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

Студент группы 8О-306 Кондратьев Егор, № по списку 14.

Контакты: egor.kondratev27@gmail.com

Работа выполнена: 15.05.2022

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 обобщенные функции и методы

3. Задание (вариант № 5.45)

Определите обычную функцию с двумя параметрами:

p - многочлен, т.е. экземпляр класса polynom,

a - список действительных чисел (a_1 ... a_n), где n - степень многочлена p.

Функция должна возвращать список действительных чисел

(d_0 ... d_n), таких что:

$$P(x) = d_0 + d_1*(x-a_1) + d_2*(x-a_1)*(x-a_2) + ... + d_n*(x-a_1)*...*(x-a_n)$$

4. Оборудование студента

Ноутбук Asus ROG GL752VW, процессор QuadCore Intel Core i7-6700HQ, 3100 MHz , память 24474 МБ (DDR4 SDRAM), 64-разрядная система.

5. Программное обеспечение

ОС Windows 10, программа VSC и компилятор Steel Bank Common Lisp

6. Идея, метод, алгоритм

Выводим формулу. Дальше вычисляем. Основная функция func принимает на вход полином и список коэффициентов. Коэффициенты многочлена получены с помощью функции coefs, которая рекурсивно все проходит. Получаем итог.

7. Сценарий выполнения работы

8. Распечатка программы и ее результаты

Программа

```
;;; lab5 Egor Kondratev
;;; 5.45

(defclass polynom ()
    ((polunom-symbol :initarg :var1 :reader var1)
     (term-list :initarg :terms :reader terms)
    )
)

(defun make-term (&key order coeff)
    (list order coeff)
)

(defun order (term) (first term))
(defun coeff (term) (second term))

(defgeneric zeropp (arg)
 (:method ((n number))   ; (= n 0)
  (zerop n)))

(defgeneric minuspp (arg)
 (:method ((n number))   ; (< n 0)
  (minusp n)))

(defun mult-list (list)
    (mapcar #'(lambda(x) (reduce '* x)) list)
)

(defun get-last-n-elems (count list)
    (last list count)
)

(defun sum-list (list)
    (reduce '+ list)
)


(defun remove-last-el (list)
    (loop for i on list
        while (rest i)
```

```
            collect (first i)
    )
)


(defmethod print-object ((p polynom) stream)
  (format stream "[Polynomial (~s)
~:{~:[~:[+~;-~]~d~[~2*~;~s~*~:;~s^~d~]~;~]~}]"
          (var1 p)
          (mapcar (lambda (term)
                    (list (zeropp (coeff term))
                          (minuspp (coeff term))
                          (if (minuspp (coeff term))
                              (abs (coeff term))
                              (coeff term))
                          (order term)
                          (var1 p)
                          (order term)))
                  (terms p))))




(defun get-zeros (n)
    (make-list n :initial-element '0)
)

(defun cur-coef (cur next tail)
    (cond ((null next) (if (= 0 (order cur))
                           (cons (coeff cur) tail)
                           (cons (coeff cur) (append (get-zeros (order
cur)) tail)))
          )
          ((= (order cur) (1+ (order next))) (cons (coeff cur) tail))
          (t (cons (coeff cur) (append (get-zeros (1- (- (order cur) (order
next)))) tail)))
    )
)

(defun coefs (p)
    (if p (cur-coef (first p) (second p) (coefs (rest p))))
)
```

```
(defun combinations (count list)
    (cond
        ((zerop count) '(()))
        ((endp list)   '())
        (t (nconc (mapcar (let ((item (first list)))
                            (lambda (comb) (cons item comb)))
                            (combinations (1- count) (rest list)))
              (combinations count (rest list))))
    )
)

(defun mult (d a j)
    (cond
        ((oddp j) (* d (sum-list (mult-list (combinations j a)))))
        (t (* -1 (* d (sum-list (mult-list (combinations j a))))))
    )
)

(defun listd (j d a i)
    (if d (cons (mult (first d) (get-last-n-elems (- (list-length a) j) a)
i) (listd (1+ j) (rest d) a (1- i))))
)

(defun sum-mult (d a)
    (sum-list (listd 0 d a (list-length d)))
)

(defun func (p a)
    (let  ((b (coefs (terms p)))
           (d (list (first (coefs (terms p)))))
           )
        (loop for i in (rest b)
            do (nconc d (list (+ i (sum-mult d (remove-last-el a)))))
        )
    (reverse d)
    )
)

(defun ex1 ()
    (let ((pol (make-instance 'polynom :var1 'x :terms
```

```
        (list   (make-term :order 5 :coeff -2)
                (make-term :order 3 :coeff 4)
                (make-term :order 1 :coeff -6))))
        (l (list 1 2 3 2 1 1)))

    (print "Polynom:")
    (print pol)
    (print "List:")
    (print l)
    (print "Result:")
    (print (func pol l))
    (values))
)

(defun ex2 ()
    (let ((pol (make-instance 'polynom :var1 'x :terms
        (list   (make-term :coeff 1 :order 3)
                (make-term :coeff 2 :order 1)
                (make-term :coeff 1 :order 0))))
        (l (list 4 4 4)))

    (print "Polynom:")
    (print pol)
    (print "List:")
    (print l)
    (print "Result:")
    (print (func pol l))
    (values))
)
(defun ex3 ()
    (let ((pol (make-instance 'polynom :var1 'x :terms
        (list   (make-term :order 2 :coeff 5)
                (make-term :order 1 :coeff 3.3)
                (make-term :order 0 :coeff -7.4))))
        (l (list 1 1 1)))

    (print "Polynom:")
    (print pol)
    (print "List:")
    (print l)
    (print "Result:")
    (print (func pol l))
    (values))
```

```
)


;; (ex1)
;; (ex2)
;; (ex3)
```

Результаты

```
* (ex1)
"Polynom:"
[Polynomial (X) -2X^5+4X^3-6X]
"List:"
(1 2 3 2 1 1)
"Result:"
(-4 -40 -156 -78 -18 -2)

* (ex2)
"Polynom:"
[Polynomial (X) +1X^3+2X+1]
"List:"
(4 4 4)
"Result:"
(1 18 8 1)

* (ex3)
"Polynom:"
[Polynomial (X) +5X^2+3.3X-7.4]
"List:"
(1 1 1)
"Result:"
(0.9000001 13.3 5)
*
```

9. Дневник отладки

№	Дата, время	Событие	Действие по исправлению	Примечание
1				

10. Замечания автора по существу работы

11. Выводы

Выполняя эту работу, я научился работать с простейшими классами, порождать экземпляры классов, производить различные действия над ними.