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

Студент группы М8О-306 МАИ *Дубинин Артем*, №5 по списку
Контакты: rusartdub@gmail.com
Работа выполнена: 29.04.2020

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 №5.38)

Определите обобщённую функцию SUB2, производящую вычитание двух чисел либо многочленов.
(defgeneric sub2 (arg1 arg2)
...)
(defmethod sub2 ((p1 polynom) (p1 polynom))
...)

4. Оборудование студента

Ноутбук Dell Vostro 5568, процессор Intel Core i5-7200U @ 4x 3.1GHz, память: 8Gb, разрядность системы: 64.

5. Программное обеспечение

ОС Ubuntu 20.04 bionic, компилятор clisp, текстовый редактор Atom.

6. Идея, метод, алгоритм

Условимся, что полиномы будут только с переменной X. Пробегаемся по полиномам в порядке убываения степени. Если степень одного полинома больше, то мы просто записываем в ответ коэффицент этой степени и уменьшаем степень этого полинома на 1, знак коэффицента определяется так, если это вычитаемое то со знаком минусом, иначе со знаком плюс. Если степени одниковые то делаем вычитание коэффицентов, уменьшаем степень полиномов на 1 и записываем в ответ.

7. Сценарий выполнения работы

8. Распечатка программы и её результаты

8.1. Исходный код

```
(defclass polynom ()
  ((polynom-symbol :initarg :var1 :reader var1)
  ;; Разреженный список термов в порядке убывания степени
   (term-list :initarg :terms :reader terms)
  )
)

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

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

(defgeneric zerop1 (arg)
 (:method ((n number))    ; (= n 0)
  (zerop n)))

(defgeneric minusp1 (arg)
 (:method ((n number))    ; (< n 0)
  (minusp n)))

(defmethod print-object ((p polynom) stream)
  (format stream "~:{~:[~:[+~;-~]~d~[~2*~;~s~*~:;~s^~d~]~;~]~}"
          (mapcar (lambda (term)
                    (list (zerop1 (coeff term))
```

```
                        (minusp1 (coeff term))
                        (if (minusp1 (coeff term))
                            (abs (coeff term))
                            (coeff term))
                        (order term)
                        (var1 p)
                        (order term)))
                (terms p))))


(defgeneric sub2 (arg1 arg2)
 (:method ((n1 number) (n2 number))
  (- n1 n2)))

(defun traverse-lists (ans l1 l2)
  (if (and l1 l2)
    (let ((order1 (car (car l1)))
          (order2 (car (car l2)))
          (coeff1 (car (cdr (car l1))))
          (coeff2 (car (cdr (car l2)))))
      (if (= order1 order2)
        (progn
         (push (list order1 (- coeff1 coeff2)) ans)
         ; (print ans)
         (setf ans (traverse-lists ans (cdr l1) (cdr l2))))
        (if (< order1 order2)
          (progn
            (push (list order2 (- coeff2)) ans)
            ; (print ans)
            (setf ans (traverse-lists ans l1 (cdr l2))))
          (progn
            (push (list order1 coeff1) ans)
            ; (print ans)
            (setf ans (traverse-lists ans (cdr l1) l2)))
          )))
    (if l1
      (let ((order1 (car (car l1)))
            (coeff1 (car (cdr (car l1)))))
        (push (list order1 coeff1) ans)
        (setf ans (traverse-lists ans (cdr l1) l2)))
      (if l2
        (let ((order2 (car (car l2)))
```

```
            (coeff2 (car (cdr (car l2)))))
            (push (list order2 (- coeff2)) ans)
            (setf ans (traverse-lists ans l1 (cdr l2) ))))
    ))
  ans)


(defun sub2-aux (p1 p2)
  (let ((ans NIL))
    (reverse (traverse-lists ans (terms p1) (terms p2)))))

(defmethod sub2 ((p1 polynom) (p2 polynom))
  (let ((ans (make-instance 'polynom
                :var1 'x
                :terms (sub2-aux p1 p2))))
    ans))

; ; 5x^2 + 3.3x
; (defvar p1 (make-instance 'polynom
;           :var1 'x
;           :terms (list (make-term :order 2 :coeff 5)
;                        (make-term :order 1 :coeff 3.3)
;                        )))
; ; 4x^3 + 7x^2 + 2.3x - 7
; (defvar p2 (make-instance 'polynom
;           :var1 'x
;           :terms (list  (make-term :order 3 :coeff 4)
;                         (make-term :order 2 :coeff 7)
;                         (make-term :order 1 :coeff 2.3)
;                         (make-term :order 0 :coeff -7)
;                         )))




; (print p1)
; (print p2)
; (print (sub2 p1 p2))
```

8.2. Результаты работы

```
[1]> (load "main.lisp")
```

```
;; Загружается файл main.lisp ...
;; Загружен файл main.lisp
#P"/home/art/study/FP/lab_5/main.lisp"
[2]> (defvar p1 (make-instance 'polynom
          :var1 'x
          :terms (list (make-term :order 2 :coeff 5)
                       (make-term :order 1 :coeff 3.3)
                       )))
P1
[3]> (defvar p2 (make-instance 'polynom
          :var1 'x
          :terms (list  (make-term :order 3 :coeff 4)
                        (make-term :order 2 :coeff 7)
                        (make-term :order 1 :coeff 2.3)
                        (make-term :order 0 :coeff -7)
                        )))
P2
[4]> p1
+5X^2+3.3X
[5]> p2
+4X^3+7X^2+2.3X-7
[6]> (sub2 p1 p2)
-4X^3-2X^2+1.0X+7
[7]> (sub2 p2 p1)
+4X^3+2X^2-1.0X-7
```

9. Замечания автора по существу работы

Чтобы выполнить эту работу пришлось очень много искать информацию о том как работать с классами, методами и обощенными функциями в lisp'е.

10. Выводы

При выполнении работы я столкнулся с проблеми, что в лиспе, насколько я узнал нельзя списки передавать по ссылке, эта проблема затормозила меня на некоторое время, ну я нашел решение в виде рекурсии и последовательного присваивания. Так же было непривычно видеть классы в лиспе, так как синтаксис немного различается с языками более привычными.