Отчет по лабораторной работе № 5 по курсу «Функциональное программирование»

Студент группы 8О-307 МАИ *Спиридонов Кирилл*, №18 по списку
Контакты: vo-ro@list.ru
Работа выполнена: 15.05.22

Преподаватель: Иванов Дмитрий Анатольевич, доц. каф. 806
Отчет сдан:
Итоговая оценка:
Подпись преподавателя:

1. Тема работы

Обобщённые функции, методы и классы объектов.

2. Цель работы

Научиться определять простейшие классы, порождать экземпляры классов, считывать и изменять значения слотов, научиться определять обобщённые функции и методы

3. Задание (вариант №5.39)

Определить обычную функцию line-intersections, принимающий один аргумент - список отрезков (экземпляров класса line). Причём концы отрезков могут задаваться как в декартовых (экземплярами cart), так и в полярных координатах (экземплярами polar).
Функция должна возвращать список всех точек взаимного пересечения отрезков между собой.
Результирующие точки могут быть получены либо в декартовых, либо в полярных координатах - на усмотрение выполняющего задание. Точки пересечения отрезков за их границами (как пересечения прямых) должны быть исключены из результирующего списка.
"Вырожденные"случаи: параллельность, взаимное наложение и т.п. - следует исключить из рассмотрения и считать, что такого во входных данных не бывает.

(setq lines (list (make-instance 'line
:start (make-instance 'cart-или-polar ...)
:end (make-instance 'cart-или-polar ...))
...))

(line-intersections lines)
=> (cart-или-polar1 ...)

4. Оборудование студента

Процессор Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz, память: 8192Gb, разрядность системы: 64.

5. Программное обеспечение

ОС Ubuntu 20.04 LTS, среда LispWorks Personal Edition 7.1.2

6. Идея, метод, алгоритм

Идея алгоритма простая. Для начала необходимо определить, пересекаются ли отрезки. Необходимое и достаточное условие пересечения, которое должно быть соблюдено для обоих отрезков следующее: конечные точки одного из отрезков должны лежать в разных полуплоскостях, если разделить плоскость линией, на которой лежит второй из отрезков. Так как все вектора лежат на плоскости X-Y, то их векторное произведение будет иметь ненулевой только компоненту Z, соответственно и отличие произведений векторов будет только в этой компоненте. Поэтому мы можем умножить попарно-векторно вектор разделяющего отрезка на векторы направленные от начала разделяющего отрезка к обеим точкам проверяемого отрезка. Если компоненты Z обоих произведений будет иметь различный знак, значит один из углов меньше 0 но больше -180, а второй больше 0 и меньше 180, соответственно точки лежат по разные стороны от прямой. Если компоненты Z обоих произведений имеют одинаковый знак, следовательно и лежат они по одну сторону от прямой. Затем нам необходимо повторить операцию для другого отрезка и прямой, и убедиться в том, что расположение его конечных точек также удовлетворяет условию. Если оказалось, что отрезки пересекаются, то из концов одного из отрезков опускаем перпендикуляры

на другой отрезок. Там у нас получаются подобные треугольники. И отсюда находим координаты:
Px = Cx + (Dx-Cx)*|Z1|/|Z2-Z1|;
Py = Cy + (Dy-Cy)*|Z1|/|Z2-Z1|;

7. Сценарий выполнения работы

8. Распечатка программы и её результаты

8.1. Исходный код

```
; точка в полярных координатах
(defclass polar ()
 ((radius :initarg :radius :accessor radius)  ; длина >=0
  (angle  :initarg :angle  :accessor angle)))

(defmethod print-object ((p polar) stream)
  (format stream "[POLAR radius ~d angle ~d]"
          (radius p) (angle p)))


; точка в декартовых координатах
(defclass cart ()                  ; имя класса и надклассы
 ((x :initarg :x :reader cart-x)   ; дескриптор слота x
  (y :initarg :y :reader cart-y))) ; дескриптор слота y

(defmethod print-object ((c cart) stream)
  (format stream "[CART x ~d y ~d]"
          (cart-x c) (cart-y c)))

(defmethod cart-x ((p polar))
  (* (radius p) (cos (angle p))))

(defmethod cart-y ((p polar))
  (* (radius p) (sin (angle p))))

(defgeneric to-cart (arg)
 (:documentation Преобразование" аргумента в декартову систему.")
 (:method ((c cart))
  c)
```

```
 (:method ((p polar))
  (make-instance 'cart
                 :x (cart-x p)
                 :y (cart-y p))) )

(defclass line ()
 ((start :initarg :start :accessor line-start)
  (end   :initarg :end   :accessor line-end)))

(defmethod print-object ((lin line) stream)
  (format stream ОТРЕЗОК"[ ~s ~s]"
          (line-start lin) (line-end lin)))

(defgeneric vector-product (vec1 vec2)
  (:documentation Векторное" произведение двух векторов vec1 и vec2 на
   плоскости. Возращает координату z полученного вектора.") )

(defgeneric find-projection-x (pt1 pt2)
      (:documentation Находит" проекцию на ось x вектора, имеющего
   начало точку pt1 и конец pt2.") )

(defmethod find-projection-x ((pt1 cart) (pt2 cart))
  (- (cart-x pt2) (cart-x pt1)))

(defmethod find-projection-x ((pt1 polar) (pt2 cart))
  (- (cart-x (to-cart pt2)) (cart-x pt1)))

(defmethod find-projection-x ((pt1 polar) (pt2 polar))
  (- (cart-x (to-cart pt2)) (cart-x (to-cart pt1))))

(defmethod find-projection-x ((pt1 cart) (pt2 polar))
  (- (cart-x pt2) (cart-x (to-cart pt1))))

(defgeneric find-projection-y (pt1 pt2)
      (:documentation Находит" проекцию на ось y вектора, имеющего
   начало точку pt1 и конец pt2.") )

(defmethod find-projection-y ((pt1 cart) (pt2 cart))
  (- (cart-y pt2) (cart-y pt1)))

(defmethod find-projection-y ((pt1 polar) (pt2 cart))
  (- (cart-y (to-cart pt2)) (cart-y pt1)))
```

```
(defmethod find-projection-y ((pt1 polar) (pt2 polar))
  (- (cart-y (to-cart pt2)) (cart-y (to-cart pt1))))

(defmethod find-projection-y ((pt1 cart) (pt2 polar))
  (- (cart-y pt2) (cart-y (to-cart pt1))))

(defmethod vector-product ((vec1 line) (vec2 line))
  (let ((vec1-proj-x (find-projection-x (line-start vec1)
   (line-end vec1)))
    (vec1-proj-y (find-projection-y (line-start vec1) (line-end
   vec1)))
    (vec2-proj-x (find-projection-x (line-start vec2) (line-end
   vec2)))
    (vec2-proj-y (find-projection-y (line-start vec2) (line-end
   vec2))))
    (- (* vec1-proj-x vec2-proj-y) (* vec1-proj-y vec2-proj-x))))

(defun check-intersection (l1 l2)
  (let ((z1 (vector-product l1 (make-instance 'line
                  :start (line-start l1)
                  :end (line-start l2))))
    (z2 (vector-product l1 (make-instance 'line
                  :start (line-start l1)
                  :end (line-end l2))))
    (z3 (vector-product l2 (make-instance 'line
                  :start (line-start l2)
                  :end (line-start l1))))
    (z4 (vector-product l2 (make-instance 'line
                  :start (line-start l2)
                  :end (line-end l1)))))
    (if (and (< (* z1 z2) 0) (< (* z3 z4) 0))
      (let ((cx (cart-x (line-start l2)))
        (dx (cart-x (line-end l2)))
        (cy (cart-y (line-start l2)))
        (dy (cart-y (line-end l2))))
      (make-instance 'cart :x (+ cx (/ (* (- dx cx) (abs z1))
   (abs (- z2 z1)))) :y (+ cy (/ (* (- dy cy) (abs z1)) (abs (-
   z2 z1))))))
      NIL)))

(defun line-intersections (lines)
```

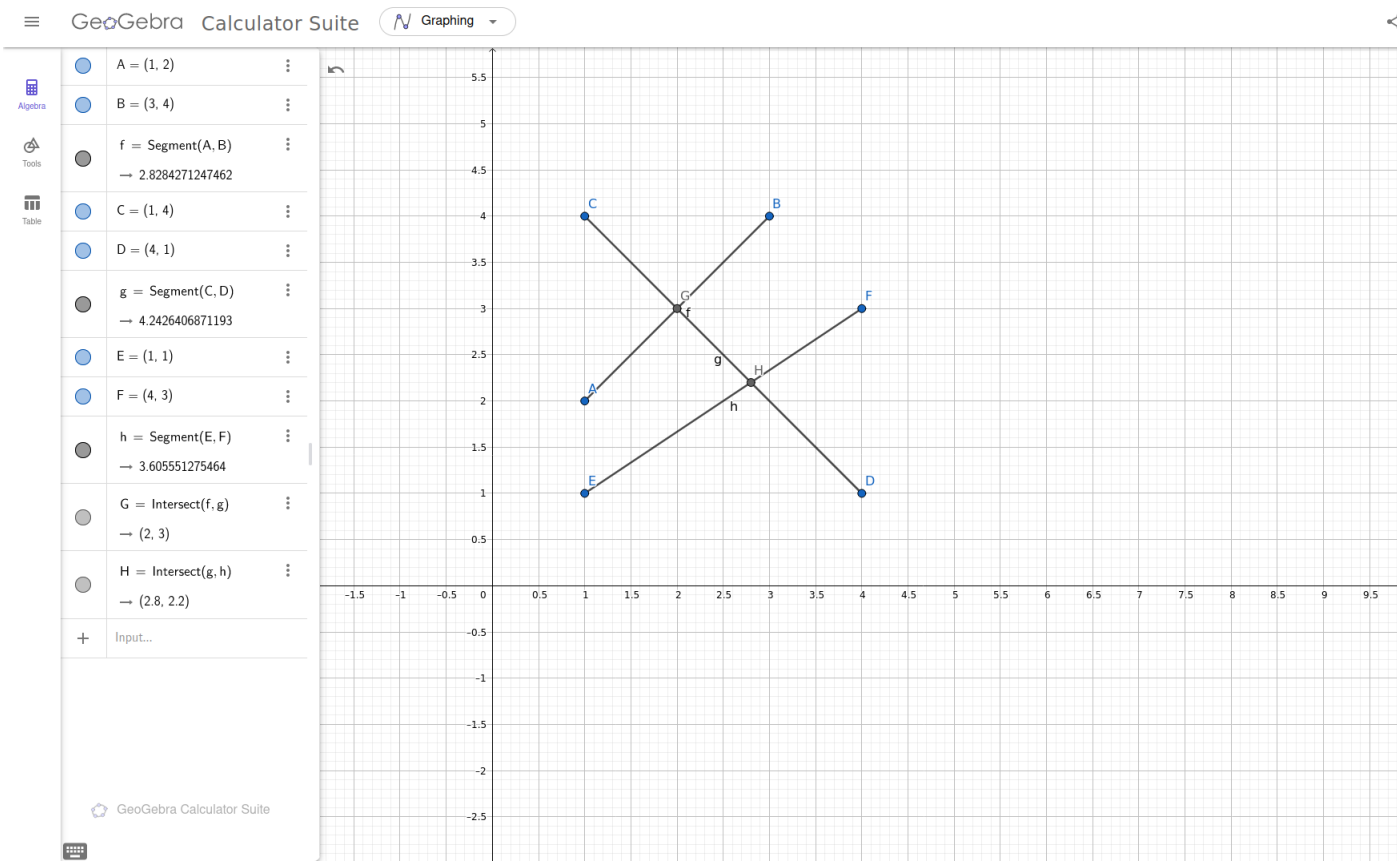
```
  (let ((res (list)))
    (loop for i from 0 below (length lines) do
    (when (= i (- (length lines) 1)) (return res))
    (loop for j from (+ i 1) below (length lines) do
        (let ((ans (check-intersection (nth i lines) (nth j
   lines))))
      (if ans
      (setq res (append res (list i j ans)))
        ))))
    ))

(setq lines (list (make-instance 'line
                  :start (make-instance 'cart :x 1 :y 4)
                  :end (make-instance 'cart :x 4 :y 1))
          (make-instance 'line
                  :start (make-instance 'cart :x 1 :y 1)
                  :end (make-instance 'cart :x 4 :y 3))
          (make-instance 'line
                  :start (make-instance 'cart :x 1 :y 2)
                  :end (make-instance 'cart :x 3 :y 4))
          ))

(line-intersections lines)


(setq lines (list (make-instance 'line
                  :start (make-instance 'polar :radius 10 :angle
   2.094)
                  :end (make-instance 'polar :radius 12 :angle
   2.792))
          (make-instance 'line
                  :start (make-instance 'cart :x -14 :y 9)
                  :end (make-instance 'polar :radius 6 :angle 1))
          (make-instance 'line
                  :start (make-instance 'cart :x 0 :y 4)
                  :end (make-instance 'cart :x 4 :y 10))
          ))

(line-intersections lines)
```

8.2. Результаты работы

Результат выводиться в виде (list i j ans ...), где i и j номера отрезков в исходном списке, поданном на вход, ans - точка, в которой эти отрезки пересекаются в декартовой системе координат. Так же для наглядности я прикрепил картнинки.

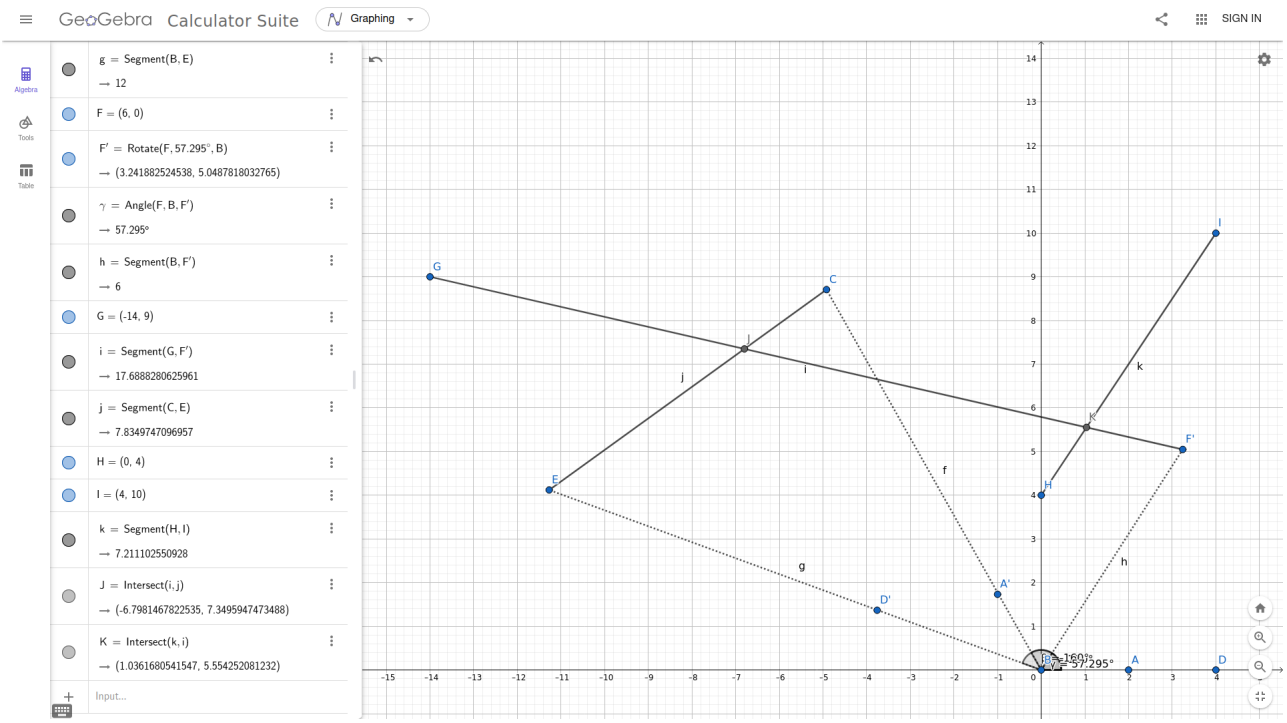
(setq lines (list (make-instance 'line
:start (make-instance 'cart :x 1 :y 4)
:end (make-instance 'cart :x 4 :y 1))
(make-instance 'line
:start (make-instance 'cart :x 1 :y 1)
:end (make-instance 'cart :x 4 :y 3))
(make-instance 'line
:start (make-instance 'cart :x 1 :y 2)
:end (make-instance 'cart :x 3 :y 4))
))
([ОТРЕЗОК [CART x 1 y 4][CART x 4 y 1]] [ОТРЕЗОК [CART x 1 y 1] [CART x 4 y 3]] [ОТРЕЗОК [CART x 1 y 2] [CART x 3 y 4]])

CL-USER 35 : 1 > (line-intersections lines)
(0 1 [CART x 14/5 y 11/5] 0 2 [CART x 2 y 3])



(setq lines (list (make-instance 'line
:start (make-instance 'polar :radius 10 :angle 2.094)
:end (make-instance 'polar :radius 12 :angle 2.792))
(make-instance 'line
:start (make-instance 'cart :x -14 :y 9)
:end (make-instance 'polar :radius 6 :angle 1))
(make-instance 'line
:start (make-instance 'cart :x 0 :y 4)
:end (make-instance 'cart :x 4 :y 10))
))
([ОТРЕЗОК [POLAR radius 10 angle 2.094] [POLAR radius 12 angle 2.792]]
[ОТРЕЗОК [CART x -14 y 9] [POLAR radius 6 angle 1]] [ОТРЕЗОК [CART
x 0 y 4] [CART x 4 y 10]])

CL-USER 27 > (line-intersections lines)
(0 1 [CART x -6.804698 y 7.3511076] 1 2 [CART x 1.0361822 y 5.5542736])



9. Дневник отладки

Дата	Событие	Действие по исправлению	Примечание

10. Замечания автора по существу работы

Вначале я хотел находить прямые, на которых лежат эти отрезки и по уравнениям этих прямых смотреть, пересекаются ли они. Но наткнулся в интернете на алгоритм выше и решил попробовать реализовать его. Общая сложность алгоритма $O(n^2)$, т.к. каждый отрезок проверяется со всеми остальными.

11. Выводы

В данной лабораторной работе я научился определять простейшие классы, порождать экземпляры классов, считывать и изменять значения слотов в Common Lisp. Так же в ходе выполнения лабораторной работы познакомился с новым алгоритмом для определения пересечения отрезков.