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

Студент группы 8О-307Б-18 МАИ *Токарев Никита*, №21 по списку
Контакты: tokarevnikita08@mail.ru
Работа выполнена: 18.04.2021

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

Изучить обобщённые функции, методы и классы объектов, а также методы работы с ними в Коммон Лисп.

## 3. Задание (вариант № 5.22)

Определить обычную функцию - предикат line-parallel2-p, возвращающий T, если два отрезка (экземпляра класса line) параллельны.

Причём концы отрезков могут задаваться как в декартовых (экземплярами cart), так и в полярных координатах (экземплярами polar).

## 4. Оборудование студента

Процессор Intel® Core™ i3-5005U CPU @ 2.00GHz × 4, память: 3,8 Gb, разрядность системы: 64.

## 5. Программное обеспечение

UBUNTU 18.04.5 LTS, компилятор sbcl

## 6. Идея, метод, алгоритм

Идея в том, чтобы сравнивать прямые по значению тангенса угла, так как прямую можно представить в виде y = kx + b(соответсвенно координаты должны быть декартовые). Также, если прямая расположена вдоль оси x, то при вычислении, k стремиться

к бесконечности(то есть происходит деление на ноль), поэтому данный случай также необходимо выделить в отдельный кейс.

- (defmethod cart-x ((p polar))) - В данной функции происходит перевод из полярных координат в декартовые).
- (defmethod line-x-const ((l line))) - $x_1? == x_2$
- (defmethod line-k ((line1 line))) - вычисление k; $k = (y_2 - y_1)/(x_2 - x_1)$.

# 7. Сценарий выполнения работы

- Анализ возможных реализаций поставленной задачи на common Lisp
- Изучение синтаксиса и основных функций работы со списками common Lisp
- Реализация поставленной задачи на common Lisp

# 8. Распечатка программы и её результаты

## 8.1. Исходный код

Значения тестовых функций представлено в исходном коде.

```
(defclass polar ()
    ((radius :initarg :radius :accessor radius)
    (angle  :initarg :angle  :accessor angle)))

(defclass cart ()
    ((x :initarg :x :reader cart-x)
    (y :initarg :y :reader cart-y)))

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

(defmethod cart-x ((p polar)) ; angle ~ (-pi,pi]
    (* (radius p) (cos (angle p))))

(defmethod cart-y ((p polar)) ; angle ~ (-pi,pi]
    (* (radius p) (sin (angle p))))

(defmethod line-k ((line1 line))
    (/ (- (cart-y (line-start line1)) (cart-y (line-end line1)))
    (- (cart-x (line-start line1)) (cart-x (line-end line1)))))

(defmethod line-x-const ((l line))
```

```
    (almost-equal (cart-x (line-start l)) (cart-x (line-end l))))

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

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

(defmethod print-object ((lin line) stream)
    (format stream "LINE (~s ~s)"
    (line-start lin) (line-end lin)))

(defun almost-equal (a b)
    (< (abs(- a b)) 0.001))

(defun line-parallel2-p (line1 line2)
    (if (or (line-x-const line1) (line-x-const line2))
        (if (and (line-x-const line1) (line-x-const line2)) T NIL)
        (if (almost-equal (line-k line1) (line-k line2)) T NIL)))

;(setq l1 (make-instance 'line :start (make-instance 'cart :x 1 :y 1)
:end   (make-instance 'cart :x 0 :y -1)))
;(setq l2 (make-instance 'line :start (make-instance 'cart :x 0 :y 2)
:end   (make-instance 'cart :x -1 :y 0)))
;(setq l2 (make-instance 'line :start (make-instance 'cart :x -1 :y 0)
:end   (make-instance 'cart :x  0 :y 1)))
;(setq l2 (make-instance 'line :start (make-instance 'polar :radius 1
:angle (/ pi 4)) :end   (make-instance 'cart :x  0 :y 2)))
;(setq l2 (make-instance 'line :start (make-instance 'polar :radius 0
:angle 0) :end   (make-instance 'cart :x  1 :y 1)))
;(setq l2 (make-instance 'line :start (make-instance 'cart :x 0 :y 2)
:end   (make-instance 'cart :x 0 :y 0)))
;(setq l2 (make-instance 'line :start (make-instance 'polar :radius 20
:angle (/ pi 2)) :end   (make-instance 'polar :radius 2 :angle (- (/ pi 2)))))
;(line-parallel2-p l1 l2)
```

## 8.2. Результаты работы

```
* (setq l1 (make-instance 'line :start (make-instance 'cart :x 1 :y 1)
:end   (make-instance 'cart :x 0 :y -1)))

LINE ([CART (x 1 y 1)] [CART (x 0 y -1)])
```

```
* (setq l2 (make-instance 'line :start (make-instance 'cart :x 0 :y 2)
:end   (make-instance 'cart :x -1 :y 0)))

LINE ([CART (x 0 y 2)] [CART (x -1 y 0)])
* (line-parallel2-p l1 l2)

T
* (setq l2 (make-instance 'line :start (make-instance 'cart :x -1 :y 0)
:end   (make-instance 'cart :x  0 :y 1)))

LINE ([CART (x -1 y 0)] [CART (x 0 y 1)])
* (line-parallel2-p l1 l2)

NIL
* (setq l2 (make-instance 'line :start (make-instance 'polar :radius 1
:angle (/ pi 4)) :end   (make-instance 'cart :x  0 :y 2)))

LINE ([POLAR (radius 1 angle 0.7853981633974483d0)] [CART (x 0 y 2)])
* (line-parallel2-p l1 l2)

NIL

LINE ([POLAR (radius 0 angle 0)] [CART (x 1 y 1)])
* (line-parallel2-p l1 l2)

NIL
* (setq l2 (make-instance 'line :start (make-instance 'cart :x 0 :y 2)
:end   (make-instance 'cart :x 0 :y 0)))

LINE ([CART (x 0 y 2)] [CART (x 0 y 0)])
* (setq l1 (make-instance 'line :start (make-instance 'cart :x 3 :y 2)
:end   (make-instance 'cart :x 3 :y 0)))

LINE ([CART (x 3 y 2)] [CART (x 3 y 0)])
* (line-parallel2-p l1 l2)

T
* (setq l2 (make-instance 'line :start (make-instance 'polar :radius 20
:angle (/ pi 2)) :end   (make-instance 'polar :radius 2 :angle (- (/ pi 2)))))

LINE ([POLAR (radius 20 angle 1.5707963267948966d0)]
[POLAR (radius 2 angle -1.5707963267948966d0)])
```

# 9. Дневник отладки

| Дата | Событие | Действие по исправлению | Примечание |
|---|---|---|---|
| 18.04.2021 | (defmethod line-k ((line1 line))) - ошибка из-за деления на 0 | Добавил поддержку вертикальных линий и дополнил главный предикат условием | |

# 10. Замечания автора по существу работы

Замечаний нет.

# 11. Выводы

В ходе данной работы мне удалось познакомиться с обобщёнными функциями, методами и классами в common Lisp. Lisp был создан за пару десятилетий до того момента, когда объектно-ориентированное программирование стало популярным. Хотел бы отметить, что Common Lisp является полноценным объектно-ориентированным языком и в данном языке заложены основные парадигмы/принципы объектно-ориентированного программирования.