Отчет по лабораторной работе № 3 по курсу «Функциональное программирование»

Студент группы М8О-307 МАИ *Безлуцкая Елизавета*, №2 по списку
Контакты: lizabezlutskaya@gmail.com
Работа выполнена: 03.04.2019

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

Тема работы

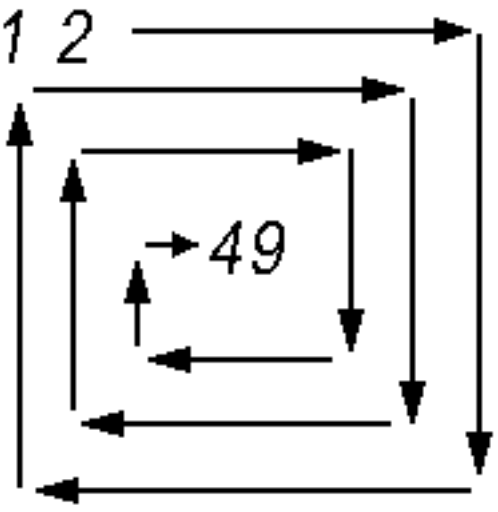
Последовательности, массивы и управляющие конструкции Коммон Лисп.

Цель работы

Научиться создавать векторы и массивы для представления матриц, освоить общие функции работы с последовательностями, инструкции цикла и нелокального выхода.

Задание (вариант №46)

Запрограммировать на языке Коммон Лисп функцию, принимающую в качестве единственного аргумента целое число n - порядок матрицы. Функция должна создавать и возвращать двумерный массив представляющий целочисленную квадратную матрицу порядка n, элементами которой являются числа $1, 2, ...n^2$, расположенные по спирали.



Оборудование студента

Процессор Intel Core i5-3230M 4 @ 2.6GHz, память: 350Gb, разрядность системы: 64.

Программное обеспечение

Ubuntu 16.04 LTS, clisp compiler

Идея, метод, алгоритм

В моём решении матрица условно делится на несколько квадратных рамок, вложенных друг в друга. Делается их обход с самой большой до самой маленькой, где каждая рамка – это четыре стороны квадрата, которые нужно обойти в направлениях для верхней стороны – слева-направо, правой стороны – сверху-вниз и т.д. по кругу.

Функция `spiral-matrix` исполняет следующие циклы:

- внешний для рамки
- четыре внутренних для каждой стороны рамки

Сценарий выполнения работы

Распечатка программы и её результаты

Исходный код

```
(defun print-matrix (matrix &optional (chars 3) stream)
  ;; Предполагаем, что требуется
  ;;  3 знака по умолчанию на каждый элемент,
  ;;  6 знаков на #2A и скобки.
  (let ((*print-right-margin* (+ 6 (* (1+ chars) ; плюс пробел
                                      (array-dimension matrix
 1)))))
    (pprint matrix stream)
    (values)))

(defun spiral-matrix (n)
    (let ((matrix (make-array (list n n)))
          (el 1))

        (dotimes (i (ceiling n 2))
              (loop for j upfrom i to (- n i 1)
```

```
                do(setf (aref matrix i j) el)
                  (setf el (1+ el))
            )
            (loop for j upfrom (+ i 1) to (- n i 1)
                do(setf (aref matrix j (- n i 1)) el)
                  (setf el (1+ el))
            )
            (loop for j downfrom (- n i 2) to i
                do(setf (aref matrix (- n i 1) j) el)
                  (setf el (1+ el))
            )
            (loop for j downfrom (- n i 2) to (+ i 1)
                do(setf (aref matrix j i) el)
                  (setf el (1+ el))
            )
        )

        matrix
    )
)
```

Результаты работы

```
(print-matrix (spiral-matrix 1))
#2A((1))
(print-matrix (spiral-matrix 2))
#2A((1 2)
    (4 3))
(print-matrix (spiral-matrix 5))
#2A((1 2 3 4 5)
    (16 17 18 19 6)
    (15 24 25 20 7)
    (14 23 22 21 8)
    (13 12 11 10 9))
(print-matrix (spiral-matrix 7))
#2A((1 2 3 4 5 6 7)
    (24 25 26 27 28 29 8)
    (23 40 41 42 43 30 9)
    (22 39 48 49 44 31 10)
    (21 38 47 46 45 32 11)
    (20 37 36 35 34 33 12)
    (19 18 17 16 15 14 13))
```

Дневник отладки

Дата	Событие	Действие по исправлению	Примечание
03.04.19	Излишнее использование глобальных переменных	Заменены на локальные переменные	Также внешний цикл был заменен на dotimes и подправлены счётчики во внутренних циклах
	(spiral-matrix 5) возвращает ноль значений	Исправлено (/ a b) на (ceiling a b)	
	(spiral-matrix 5) возвращает ноль значений	Функция spiral-matrix теперь не печатает матрицу, а возвращает её	

Замечания автора по существу работы

Я считаю, что эта задача не очень сложная с алгоритмической точки зрения, но интересная с точки зрения рассмотрения нового языка.

Выводы

Решая задачи подобного рода, я предпочитаю начинать с рассмотрения частного случая, где выявляется какая-то закономерность. Затем перехожу к выводу формул для общего случая.