Практическая работа 3. Фигуры и точки

В работе предполагается введение пользовательских типов данных в виде структур или классов для работы с точками и фигурами в двумерном пространстве.

В каждой работе необходимо ввести тип Point для работы с точками. Точка задается двумя координатами типа double (структура\класс Point должен содержать конструктор для создания точки по двум аргументам типа double, а также методы getX() и getY() для получения координат).

В зависимости от варианта необходимо также ввести пользовательский тип данных для работы с определенной фигурой: «Треугольник», «Прямоугольник», «Квадрат», «Ромб» или «Трапеция». Для класса фигуры необходимо предусмотреть инкапсуляцию внутренних элементов и реализовать открытые методы для работы с элементами, для вычисления площади и периметра фигуры (использовать имена **getArea** и **getPerimeter**). Класс фигуры должен содержать конструктор для создания объекта по точкам (например, Square(Point p1, Point p2, Point p3, Point p4)). При создании фигуры точки могут быть заданы в произвольном порядке. Для получения точек класс фигуры должен содержать метод **getPoints()**, возвращающий массив из 4-х точек.

В зависимости от варианта необходимо реализовать вычислительные функции для работы с точками и фигурами, которые решают определенные задачи (см. ниже в таблице). Для структурирования программы используйте несколько файлов.

Для проверки корректности работы вычислительных функций необходимо реализовать:

- консольное меню с возможностью ввода данных пользователем;
- тестовые сценарии с заранее подготовленными данными для проверки вычислительных функций с выводом на консоль ответа от проверяемой функции и правильного ответа.

Функциональность по вариантам

Номер варианта определяем по формуле:

int V = (10 * int(c1) + int(c2)) % 12, где **c1** – первая буква фамилии на *английском* языке в *верхнем* регистре, **c2** – первая буква имени на *английском* языке в *верхнем* регистре.

Указывать номер варианта в названии архива с проектом (в формате: ФамилияИО_Группа_Вариант, например, ИвановАА_ 5130903_30001_11).

Вариант	Типы данных	Функции по вариантам
0	Point, Trapeze	L1, P1, F2
1	Point, Triangle	L2, P2, T3
2	Point, Triangle	L3, P3, T4
3	Point, Rectangle	L4, P4, F2
4	Point, Rectangle	L2, P1, F3
5	Point, Square	L3, P2, F2
6	Point, Square	L4, P3, F4
7	Point, Rhomb	L1, P4, F2
8	Point, Rhomb	L2, P1, F4
9	Point, Trapeze	L3, P2, F3
10	Point, Triangle	L4, P4, T2
11	Point, Triangle	L2, P4, T1

Вычислительные функции

Вычислительные функции необходимо реализовать в соответствие с предложенной сигнатурой, соблюдая названия и типы аргументов. Обратите внимание, что в задачах F1-F4 типы аргументов и\или тип возврата необходимо изменить с учетом варианта.

№	Описание функции
L1	Принадлежат ли точки одной прямой? **bool inLine(Point points[], int size);**
L2	Нахождение *максимальной группы точек*, которые лежат на одной прямой (нет другой группы, состоящей из большего количества точек, которые лежат на одной прямой). Результирующее значение - количество точек в найденной группе, массив indices - индексы найденных точек. **int getPointsInLine(Point* points, int size, int** indices**);
L3	Найти все тройки точек массива, через которые можно провести прямую линию. Возвращаемое значение - число найденных троек, **indices** - двумерный массив с индексами точек (indices[1][2] - третья точка второй линии). **int countLines(Point* points, int size, int*** indices**);
L4	Нахождение двух пар точек, которые определяют параллельные линии. Аргументы (p11, p12) и (p21, p22) соответствуют парам точек, которые задают параллельные линии. **bool getParallelLines(Point* points, int size, Point& p11, Point& p12, Point& p21, Point& p22);**
P1	Найти такую точку, что окружность радиуса R с центром в этой точке содержит максимальное число точек заданного множества. **Point getMaxCirclePoint(Point* points, int size, int R)**
P2	Найти такую точку, сумма расстояний от которой до остальных точек множества максимальна. **Point getFarestPoint(Point* points, int size)**
P3	Найти такую точку, сумма расстояний от которой до остальных точек множества минимальна. **Point getClosestPoint(Point* points, int size)**
P4	Найти такую точку, что окружность радиуса R с центром в этой точке содержит минимальное число точек заданного множества. **Point getMinCirclePoint(Point* points, int size, int R)**
T1	Нахождение всех прямоугольных треугольников. selected - массив с найденными треугольниками (данные из t копируются в selected), размер массива возвращается функцией.

	int getRectTriangles(Triangles t[], int size, Triangles selected);**
T2	Нахождение всех равнобедренных треугольников. selected - массив с найденными треугольниками (данные из t копируются в selected), размер массива возвращается функцией. **int getIsoscelesTriangles(Triangles t[], int size, Triangles** selected);**
T3	Найти три точки, образующие треугольник наибольшего периметра **Triangle getMaxLengthTriangle(Point* points, int size)**
T4	Найти три точки, образующие треугольник наименьшего периметра. Возвращается периметр найденного треугольника, triangle заполняется найденным треугольником. **int getMinLengthTriangle(Point* points, int size, Triangle& triangle)**
F1	Нахождение фигуры наибольшего периметра, которую можно сконструировать из произвольных 4-х точек массива (в зависимости от варианта для типа результата вместо Figure использовать Rectangle, Square, Rhomb или Trapeze; название функции не изменять. Фигура может быть ориентирована под любым углом к осям координат. **Figure getMaxFigure(Point* points, int size);**
F2	Создание фигуры по точкам (в зависимости от варианта для типов аргументов вместо Figure использовать Rectangle, Square, Rhomb или Trapeze; название функции не изменять). Передается массив из 4-х точек в произвольном порядке. Фигура может быть ориентирована под любым углом к осям координат. **bool getFigure(Point points[], Figure& figure);**
F3	Для заданного массива фигур найти пару фигур, центры которых наиболее близки к друг другу (в зависимости от варианта для типов аргументов вместо Figure использовать Rectangle, Square, Rhomb или Trapeze; название функции не изменять) **void getClosestFigures(Figure figures[], int size, Figure& f1, Figure& f2);**
F4	Проверка пересечения двух фигур (в зависимости от варианта для типов аргументов вместо Figure использовать Rectangle, Square, Rhomb или Trapeze; название функции не изменять). Фигура может быть ориентирована под любым углом к осям координат. **bool checkFigureCrossing(const Figure& f1, const Figure& f2)**