort3(a b c) - сортировка трех чисел
Сортирует три числа по возрастанию
Обеспечивает корректное сравнение сторон треугольников

3.5 Координация работы пункта C

Главная функция координации - triangleC(l)
Входные данные: Список точек на плоскости вида ((x1 y1) (x2 y2) (x3 y3) ...)
Вывод: Полный анализ всех треугольников с классификацией по типам
Цель: Координирует весь процесс построения и анализа треугольников, последовательно вызывая функции построения и классификации.
Логика: Функция последовательно строит все треугольники, затем находит и выводит все типы треугольников: обычные, равнобедренные, равносторонние, прямоугольные и подобные.
Пошаговое описание:

I. Вывод заголовка пункта
II. Построение и вывод всех треугольников

III. Поиск и вывод равнобедренных треугольников
IV. Поиск и вывод равносторонних треугольников
V. Поиск и вывод прямоугольных треугольников
VI. Поиск и вывод подобных треугольников

3.6 Дополнение к пункту - поиск равных треугольников

Цель: поиск равных треугольников
Вход: список подобных треугольников
Логика: Среди списка подобных ищем треугольники с равными сторонами - достаточно найти одну пару равных сторон, перед этим отсортировав стороны треугольников по возрастание
Используемые функции:
equal_triangles (similar_pairs) -фильтрует список пар подобных треугольников, оставляя только те пары, которые являются равными треугольниками.
Вход: список пар треугольников
Вывод: Список пар равных треугольников
Логика работы:
I. Базовый случай - пустой список возвращает nil
II. Проверка равенства - если текущая пара треугольников равны (через equal_side_p), добавляет её в результат
III. Рекурсия - обрабатывает оставшуюся часть списка

equal_side_p (t1 t2) - проверяет, являются ли два треугольника равными по длине наибольшей стороны.
Вход: два треугольника (списки из трёх точек)
Выход:
• T - если треугольники равны
• nil - в противном случае
Логика работы:
I. Вычисление сторон - находит длины всех сторон каждого треугольника
II. Сортировка - упорядочивает стороны по убыванию (sort3)
III. Сравнение - проверяет приблизительное равенство наибольших сторон

4.Тесты для пункта B

```
;; Тест 1: Нормальное построение двух парабол - проверка основного функционала
(print "=== Тест 1: Нормальное построение двух парабол ===")
(print "Цель: Проверить корректное построение парабол и их анализ")
(parB '((0 0) (1 1) (2 4)) '((0 1) (1 0) (2 1)))

;; Тест 2: Параболы не пересекаются - проверка обработки непересекающихся кривых
(print "=== Тест 2: Параболы не пересекаются ===")
(print "Цель: Проверить обработку случая непересекающихся парабол")
(parB '((0 2) (1 3) (2 6)) '((0 -1) (1 -2) (2 -5)))

;; Тест 3: Параболы касаются - проверка граничного случая касания
(print "=== Тест 3: Параболы касаются ===")
```

```
(print "Цель: Проверить случай близких, но не пересекающихся парабол")
(parB '((0 0) (1 1) (2 4)) '((0 1) (1 2) (2 5)))

;; Тест 4: Вырожденный случай - точки на одной прямой - проверка обработки ошибок
(print "=== Тест 4: Вырожденный случай - точки на одной прямой ===")
(print "Цель: Проверить обработку невозможности построения параболы")
(parB '((0 0) (1 1) (2 2)) '((0 1) (1 2) (2 3)))

;; Тест 5: Параболы совпадают - проверка идентичных кривых
(print "=== Тест 5: Параболы совпадают ===")
(print "Цель: Проверить обработку идентичных парабол")
(parB '((0 0) (1 1) (2 4)) '((0 0) (1 1) (2 4)))

;; Тест 6: Одна точка пересечения - проверка вырожденного пересечения
(print "=== Тест 6: Одна точка пересечения ===")
(print "Цель: Проверить нахождение одной точки пересечения")
(parB '((0 0) (1 1) (2 4)) '((0 0) (1 0) (2 0)))

;; Тест 7: Вертикальные "параболы" - проверка особых случаев построения
(print "=== Тест 7: Вертикальные параболы ===")
(print "Цель: Проверить обработку вертикальных линий")
(parB '((0 0) (0 1) (0 4)) '((1 0) (1 1) (1 4)))

;; Тест 8: Комплексные корни - проверка парабол без действительных корней
(print "=== Тест 8: Комплексные корни ===")
(print "Цель: Проверить обработку парабол без действительных пересечений с осью X")
(parB '((0 2) (1 3) (2 6)) '((0 1) (1 2) (2 5)))

;; Тест 9: Большие числа - проверка устойчивости к большим значениям
(print "=== Тест 9: Большие числа ===")
(print "Цель: Проверить устойчивость к большим значениям")
(parB '((100 10000) (200 40000) (300 90000)) '((100 5000) (200 10000) (300 15000)))

;; Тест 10: Отрицательные координаты - проверка работы с отрицательными значениями
(print "=== Тест 10: Отрицательные координаты ===")
(print "Цель: Проверить работу с отрицательными координатами")
(parB '((-2 4) (-1 1) (0 0)) '((-2 1) (-1 0) (0 1)))

;; Тест 11: Только одна парабола строится - проверка частичного успеха
(print "=== Тест 11: Только одна парабола строится ===")
(print "Цель: Проверить обработку частично успешного построения")
(parB '((0 0) (1 1) (2 2)) '((0 0) (1 1) (2 4)))

;; Тест 12: Горизонтальные линии - проверка вырожденных парабол
(print "=== Тест 12: Горизонтальные линии ===")
(print "Цель: Проверить построение горизонтальных линий")
(parB '((0 0) (1 0) (2 0)) '((0 1) (1 1) (2 1)))
```

```
;; Тест 13: Симметричные параболы - проверка симметричных случаев
(print "=== Тест 13: Симметричные параболы ===")
(print "Цель: Проверить обработку симметричных парабол")
(parB '((-1 1) (0 0) (1 1)) '((-1 2) (0 1) (1 2)))

;; Тест 14: Параболы с разными направлениями - проверка разнонаправленных кривых
(print "=== Тест 14: Параболы с разными направлениями ===")
(print "Цель: Проверить параболы с разной выпуклостью")
(parB '((0 0) (1 1) (2 4)) '((0 4) (1 1) (2 0)))
```

5.Тесты для пункта C

```
;; Тест 1: Нормальные треугольники - проверка основного функционала
(print "=== Тест 1: Нормальные треугольники ===")
(print "Цель: Проверить корректное построение треугольников и их анализ")
(triangleC '((0 0) (3 0) (0 4) (1 1) (2 2)))

;; Тест 2: Равносторонний треугольник
(print "=== Тест 2: Равносторонний треугольник ===")
(print "Цель: Проверить обнаружение равносторонних треугольников")
(triangleC '((0 0) (2 0) (1 1.732) (1 0) (0 1)))

;; Тест 3: Равнобедренные треугольники
(print "=== Тест 3: Равнобедренные треугольники ===")
(print "Цель: Проверить обнаружение равнобедренных треугольников")
(triangleC '((0 0) (4 0) (2 3) (1 0) (0 2)))

;; Тест 4: Прямоугольные треугольники
(print "=== Тест 4: Прямоугольные треугольники ===")
(print "Цель: Проверить обнаружение прямоугольных треугольников")
(triangleC '((0 0) (3 0) (0 4) (1 0) (0 1)))

;; Тест 5: Подобные треугольники
(print "=== Тест 5: Подобные треугольники ===")
(print "Цель: Проверить обнаружение подобных треугольников")
(triangleC '((0 0) (3 0) (0 4)
             (0 0) (6 0) (0 8)
             (1 1) (4 1) (1 5)))

;; Тест 6: Точки на одной прямой
(print "=== Тест 6: Точки на одной прямой ===")
(print "Цель: Проверить обработку вырожденных случаев")
(triangleC '((0 0) (1 1) (2 2) (3 3) (4 4)))

;; Тест 7: Все точки совпадают
(print "=== Тест 7: Все точки совпадают ===")
```

```
(print "Цель: Проверить обработку идентичных точек")
(triangleC '((0 0) (0 0) (0 0) (0 0) (0 0)))

;; Тест 8: Только 3 точки
(print "=== Тест 8: Только 3 точки ===")
(print "Цель: Проверить работу с минимальным количеством точек")
(triangleC '((0 0) (3 0) (0 4)))

;; Тест 9: Большие координаты
(print "=== Тест 9: Большие координаты ===")
(print "Цель: Проверить устойчивость к большим значениям")
(triangleC '((1000 1000) (3000 1000) (1000 4000) (2000 2000) (1500 1500)))

;; Тест 10: Отрицательные координаты
(print "=== Тест 10: Отрицательные координаты ===")
(print "Цель: Проверить работу с отрицательными координатами")
(triangleC '((-2 -2) (1 -2) (-2 1) (0 0) (-1 -1)))

;; Тест 11: Разные типы треугольников в одном наборе
(print "=== Тест 11: Разные типы треугольников в одном наборе ===")
(print "Цель: Проверить классификацию смешанного набора треугольников")
(triangleC '((0 0) (3 0) (0 4)          ; прямоугольный
           (0 0) (2 0) (1 1.732)       ; равносторонний
           (0 0) (4 0) (2 3)           ; равнобедренный
           (1 1) (2 2) (3 3)))         ; вырожденный

;; Тест 12: Точки образуют только вырожденные треугольники
(print "=== Тест 12: Точки образуют только вырожденные треугольники ===")
(print "Цель: Проверить обработку когда все треугольники вырожденные")
(triangleC '((0 0) (1 1) (2 2) (3 3) (0 1) (1 2)))

;; Тест 13: Точки в вершинах квадрата
(print "=== Тест 13: Точки в вершинах квадрата ===")
(print "Цель: Проверить построение треугольников из точек квадрата")
(triangleC '((0 0) (2 0) (2 2) (0 2) (1 1)))

;; Тест 14: Много точек с дубликатами
(print "=== Тест 14: Много точек с дубликатами ===")
(print "Цель: Проверить обработку дублирующихся точек")
(triangleC '((0 0) (3 0) (0 4) (0 0) (3 0) (0 4) (1 1) (1 1)))

;; Тест 15: Комплексный тест со всеми типами
(print "=== Тест 15: Комплексный тест со всеми типами ===")
(print "Цель: Комплексная проверка всех функций анализа треугольников")
(triangleC '((0 0) (4 0) (0 3)          ; прямоугольный
           (5 0) (7 0) (6 1.732)       ; равносторонний
           (8 0) (12 0) (10 4)         ; равнобедренный
           (0 5) (3 5) (0 9)           ; прямоугольный (подобный первому)
```

(1 1) (2 2) (3 3))) ; вырожденный