Лабораторная работа № 3. Управляющие структуры

3.1. Цель работы

Основная цель работы — освоить применение циклов функций и сторонних для Julia пакетов для решения задач линейной алгебры и работы с матрицами.

3.2. Предварительные сведения

3.2.1. Циклы while и for

Для различных операций, связанных с перебором индексируемых элементов структур данных, традиционно используются циклы `while` и `for`.

Синтаксис `while`

```
while <условие>
    <тело цикла>
end
```

Например, `while` можно использовать для формирования элементов массива:

```
# пока n<10 прибавить к n единицу и распечатать значение:
n = 0
while n < 10
    n += 1
    println(n)
end
```

Другой пример демонстрирует использование `while` при работе со строковыми элементами массива, подставляя имя из массива в заданную строку приветствия и выводя получившуюся конструкцию на экран:

```
myfriends = ["Ted", "Robyn", "Barney", "Lily", "Marshall"]

i = 1
while i <= length(myfriends)
    friend = myfriends[i]
    println("Hi $friend, it's great to see you!")
    i += 1
end
```

Такие же результаты можно получить при использовании цикла `for`.

Синтаксис `for`

```
for <переменная> in <диапазон>
    <тело цикла>
end
```

Рассмотренные выше примеры, но с использованием цикла `for`:

```
for n in 1:2:10
    println(n)
end
myfriends = ["Ted", "Robyn", "Barney", "Lily", "Marshall"]

for friend in myfriends
    println("Hi $friend, it's great to see you!")
end
```

Пример использования цикла `for` для создания двумерного массива, в котором значение каждой записи является суммой индексов строки и столбца:

```
# инициализация массива m x n из нулей:
m, n = 5, 5
A = fill(0, (m, n))

# формирование массива, в котором значение каждой записи
# является суммой индексов строки и столбца:
for i in 1:m
  for j in 1:n
    A[i, j] = i + j
  end
end
A
```

Другая реализация этого же примера:

```
# инициализация массива m x n из нулей:
B = fill(0, (m, n))

for i in 1:m, j in 1:n
  B[i, j] = i + j
end
B
```

Ещё одна реализация этого же примера:

```
C = [i + j for i in 1:m, j in 1:n]
C
```

3.2.2. Условные выражения

Довольно часто при решении задач требуется проверить выполнение тех или иных условий. Для этого используют условные выражения.

Синтаксис условных выражений с ключевым словом:

```
if <условие 1>
    <действие 1>
elseif <условие 2>
    <действие 2>
else
    <действие 3>
end
```

Например, пусть для заданного числа N требуется вывести слово «Fizz», если N делится на 3, «Buzz», если N делится на 5, и «FizzBuzz», если N делится на 3 и 5:

```
# используем `&&` для реализации операции "AND"
# операция % вычисляет остаток от деления
if (N % 3 == 0) && (N % 5 == 0)
    println("FizzBuzz")
elseif N % 3 == 0
    println("Fizz")
elseif N % 5 == 0
    println("Buzz")
else
    println(N)
end
```

Синтаксис условных выражений с тернарными операторами:

```
a ? b : c
```
(если выполнено a, то выполнить b, если нет, то c).

Такая запись эквивалентна записи условного выражения с ключевым словом:
```
if a
    b
else
    c
end
```
Пример использования тернарного оператора:
```
x = 5
y = 10

(x > y) ? x : y
```

3.2.3. Функции

Julia дает нам несколько разных способов написать функцию. Первый требует ключевых слов `function` и `end`:
```
function sayhi(name)
    println("Hi $name, it's great to see you!")
end
# функция возведения в квадрат:
function f(x)
    x^2
end
```
Вызов функции осуществляется по её имени с указанием аргументов, например:
```
sayhi("C-3PO")
f(42)
```
В качестве альтернативы, можно объявить любую из выше определённых функций в одной строке:
```
sayhi2(name) = println("Hi $name, it's great to see you!")
f2(x) = x^2
```
Наконец, можно объявить выше определённые функции как «анонимные»:
```
sayhi3 = name -> println("Hi $name, it's great to see you!")
f3 = x -> x^2
```
По соглашению в Julia функции, сопровождаемые восклицательным знаком, изменяют свое содержимое, а функции без восклицательного знака не делают этого.

Например, сравните результат применения `sort` и `sort!`:
```
# задаём массив v:
v = [3, 5, 2]
sort(v)
v
sort!(v)
v
```
Функция `sort(v)` возвращает отсортированный массив, который содержит те же элементы, что и массив `v`, но исходный массив `v` остаётся без изменений. Если же использовать `sort!(v)`, то отсортировано будет содержимое исходного массива `v`.

В программировании под функцией высшего порядка понимается функция, принимающая в качестве аргументов другие функции или возвращающая другую функцию

в качестве результата. Основная идея состоит в том, что функции имеют тот же статус, что и другие объекты данных.

В Julia функция `map` является функцией высшего порядка, которая *принимает функцию* в качестве одного из своих входных аргументов и применяет эту функцию к каждому элементу структуры данных, которая ей передаётся также в качестве аргумента.

Например, в случае выполнения выражения:

```
map(f, [1, 2, 3])
```

на выходе получим массив, в котором функция `f` была применена ко всем элементам массива `[1, 2, 3]`:

```
[f(1), f(2), f(3)]
```

Если $f(x) = x^2$, то получим следующий результат:

```
f(x) = x^2
map(f, [1, 2, 3])
```

```
3-element Array{Int64,1}:
1
4
9
```

То есть в квадрат возведены все элементы массива `[1, 2, 3]`, но не сам массив (вектор).

В `map` можно передать и анонимно заданную, а не именованную функцию:

```
x -> x^3
map(x -> x^3, [1, 2, 3])
```

Результат:

```
3-element Array{Int64,1}:
1
8
27
```

Функция `broadcast` — ещё одна функция высшего порядка в Julia, представляющая собой обобщение функции `map`. Функция `broadcast()` будет пытаться привести все объекты к общему измерению, `map()` будет напрямую применять данную функцию поэлементно.

Синтаксис для вызова `broadcast` такой же, как и для вызова`map`, например применение функции `f` к элементам массива `[1, 2, 3]`:

```
f(x) = x^2
broadcast(f, [1, 2, 3])
```

или

```
f.([1, 2, 3])
```

Пример:

```
# Задаём матрицу A:
A = [i + 3*j for j in 0:2, i in 1:3]
```

Результат

```
3×3 Array{Int64,2}:
1  2  3
4  5  6
7  8  9
```

```
# Вызываем функцию f возведения в квадрат
f(A)
```

Результат ($f(A) = A^2 = A * A$):

```
3×3 Array{Int64,2}:
30   36   42
66   81   96
102  126  150
```

С другой стороны,

```
B = f.(A)
```

Результат (содержит квадраты всех элементов A):

```
3×3 Array{Int64,2}:
1   4   9
16  25  36
49  64  81
```

Точечный синтаксис для `broadcast()` позволяет записать относительно сложные составные поэлементные выражения в форме, близкой к математической записи.

Например, запись

```
A .+ 2 .* f.(A) ./ A
```

или запись

```
@. A + 2 * f(A) / A
```

аналогичны записи

```
broadcast(x -> x + 2 * f(x) / x, A)
```

и дадут идентичный результат:

```
3×3 Array{Float64,2}:
3.0   6.0   9.0
12.0  15.0  18.0
21.0  24.0  27.0
```

3.2.4. Сторонние библиотеки (пакеты) в Julia

Julia имеет более 2000 зарегистрированных пакетов, что делает их огромной частью экосистемы Julia. Есть вызовы функций первого класса для других языков, обеспечивающие интерфейсы сторонних функций. Можно вызвать функции из Python или R, например, с помощью `PyCall` или `Rcall`.

С перечнем доступных в Julia пакетов можно ознакомиться на страницах следующих ресурсов:

- `https://julialang.org/packages/`
- `https://juliahub.com/ui/Home`
- `https://juliaobserver.com/`
- `https://github.com/svaksha/Julia.jl`

При первом использовании пакета в вашей текущей установке Julia вам необходимо использовать менеджер пакетов, чтобы явно его добавить:

```
import Pkg
Pkg.add("Example")
```

При каждом новом использовании Julia (например, в начале нового сеанса в REPL или открытии блокнота в первый раз) нужно загрузить пакет, используя ключевое слово `using`:

Например, добавим и загрузим пакет Colors:

```
Pkg.add("Colors")
using Colors
```

Затем создадим палитру из 100 разных цветов:

```
palette = distinguishable_colors(100)
```

а затем определим матрицу 3×3 с элементами в форме случайного цвета из палитры, используя функцию `rand`:

```
rand(palette, 3, 3)
```

3.3. Задание

1. Используя Jupyter Lab, повторите примеры из раздела 3.2.
2. Выполните задания для самостоятельной работы (раздел 3.4).

3.4. Задания для самостоятельного выполнения

1. Используя циклы `while` и `for`:
   - выведите на экран целые числа от 1 до 100 и напечатайте их квадраты;
   - создайте словарь `squares`, который будет содержать целые числа в качестве ключей и квадраты в качестве их пар-значений;
   - создайте массив `squares_arr`, содержащий квадраты всех чисел от 1 до 100.
2. Напишите условный оператор, который печатает число, если число чётное, и строку «нечётное», если число нечётное. Перепишите код, используя тернарный оператор.
3. Напишите функцию `add_one`, которая добавляет 1 к своему входу.
4. Используйте `map()` или `broadcast()` для задания матрицы A, каждый элемент которой увеличивается на единицу по сравнению с предыдущим.
5. Задайте матрицу A следующего вида:

$$A = \begin{pmatrix} 1 & 1 & 3 \\ 5 & 2 & 6 \\ -2 & -1 & -3 \end{pmatrix}.$$

   - Найдите A^3.
   - Замените третий столбец матрицы A на сумму второго и третьего столбцов.
6. Создайте матрицу B с элементами $B_{i1} = 10$, $B_{i2} = -10$, $B_{i3} = 10$, $i = 1, 2, \ldots, 15$. Вычислите матрицу $C = B^T B$.
7. Создайте матрицу Z размерности 6×6, все элементы которой равны нулю, и матрицу E, все элементы которой равны 1. Используя цикл `while` или `for` и закономерности расположения элементов, создайте следующие матрицы размерности 6×6:

$$Z_1 = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}, \quad Z_2 = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \end{pmatrix},$$

$$Z_3 = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}, \quad Z_4 = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}.$$

8. В языке R есть функция `outer()`. Фактически, это матричное умножение с возможностью изменить применяемую операцию (например, заменить произведение на сложение или возведение в степень).

– Напишите свою функцию, аналогичную функции `outer()` языка R. Функция должна иметь следующий интерфейс: `outer(x,y,operation)`. Таким образом, функция вида `outer(A,B,*)` должна быть эквивалентна произведению матриц A и B размерностями $L \times M$ и $M \times N$ соответственно, где элементы результирующей матрицы C имеют вид $C_{ij} = \sum_{k=1}^{M} A_{ik}B_{kj}$ (или в тензорном виде $C_i^j = \sum_{k=1}^{M} A_k^i B_j^k$).
– Используя написанную вами функцию `outer()`, создайте матрицы следующей структуры:

$$A_1 = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 & 5 \\ 2 & 3 & 4 & 5 & 6 \\ 3 & 4 & 5 & 6 & 7 \\ 4 & 5 & 6 & 7 & 8 \end{pmatrix}, \quad A_2 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 2 & 4 & 8 & 16 & 32 \\ 3 & 9 & 27 & 81 & 243 \\ 4 & 16 & 64 & 256 & 1024 \end{pmatrix}.$$

$$A_3 = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 & 0 \\ 2 & 3 & 4 & 0 & 1 \\ 3 & 4 & 0 & 1 & 2 \\ 4 & 0 & 1 & 2 & 3 \end{pmatrix}, \quad A_4 = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 8 & 9 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 9 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{pmatrix},$$

$$A_5 = \begin{pmatrix} 0 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 \\ 1 & 0 & 8 & 7 & 6 & 5 & 4 & 3 & 2 \\ 2 & 1 & 0 & 8 & 7 & 6 & 5 & 4 & 3 \\ 3 & 2 & 1 & 0 & 8 & 7 & 6 & 5 & 4 \\ 4 & 3 & 2 & 1 & 0 & 8 & 7 & 6 & 5 \\ 5 & 4 & 3 & 2 & 1 & 0 & 8 & 7 & 6 \\ 6 & 5 & 4 & 3 & 2 & 1 & 0 & 8 & 7 \\ 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 & 8 \\ 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \end{pmatrix}.$$

В каждом случае ваше решение должно быть легко обобщаемым на случай создания матриц большей размерности, но той же структуры.

9. Решите следующую систему линейных уравнений с 5 неизвестными:

$$\begin{cases} x_1 + 2x_2 + 3x_3 + 4x_4 + 5x_5 = 7, \\ 2x_1 + x_2 + 2x_3 + 3x_4 + 4x_5 = -1, \\ 3x_1 + 2x_2 + x_3 + 2x_4 + 3x_5 = -3, \\ 4x_1 + 3x_2 + 2x_3 + x_4 + 2x_5 = 5, \\ 5x_1 + 4x_2 + 3x_3 + 2x_4 + x_5 = 17, \end{cases}$$

рассмотрев соответствующее матричное уравнение Ax $= y$. Обратите внимание на особый вид матрицы A. Метод, используемый для решения данной системы уравнений, должен быть легко обобщаем на случай большего числа уравнений, где матрица A будет иметь такую же структуру.

10. Создайте матрицу M размерности 6×10, элементами которой являются целые числа, выбранные случайным образом с повторениями из совокупности $1, 2, \ldots, 10$.
    - Найдите число элементов в каждой строке матрицы M, которые больше числа N (например, $N = 4$).
    - Определите, в каких строках матрицы M число M (например, $M = 7$) встречается ровно 2 раза?
    - Определите все пары столбцов матрицы M, сумма элементов которых больше K (например, $K = 75$).
11. Вычислите:
    - $\sum_{i=1}^{20} \sum_{j=1}^{5} \frac{i^4}{(3+j)}$,
    - $\sum_{i=1}^{20} \sum_{j=1}^{5} \frac{i^4}{(3+ij)}$.

3.5. Содержание отчёта

1. Титульный лист с указанием номера лабораторной работы и ФИО студента.
2. Формулировка задания работы.
3. Описание выполнения задания:
   - подробное пояснение выполняемых в соответствии с заданием действий;
   - скриншоты (снимки экрана), фиксирующие выполнение лабораторной работы;
   - листинги (исходный код) программ и результаты его выполнения;
4. Выводы, согласованные с заданием работы.