```
(defun parB (l1 l2)
        (print "Пункт B")
        (print "Парабола 1")
        (print (build_parabola l1))
        (print "Парабола 2")
        (print (build_parabola l2))
        (bdop (build_parabola l1)(build_parabola l2))
)

(defun bdop (coef1 coef2)(cond
      ((and (listp coef1) (listp coef2))
        (print "Пересечение с осью x параболы 1")
        (print (x_intersect coef1))
        (print "Пересечение с осью x параболы 2")
        (print (x_intersect coef2))
        (print "Пересечение парбол (точки если есть)")
        (print (parabolas_intersect coef1 coef2))
        (print "Функция, которая в любой точке X принимает значение, равное среднему
арифметическому значений заданных парабол в этой точке")
        (print (arith_mean coef1 coef2))
      )
      ((listp coef1)
        (print "Пересечение с осью x параболы 1")
        (print (x_intersect coef1))
      )
      ((listp coef2)
        (print "Пересечение с осью x параболы 2")
        (print (x_intersect coef2))
      )
))

(defun build_parabola (set)(cond
   ((= (full_det (caar set) (caadr set) (caaddr set)) 0) "Невозможно построить параболу!")
   ((= (/ (det1 (car set)(cadr set)(caddr set)) ( full_det (caar set)(caadr set)(caaddr set))) 0)
                          "Невозможно построить параболу!")
   (T (list (/ (det1 (car set) (cadr set) (caddr set)) (full_det (caar set) (caadr set) (caaddr set)))
         (/ (det2 (car set) (cadr set) (caddr set)) (full_det (caar set) (caadr set) (caaddr set)))
         (/ (det3 (car set) (cadr set) (caddr set)) (full_det (caar set) (caadr set) (caaddr set)))))
))

(defun full_det (x1 x2 x3)(* (- x1 x2) (- x2 x3) (- x3 x1)))

(defun det1 (p1 p2 p3)
   (- (* (car p1) (- (cadr p3) (cadr p2)))
   (* (car p2) (- (cadr p1) (cadr p3)))
   (* (car p3) (- (cadr p2) (cadr p1))))
)
```

```
(defun det2 (p1 p2 p3)
   (- (* (cadr p1) (- (*(car p3)(car p3)) (*(car p2)(car p2))))
   (* (cadr p2) (- (*(car p1)(car p1)) (*(car p3)(car p3))))
         (* (cadr p3) (- (*(car p2)(car p2)) (*(car p1)(car p1)))))
)

(defun det3 (p1 p2 p3)
   (- (* (cadr p1) (- (*(car p2)(car p2)(car p3)) (*(car p3)(car p3)(car p2))))
   (* (cadr p2) (- (*(car p3)(car p3)(car p1)) (*(car p1)(car p1)(car p3))))
   (* (cadr p3) (- (*(car p1)(car p1)(car p2)) (*(car p2)(car p2)(car p1)))))
)


(defun x_intersect (coef)(cond
      ((< (- (* (cadr coef) (cadr coef)) (* 4 (car coef) (caddr coef))) 0) "Нет пересечений параболы
с осью X!")
      ((= (- (* (cadr coef) (cadr coef)) (* 4 (car coef) (caddr coef))) 0) (list (list (/ (- (cadr coef)) (* 2
(car coef))) 0)))
      (T (list (list (/ (+ (- (cadr coef)) (sqrt (- (* (cadr coef) (cadr coef)) (* 4 (car coef) (caddr
coef))))) (* 2 (car coef))) 0)
            (list (/ (- (- (cadr coef)) (sqrt (- (* (cadr coef) (cadr coef)) (* 4 (car coef) (caddr coef)))))
(* 2 (car coef))) 0)))
))

(defun parabolas_intersect (coef1 coef2)(cond
      ((and (= (car coef1) (car coef2))
       (= (cadr coef1) (cadr coef2))
       (= (caddr coef1) (caddr coef2))) "Параболы совпадают!")
      ((and (= (car coef1) (car coef2))
       (= (cadr coef1) (cadr coef2))) "Параболы не пересекаются!")
      ((= (- (car coef1) (car coef2)) 0)
       (solve_same_a_case coef1 coef2))
      (T (solve_different_a_case coef1 coef2))
))


(defun solve_same_a_case (coef1 coef2)(cond
      ((= (- (cadr coef1) (cadr coef2)) 0) "Параболы не пересекаются!")
      (T (list (list (/ (- (- (caddr coef1) (caddr coef2)))
                  (- (cadr coef1) (cadr coef2)))
               (+ (* (car coef1)
                     (/ (- (- (caddr coef1) (caddr coef2)))
                        (- (cadr coef1) (cadr coef2)))
                     (/ (- (- (caddr coef1) (caddr coef2)))
                        (- (cadr coef1) (cadr coef2))))
                  (* (cadr coef1)
                     (/ (- (- (caddr coef1) (caddr coef2)))
                        (- (cadr coef1) (cadr coef2))))
```

```
                    (caddr coef1)))))
))

(defun solve_different_a_case (coef1 coef2)
   (process_discriminant_result
      (- (car coef1) (car coef2))
      (- (cadr coef1) (cadr coef2))
      (- (caddr coef1) (caddr coef2))
      coef1)
)

(defun process_discriminant_result (a b c coef1)(cond
      ((< (- (* b b) (* 4 a c)) 0) "Параболы не пересекаются!")
      ((= (- (* b b) (* 4 a c)) 0)
       (list (single_intersection_point a b coef1)))
      (T (list (first_intersection_point a b (- (* b b) (* 4 a c)) coef1)
            (second_intersection_point a b (- (* b b) (* 4 a c)) coef1)))
))

(defun single_intersection_point (a b coef1)
   (list (/ (- b) (* 2 a))
      (+ (* (car coef1)
            (/ (- b) (* 2 a))
            (/ (- b) (* 2 a)))
         (* (cadr coef1) (/ (- b) (* 2 a)))
         (caddr coef1)))
)

(defun first_intersection_point (a b discriminant coef1)
   (list (/ (+ (- b) (sqrt discriminant)) (* 2 a))
      (+ (* (car coef1)
            (/ (+ (- b) (sqrt discriminant)) (* 2 a))
            (/ (+ (- b) (sqrt discriminant)) (* 2 a)))
         (* (cadr coef1) (/ (+ (- b) (sqrt discriminant)) (* 2 a)))
         (caddr coef1)))
)

(defun second_intersection_point (a b discriminant coef1)
   (list (/ (- (- b) (sqrt discriminant)) (* 2 a))
      (+ (* (car coef1)
            (/ (- (- b) (sqrt discriminant)) (* 2 a))
            (/ (- (- b) (sqrt discriminant)) (* 2 a)))
         (* (cadr coef1) (/ (- (- b) (sqrt discriminant)) (* 2 a)))
         (caddr coef1)))
)

(defun arith_mean (coef1 coef2)
   (cond
```

```
      ((and (= (+ (car coef1) (car coef2)) 0) (= (+ (cadr coef1) (cadr coef2)) 0)) (list "Y=" (/ (+
(caddr coef1) (caddr coef2)) 2)))
      ((= (+ (car coef1) (car coef2)) 0) (list "Y=" (/ (+ (cadr coef1) (cadr coef2)) 2) "x+" (/ (+ (caddr
coef1) (caddr coef2)) 2)))
      (T (list "Y=" (/ (+ (car coef1) (car coef2)) 2) "x^2+" (/ (+ (cadr coef1) (cadr coef2)) 2) "x+" (/
(+ (caddr coef1) (caddr coef2)) 2)))
   )
)
;; 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)))

(defun build_triangle(l) (build_triangles (free_points (rem_dup l))))
(defun rem_dup (l)(cond
((null l) nil)
((find_dup (car l) (cdr l)) (rem_dup (cdr l)))
(T (cons (car l) (rem_dup (cdr l))))
))
(defun find_dup (e l)(cond
((null l) nil)
((equal e (car l)) T)
(T (find_dup e (cdr l)))
))
(defun build_triangles (l)(cond
((null l) nil)
(T (append (check_triangles (car l)) (build_triangles (cdr l))))
))
(defun check_triangles (tri)(cond
((null tri) nil)
((= (area tri) 0) nil)
(T (list tri))
))
(defun area (tri)(abs(/ (+ (* (caar tri) (- (cadadr tri) (cadr(caddr tri))))
(* (caadr tri) (- (cadr(caddr tri)) (cadar tri)))
(* (caaddr tri) (- (cadar tri) (cadadr tri)))) 2)))
```

```
(defun free_points (l)(cond
((null (cddr l)) nil)
(T (append (free_dop (car l) (cdr l)) (free_points (cdr l))))
))
(defun free_dop (a l)(cond
((null (cdr l)) nil)
(T (append (free_ddop a (car l) (cdr l)) (free_dop a (cdr l))))
))
(defun free_ddop (a b l)(cond
((null l) nil)
(T (cons (list a b (car l)) (free_ddop a b (cdr l))))
))
(defun distance (p1 p2)
(sqrt (+ (* (- (car p2) (car p1)) (- (car p2) (car p1)))
(* (- (cadr p2) (cadr p1)) (- (cadr p2) (cadr p1))))))
(defun approx_equal (a b) (< (abs (- a b)) 0.001))
(defun equilateral (l)(cond
((null l) nil)
((equilateral1 (car l)) (cons (car l) (equilateral (cdr l))))
(T (equilateral (cdr l)))
))
(defun equilateral1 (l)(cond
((and (approx_equal (distance (car l) (cadr l)) (distance (car l) (caddr l)))
(approx_equal (distance (car l) (cadr l)) (distance (cadr l) (caddr l)))) T)
))
(defun isosceles (l)(cond
((null l) nil)
((isosceles1 (car l)) (cons (car l) (isosceles (cdr l))))
(T (isosceles (cdr l)))
))
(defun isosceles1 (l)(cond
((or (approx_equal (distance (car l) (cadr l)) (distance (car l) (caddr l)))
(approx_equal (distance (car l) (cadr l)) (distance (cadr l) (caddr l)))
(approx_equal (distance (cadr l) (caddr l)) (distance (car l) (caddr l)))) T)
))
(defun right_triangles (l)(cond
((null l) nil)
((right_triangle_p (car l)) (cons (car l) (right_triangles (cdr l))))
(T (right_triangles (cdr l)))
))
(defun right_triangle_p (tri)
(right_triangle_p1 (distance (car tri) (cadr tri))
(distance (cadr tri) (caddr tri))
(distance (caddr tri) (car tri))))
(defun right_triangle_p1 (a b c)
(or (approx_equal (+ (* a a) (* b b)) (* c c))
(approx_equal (+ (* a a) (* c c)) (* b b))
(approx_equal (+ (* b b) (* c c)) (* a a))))
(defun similar_triangles (l)(cond
((null l) nil)
(T (append (find_similar (car l) (cdr l)) (similar_triangles (cdr l))))
))
(defun find_similar (t1 l)(cond
((null l) nil)
((similar_p t1 (car l)) (cons (list t1 (car l)) (find_similar t1 (cdr l))))
(T (find_similar t1 (cdr l)))
))
(defun similar_p (t1 t2)
(similar_check (sort3 (distance (car t1) (cadr t1)) (distance (cadr t1) (caddr t1)) (distance (caddr t1)
(car t1)))
```

```
(sort3 (distance (car t2) (cadr t2)) (distance (cadr t2) (caddr t2)) (distance (caddr t2) (car
t2)))))
(defun similar_check (s1 s2)(cond
((<= (car s2) 0) nil)
((and (approx_equal (/ (car s1) (car s2)) (/ (cadr s1) (cadr s2)))
(approx_equal (/ (car s1) (car s2)) (/ (caddr s1) (caddr s2)))) T)
))
(defun sort3 (a b c)(cond
((and (<= a b) (<= b c)) (list a b c))
((and (<= a c) (<= c b)) (list a c b))
((and (<= b a) (<= a c)) (list b a c))
((and (<= b c) (<= c a)) (list b c a))
((and (<= c a) (<= a b)) (list c a b))
(T (list c b a))
))

(defun equal_triangles (similar_pairs)(cond
    ((null similar_pairs) nil)
    ((equal_side_p (caar similar_pairs) (cadar similar_pairs))
     (cons (car similar_pairs) (equal_triangles (cdr similar_pairs))))
    (T (equal_triangles (cdr similar_pairs)))
))

(defun equal_side_p (t1 t2)
  (approx_equal (car (sort3 (distance (car t1) (cadr t1))
                            (distance (cadr t1) (caddr t1))
                            (distance (caddr t1) (car t1))))
                (car (sort3 (distance (car t2) (cadr t2))
                            (distance (cadr t2) (caddr t2))
                            (distance (caddr t2) (car t2)))))
)

(defun triangleC (l)
(print "Пункт C")
(print "Треугольники:") (print (build_triangle l))
(print "Равнобедренные:") (print (isosceles (build_triangle l)))
(print "Равносторонние:") (print (equilateral (build_triangle l)))
(print "Прямоугольные:") (print (right_triangles (build_triangle l)))
(print "Подобные:") (print (similar_triangles (build_triangle l)))
(print "Равные:") (print (equal_triangles (similar_triangles (build_triangle l))))  ; ← новая строка
)
```