Циклы

Для организации повторяющихся вычислений в **Julia** есть две конструкции: цикл `for` и цикл `while`.

Цикл for

Цикл `for` имеет следующую структуру:

```
for i = m:n
    действия
end
```

Здесь `i` – переменная-счетчик, которая принимает значения из диапазона, заданного справа от знака равенства; `m:n` – объект-диапазон, представляющий собой последовательность чисел от `m` до `n` с шагом 1. Цикл `for` перебирает эти значения, по очереди присваивая их переменной `i`.

Пример. С помощью цикла `for` выведем на экран все натуральные числа от 1 до 10.

[ ]:
```
for i = 1:10
    println(i)
end
```

✏Задание 1

С помощью цикла `for` вычислите сумму всех натуральных чисел от 1 до 100.

[ ]:
```

```

Подсказка Перед началом цикла задайте начальное значение суммы `s` равным нулю. Затем откройте цикл по переменной `i`, которая будет пробегать все натуральные значения от 1 до 100. В теле цикла запишите выражение для вычисления суммы: `s = s + i`. После окончания цикла выведите на экран значение суммы с помощью функции `println`.

Решение

[ ]:
```
s = 0
for i = 1:100
    s = s + i
end
println(s)
```

Внутри тела цикла `for` всегда вводится новая переменная-счетчик независимо от того, есть ли переменная с таким именем во внешней области. Из этого сле-

дует, что, с одной стороны, переменную `i` не нужно объявлять перед циклом. С другой стороны, она будет недоступна вне цикла и не будет влиять на внешнюю переменную с тем же именем.

Внутри цикла по одной переменной может находиться второй цикл по другой переменной. Такой цикл называется **двойным**. С помощью двойного цикла удобно производить действия с элементами двумерных массивов (матриц). Количество вложенных друг в друга циклов может быть любым.

✏Задание 2

Задайте матрицу

$$A = \begin{pmatrix} 2 & 8 & -3 \\ -4 & 9 & 8 \\ 6 & -5 & 1 \end{pmatrix}.$$

С помощью двойного цикла по индексам `i` и `j` вычислите произведение всех элементов этой матрицы.

[ ]:

Подсказка Перед началом цикла задайте начальное знчение произведения `P` равным 1. Первый цикл по переменной `i` будет перебирать индексы строк матрицы (от 1 до 3), а второй цикл по переменной `j` – индексы столбцов (также от 1 до 3). В теле цикла запишите выражение для вычисления произведения: `P = P * A[i,j]`. После окончания цикла выведите на экран значение `P` с помощью функции `println`.

Решение

[ ]:
```
A = [2 8 -3; -4 9 8; 6 -5 1]
P = 1
for i = 1:3
    for j = 1:3
        P = P * A[i,j]
    end
end
println(P)
```

В общем случае в цикле `for` возможна итерация по любому контейнеру. В таких случаях в качестве равносильной альтернативы символу `=` применяется ключевое слово `in` или `∈`, что делает код более понятным:

[ ]:
```
for i in [1, 4, 7, 2, 0]
    println(i)
```

```
end
```

[ ]:
```
for s ∈ ["red", "green", "blue"]
    println(s)
end
```

Несколько вложенных циклов `for` можно объединить в один внешний цикл, получив декартово произведение итерируемых объектов:

[ ]:
```
for i = 1:3, j = 4:6
    println((i, j))
end
```

При использовании такого синтаксиса в итерируемых объектах по-прежнему можно ссылаться на переменные внешних циклов. Например, выражение `for i = 1:n, j = 1:i` будет допустимым.

В одном цикле `for` можно выполнять итерацию одновременно по нескольким контейнерам с помощью функции `zip`:

[ ]:
```
for (j, k) in zip([1 2 3], [4 5 6 7])
    println((j,k))
end
```

С помощью функции `zip` создается итератор, представляющий собой кортеж из элементов переданных контейнеров. Итератор `zip` по порядку перебирает вложенные итераторы, выбирая i-й элемент каждого из них на i-й итерации цикла `for`. Когда элементы в каком-либо вложенном итераторе заканчиваются, выполнение цикла `for` останавливается.

Цикл while

Цикл `while` имеет следующую структуру:

```
while условие
    действия
end
```

Здесь `условие` – некоторое логическое выражение. Пока оно принимает значение `true`, раз за разом выполняются действия в теле цикла `while`. Как только `условие` станет равным `false`, произойдет выход из цикла. Если на первой же итерации цикла `условие` равно `false`, то действия в теле цикла `while` не выполняются ни разу.

Пример. С помощью цикла `while` выведем на экран все натуральные числа от 1 до 10.

[ ]:
```
i = 1
while i <= 10
    println(i)
```

```
    i = i + 1
end
```

В теле цикла `while` переменная `i` должна изменять свое значение таким образом, чтобы на какой-нибудь итерации условие выполнения цикла `i <= 10` приняло значение `false`. В противном случае цикл будет выполняться бесконечно долго.

✏Задание 3

С помощью цикла `while` вычислите сумму всех натуральных чисел от 25 до 75.

[ ]:
```

```

Подсказка Перед началом цикла задайте начальное значение индекса `i` равным 25, а начальное значение суммы `s` равным нулю. Затем откройте цикл `while`. Поскольку вам нужно перебрать все натуральные значения `i` от 25 до 75, то цикл должен выполняться, пока `i` не превышает 75. Т.е. условие выполнения цикла запишется в виде: `i <= 75`. В теле цикла запишите выражение для вычисления суммы `s = s + i`, а затем увеличьте значение индекса `i` на единицу. После окончания цикла выведите на экран значение суммы с помощью функции `println`.

Решение

[ ]:
```
i = 25
s = 0
while i <= 75
    s = s + i
    i = i + 1
end
println(s)
```

Ключевое слово break

Иногда бывает необходимо завершить выполнение цикла `while` до того, как условие примет значение `false`, или прервать выполнение цикла `for` до того, как будет достигнут конец итерируемого объекта. Для этого можно использовать ключевое слово `break`, которое передает управление оператору, расположенному сразу после тела цикла.

Пример. Следующий код выводит на экран сумму всех натуральных чисел от 1 до 10.

[ ]:
```
i = 0
s = 0
while true
    i = i + 1
    s = s + i
```

```
    if i >= 10
        break
    end
end
println(s)
```

Как только счетчик `i` достигнет значения 10, выполнение цикла будет прервано благодаря ключевому слову `break`.

Без ключевого слова `break` выполнение приведенного выше цикла `while` никогда бы не завершилось само по себе, т.к. условие, стоящее в заголовке цикла, всегда равно `true`.

✏Задание 4

Составьте функцию `ifzero`, которая принимает на вход одномерный числовой массив `x` и с помощью цикла `for` проверяет, содержит ли массив хотя бы один нулевой элемент или нет. Функция должна возвращать текстовую строку “Массив содержит нулевой элемент” или “Массив не содержит нулевых элементов”. Затем вызовите функцию `ifzero` с входными массивами `[1 2 0 4]` и `[3 7 8 5]`.

[ ]:
```

```

Подсказка Сначала в теле функции `fzero(x)` задайте значение строковой переменной `s` равным “Массив не содержит нулевых элементов”. Затем откройте цикл `for` по индексу `i`, пробегающему значения от 1 до числа элементов в массиве `x` (которое можно найти с помощью функции `length(x)`). В теле цикла проверьте условие: если `i`-й элемент массива `x` равен нулю, то нужно выполнить блок, состоящий из двух действий. Во-первых, значение переменной `s` измените на “Массив содержит нулевой элемент”. Т.к. один нулевой элемент в массиве `x` уже найден, то проверять все остальные элементы не нужно. Поэтому вторым действием внутри блока `if` будет преждевременный выход из цикла с помощью ключевого слова `break`. После окончания цикла задайте переменную `s` в качестве возвращаемого значения функции (`return s`).

Решение

[ ]:
```
function ifzero(x)
    s = "Массив не содержит нулевых элементов"
    for i = 1:length(x)
        if x[i] == 0
            s = "Массив содержит нулевой элемент"
            break
        end
    end
    return s
end
```

[ ]:
```
ifzero([1 2 0 4])
```

[ ]:
```
ifzero([3 7 8 5])
```

Ключевое слово continue

В других случаях бывает полезно прервать итерацию и сразу перейти к следующей. Для этого служит ключевое слово `continue`.

Пример. Следующий код выводит на экран все числа, кратные 7 и принадлежащие отрезку $[1, 100]$.

[ ]:
```
for i = 1:100
    if mod(i,7) != 0
        continue
    else
        println(i)
    end
end
```

В теле цикла проверяется условие: если переменная `i` не кратна 7 (т.е. если остаток от деления `i` на 7 не равен нулю), то мы переходим к следующей итерации цикла. В противном случае выводится на экран значение `i`.

Эту задачу можно было бы решить без использования `continue`, поменяв условие на противоположное и поместив вызов `println` внутрь блока `if`:

[ ]:
```
for i = 1:100
    if mod(i,7) == 0
        println(i)
    end
end
```

На практике после ключевого слова `continue` следуют более сложные вычисления, а точек вызова `continue` обычно несколько.

✏Задание 5

С помощью цикла `for` вычислите величины, обратные ко всем отличным от нуля элементам одномерного массива `x = [4 5 0 10 2 0 8 0 1 16]`. Чтобы предотвратить деление на ноль, используйте ключевое слово `continue`.

[ ]:
```

```

Подсказка Сначала откройте цикл `for` по индексу `i`, пробегающему значения от 1 до числа элементов в массиве `x` (которое можно найти с помощью функции `length(x)`). В теле цикла проверьте условие: если `i`-й элемент массива `x` равен нулю, то вычислять величину, обратную к этому элементу, не нужно, поэтому нужно перейти к следующей итерации цикла с помощью ключевого слова

continue. В теле цикла после блока if вычислите величину, обратную к i-му элементу массива x, и выведите ее на экран: println(1/x[i]).

Решение

[ ]:
```
x = [4 5 0 10 2 0 8 0 1 16]
for i = 1:length(x)
    if x[i] == 0
        continue
    end
    println(1/x[i])
end
```

Замечание. Эту задачу можно было бы решить и без использования continue, поменяв условие на противоположное и поместив вычисление обратной величины внутрь блока if:

[ ]:
```
x = [4 5 0 10 2 0 8 0 1 16]
for i = 1:length(x)
    if x[i] != 0
        println(1/x[i])
    end
end
```

Тест для получения сертификата

Пройти тест по теме **Циклы**.