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

Студент группы 8О-306 Гаврилов Максим, № по списку 7.

Контакты: cobraj@yandex.ru
Работа выполнена: 25.04.2022
Преподаватель: Иванов Дмитрий Анатольевич, доц. каф. 806
Отчет сдан:
Итоговая оценка:
Подпись преподавателя:

1. Тема работы
Последовательности, массивы и управляющие конструкции Коммон Лисп

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

3. Задание (вариант № 3.46)

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.

Примеры

```
(defun spiral-matrix (n)
  ...)

(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))
```

4. Оборудование студента
процессор AMD FX(tm)-6300 Six-Core Processor 3.50 GHz, память 16ГБ, 64-разрядная система.

5. Программное обеспечение
ОС Windows 10, программа portacle, версия slime 2.24

6. Идея, метод, алгоритм
Итеративный процесс с использованием циклов.

7. Сценарий выполнения работы

1. Изучить функции lisp для работы с массивами и обращения к элементам массивов по индексу.
2. Построить функцию, выполняющую заполнение матрицы по спирали.
3. Провести тестирование работы построенной функции, используя пример из задания.

8. Распечатка программы и её результаты

Программа

```
(defun spiral (sz)
  (let ((arr ( make-array (list sz sz))) (s sz) (gc 1))
    ;; вправо вниз влево
    (dotimes (x s)
      (setf (aref arr 0 x) gc)
      (setf gc (1+ gc)))
    (setf gc (1- gc))
    (dotimes (x s)
      (setf (aref arr x (- s 1)) gc)
      (setf gc (1+ gc)))
    (setf gc (1- gc))
    (dotimes (x s)
      (setf (aref arr (- s 1) (- (- s x) 1)) gc)
      (setf gc (1+ gc)))
    (setf gc (1- gc))

    (dotimes (y (floor s 2))
      ;;вверх вправо
      (setf s (1- s))
      (dotimes (x s)
        (setf (aref arr (+ (- s x) y) y) gc)
        (setf gc (1+ gc)))
      (setf gc (1- gc))
      (dotimes (x s)
        (setf (aref arr (+ y 1) (+ x y)) gc)
        (setf gc (1+ gc)))
      (setf gc (1- gc))

      (if (> (- s 2) 0)
        ;;вниз влево
        (progn (setf s (1- s))
        (dotimes (x s)
          (setf (aref arr (+ (+ x 1) y) (+ s y)) gc)
          (setf gc (1+ gc)))
        (setf gc (1- gc))
        (dotimes (x s)
          (setf (aref arr (+ s y) (+ (- s x) y)) gc)
          (setf gc (1+ gc)))
        (setf gc (1- gc)))))
    arr))
```

Результаты

```
; SLIME 2.24
CL-USER> (defun spiral (sz)
  (let ((arr ( make-array (list sz sz))) (s sz) (gc 1))
    ;; вправо вниз влево
    (dotimes (x s)
      (setf (aref arr 0 x) gc)
      (setf gc (1+ gc)))
    (setf gc (1- gc))
    (dotimes (x s)
      (setf (aref arr x (- s 1)) gc)
      (setf gc (1+ gc)))
    (setf gc (1- gc))
    (dotimes (x s)
      (setf (aref arr (- s 1) (- (- s x) 1)) gc)
      (setf gc (1+ gc)))
    (setf gc (1- gc))

    (dotimes (y (floor s 2))
      ;;вверх вправо
      (setf s (1- s))
      (dotimes (x s)
        (setf (aref arr (+ (- s x) y) y) gc)
        (setf gc (1+ gc)))
      (setf gc (1- gc))
      (dotimes (x s)
        (setf (aref arr (+ y 1) (+ x y)) gc)
        (setf gc (1+ gc)))
      (setf gc (1- gc))

      (if (> (- s 2) 0)
        ;;вниз влево
        (progn (setf s (1- s))
        (dotimes (x s)
          (setf (aref arr (+ (+ x 1) y) (+ s y)) gc)
          (setf gc (1+ gc)))
        (setf gc (1- gc))
        (dotimes (x s)
          (setf (aref arr (+ s y) (+ (- s x) y)) gc)
          (setf gc (1+ gc)))
        (setf gc (1- gc)))))
    arr))

SPIRAL
CL-USER> (spiral 1)
#2A((1))
CL-USER> (spiral 2)
#2A((1 2) (4 3))
CL-USER> (spiral 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))
CL-USER> (spiral 6)
#2A((1 2 3 4 5 6)
    (20 21 22 23 24 7)
    (19 32 33 34 25 8)
    (18 31 36 35 26 9)
    (17 30 29 28 27 10)
    (16 15 14 13 12 11))
CL-USER> (spiral 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))
CL-USER> (spiral 10)
#2A((1 2 3 4 5 6 7 8 9 10)
    (36 37 38 39 40 41 42 43 44 11)
    (35 64 65 66 67 68 69 70 45 12)
    (34 63 84 85 86 87 88 71 46 13)
    (33 62 83 96 97 98 89 72 47 14)
    (32 61 82 95 100 99 90 73 48 15)
    (31 60 81 94 93 92 91 74 49 16)
    (30 59 80 79 78 77 76 75 50 17)
    (29 58 57 56 55 54 53 52 51 18)
    (28 27 26 25 24 23 22 21 20 19))
CL-USER>
```

9. Дневник отладки

№	Дата, время	Событие	Действие по исправлению	Примечание
1				

10. Замечания автора по существу работы

Не придумав ничего более оптимального, я решил сделать программу, заполняющую матрицу по спирали, так же, как это делал бы человек. Процесс спирального обхода я разбил на три основные части – первый проход из элемента в позиции 0,0 вправо до упора, вниз до упора и влево до упора. Затем последовательно выполняются проходы «вверх-вправо» и «вниз-влево», каждый раз длина проходов уменьшается на один, пока не станет равна 1. Тем самым, достигается полный диагональный обход матрицы. Все это время ведется дополнительный счетчик, который начинается с 1 и инкрементируется каждый раз, когда осуществляется запись его значения в ячейку матрицы. Так осуществляется заполнение массива.

11. Выводы

В данной лабораторной работе я получил опыт написания на языке Коммон Лисп функций, не осуществляющих рекурсивный процесс, а, с помощью циклов, обрабатывающих массивы. Меня все еще не отпускает чувство, что я при этом делаю что-то не так, что, впрочем, не исключено, ведь моя реализация до предела наивно следует алгоритму, по которому такую матрицу заполнил бы человек. Вполне вероятно, можно придумать серьезно улучшающие производительность этой функции оптимизации, использующие преимущества языка Коммон Лисп.