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

Студент группы 8О-307 Хренов Геннадий, № по списку 23.

Контакты: khrenov.gena@yandex.ru
Работа выполнена: 31.03.2021
Преподаватель: Дмитрий Анатольевич Иванов, доц. каф. 806
Отчет сдан:
Итоговая оценка:
Подпись преподавателя:

1. Тема работы

Последовательности, массивы и управляющие конструкции Коммон Лисп.

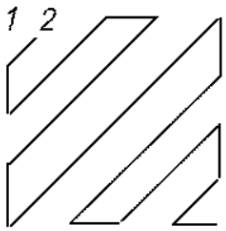
2. Цель работы

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

3. Задание (вариант № 3.44)

Вариант 3.44 (сложность 3)

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 схеме, показанной на рисунке.




```
(defun matrix-t1-t2 (n)
  ...)

(matrix-tl-bl 4) =>
#2A((1   2  6  7)
    (3   5  8 13)
    (4   9 12 14)
    (10 11 15 16))
```

4. Оборудование студента

Ноутбук ASUS TUF GAMING, процессор AMD Ryzen 7 3750H 2.30GHz, память 8ГБ, 64-разрядная система.

5. Программное обеспечение

ОС Windows 10, программа LispWorks Personal Edition 6.1.1

6. Идея, метод, алгоритм

Так как размеры матрицы не фиксированы и задаются пользователем, в данном задании использовать циклы с заданным числом итераций нецелесообразно. Гораздо проще указывать условия окончания. Исходя из логики, задаем 4 условия для обхода матрицы:

1) Если при обходе упираемся сверху и справа есть место, то делаем шаг вправо и спускаемся по диагонали влево до упора.

2) Если упираемся слева и снизу есть место, то делаем шаг вниз и поднимаемся по диагонали вправо до упора.
3) Если упираемся справа и внизу есть место, то делаем шаг вниз и спускаемся по диагонали влево до упора.
4) Если упираемся снизу и справа есть место, то делаем шаг вправо и поднимаемся по диагонали вправо до упора.

Шагая по матрице, не забываем присвоить текущей ячейке значение и увеличить счетчик. Останавливаемся когда заполнили все элементы матрицы.

7. Сценарий выполнения работы

Изучить синтаксис циклов loop и do. Разобраться в методах создания матриц и способах присвоения значений ее элементам. Написать функцию, используя алгоритм п.6.

8. Распечатка программы и её результаты

Программа

```
(defun matrix-t1-t2(n)
  (let ((i 0)                                          ;индексы для передвижения
        (j 0)                                          ;по массиву
        (count 1)
        (a (make-array (list n n) :initial-element 0)))  ;создаем матрицу n*n
                                                       ;с нулевыми элементами
    (setf (aref a i j) count)
    (loop
     (when (= count (* n n)) (return a))             ;условие окончания цикла

     (when (and (= i 0) (< j (- n 1))) ;упираемся сверху и справа есть место
       (incf j)                                      ;тогда делаем шаг вправо и
       (incf count)
       (setf (aref a i j) count)
       (do () ((= j 0))                ;спускаемся влево по диагонали до упора
         (incf i)
         (decf j)
         (incf count)
         (setf (aref a i j) count)))

     (when (and (= j 0) (< i (- n 1)))     ;упираемся слева и снизу есть место
       (incf i)                                      ;тогда делаем шаг вниз и
       (incf count)
       (setf (aref a i j) count)
       (do () ((= i 0))                ;поднимаемся вправо по диагонали до упора
         (decf i)
         (incf j)
         (incf count)
         (setf (aref a i j) count)))

     (when (and (= j (- n 1)) (< i (- n 1))) ;упираемся справа и внизу есть
место
       (incf i)                                      ;тогда делаем шаг вниз и
       (incf count)
       (setf (aref a i j) count)
       (do () ((= i (- n 1)))          ;спускаемся влево по диагонали до упора
         (incf i)
         (decf j)
         (incf count)
         (setf (aref a i j) count)))
```

```
      (when (and (= i (- n 1)) (< j (- n 1))) ;упираемся снизу и справа есть
место
        (incf j)                                    ;тогда делаем шаг вправо и
        (incf count)
        (setf (aref a i j) count)
        (do () ((= j (- n 1)))        ;поднимаемся вправо по диагонали до упора
          (decf i)
          (incf j)
          (incf count)
          (setf (aref a i j) count))))))

(defun print-matrix (matrix &optional (chars 3) stream)  ;печать матрицы
  (let ((*print-right-margin* (+ 6  (* (1+ chars)
                                       (array-dimension matrix 1)))))
    (pprint matrix stream)
    (values)))
```

Результаты

```
CL-USER 1 > (matrix-t1-t2 1)
#2A((1))

CL-USER 2 > (matrix-t1-t2 2)
#2A((1 2) (3 4))

CL-USER 3 > (print-matrix (matrix-t1-t2 3))

#2A((1 2 6)
    (3 5 7)
    (4 8 9))

CL-USER 4 > (print-matrix (matrix-t1-t2 5))

#2A((1 2 6 7 15)
    (3 5 8 14 16)
    (4 9 13 17 22)
    (10 12 18 21 23)
    (11 19 20 24 25))

CL-USER 5 >
```

9. Дневник отладки

№	Дата, время	Событие	Действие по исправлению	Примечание
1				

10. Замечания автора по существу работы

При использовании циклов с заданным числом итераций необходимо было бы высчитывать, сколько шагов делать по диагонали, ведь это число постоянно меняется. Также нужно было бы учитывать размерность матриц (четная или нечетная). С циклами с незаданным числом итераций таких проблем не возникает, и такая реализация проще и интуитивно понятнее.

11. Выводы

В данной лабораторной работе я научился работать с массивами в Коммон Лисп, а также познакомился с основными циклами. Функционал в этой области не уступает другим языкам программирования, которые я знаю. Также стоит обратить внимание на модифицирующие макросы, благодаря которым код становится проще и понятнее.