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

Студент группы М8О-307-19 МАИ *Ежов Никита Павлович*, №9 по списку
Контакты: nikita.ejov2012@yandex.ru
Работа выполнена: 25.05.2022

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

1. Тема работы

Последовательности, массивы и управляющие конструкции Common Lisp.

2. Цель работы

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

3. Задание (вариант № 5.27)

Даны три точки (радиус-вектора), которые могут быть заданы как декартовыми координатами (экземплярами cart), так и полярными (экземплярами polar).

Задание: Определить обычную функцию, возвращающая четвёртую точку такую, что все четыре точки задают параллелограмм.

Результирующая вершина должна быть получена в декартовых координатах - в виде экземпляра класса cart.

(defvar *tolerance* 0.001)

(defun fourth-vertex-cart (v1 v2 v3) ...)

4. Оборудование студента

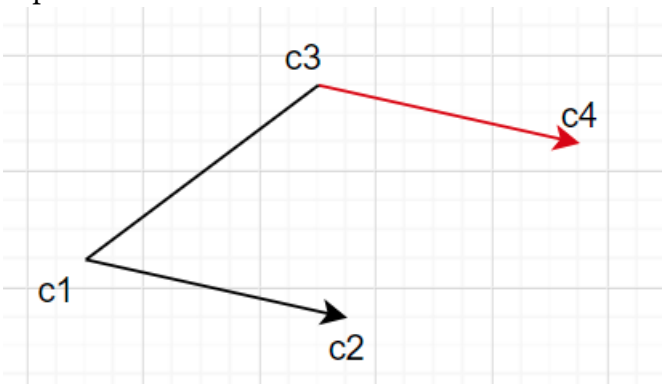
Процессор Intel i7-4770 (8) @ 3.9GHz, память: 16 Gb, разрядность системы: 64.

5. Программное обеспечение

ОС Kubuntu 20.04.4 LTS, комилятор GNU CLISP 2.49.92, текстовый редактор Visual Studio Code 1.67.1

6. Идея, метод, алгоритм

Идея крайне проста – к точке c3 нужно прибавить вектор, равный разности радиус векторов точек c2 и c1.



7. Сценарий выполнения работы

Проще всего воспользоваться предоставленными определениями классов polar и cart, функциями для вывода, приведения точек из декартовых координат в полярные и обратно, а так же для сложения точек. После этого, для окончательного выполнения работы, требуется создать оператор вычитания точек и непосредственно саму функцию для нахождения последней точки параллелограмма.

8. Распечатка программы и её результаты

8.1. Исходный код

```
;5.27

(defclass polar ()
  ((radius :initarg :radius :accessor radius)    ; длина >=0
   (angle  :initarg :angle  :accessor angle)))  ; угол (-;]

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

(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 radius ((c cart))
  (sqrt (+ (square (cart-x c))
           (square (cart-y c)))))

(defmethod angle ((c cart))
  (atan (cart-y c) (cart-x c)))

(defgeneric to-polar (arg)
 (:documentation Преобразование" аргумента в полярную систему.")
 (:method ((p polar))
  p)
 (:method ((c cart))
  (make-instance 'polar
                 :radius (radius c)
                 :angle (angle 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))) )

(defmethod sub2 ((c1 cart) (c2 cart))
  (make-instance 'cart
                 :x (- (cart-x c1) (cart-x c2))
                 :y (- (cart-y c1) (cart-y c2))))

(defmethod sub2 ((p1 polar) (p2 polar))
  (to-polar (sub2 (to-cart p1)
                  (to-cart p2))))
```

```
(defmethod sub2 ((c cart) (p polar))
  (sub2 c (to-cart p)))

(defmethod add2 ((c1 cart) (c2 cart))
  (make-instance 'cart
                 :x (+ (cart-x c1) (cart-x c2))
                 :y (+ (cart-y c1) (cart-y c2))))

(defmethod add2 ((p1 polar) (p2 polar))
  (to-polar (add2 (to-cart p1)
                  (to-cart p2))))

(defmethod add2 ((c cart) (p polar))
  (add2 c (to-cart p)))

(defgeneric mul2 (arg1 arg2)
 (:method ((n1 number) (n2 number))
  (* n1 n2)))

(defmethod fourth-vertex-cart((c1 cart) (c2 cart) (c3 cart))
    (add2 (sub2 c2 c1) c3))


(defmethod fourth-vertex-cart((p1 polar) (p2 polar) (p3 polar))
    (fourth-vertex-cart (to-cart p1) (to-cart p2) (to-cart p3)))
```

8.2. Результаты работы

```
[8]> (fourth-vertex-cart (make-instance 'cart :x 1 :y 1)
   (make-instance 'cart :x 1 :y 4) (make-instance 'cart :x 2 :y
   3))
[CART x 2 y 6]
[9]> (fourth-vertex-cart (make-instance 'polar :angle 0.79
   :radius 1.41) (make-instance 'polar :angle 1.33 :radius 4.12)
   (make-instance 'polar :angle 0.98 :radius 3.61))
[CART x 2.0009503 y 5.997628]
```

9. Дневник отладки

Дата	Событие	Действие по исправлению	Примечание

10. Замечания автора по существу работы

Это первая работу в данном курсе, в которой мне потребовалось проводить исследование не только лишь средств языка Common Lisp.

11. Выводы

Я познакомился с обобщёнными функциями, методами и классами объектов в Common Lisp. Описание классов напомнило язык C#, где нужно описывать getter и setter для полей. До этого я не использовал ООП вместе с функциональной парадигмой, в целом, я не заметил ощутимой разницу между ООП на Common Lisp и другими языками (не функциональными).