Условные операторы

Операторы if и else

Условные операторы позволяют вычислять или не вычислять части кода в зависимости от значения логического выражения. Синтаксис условной конструкции `if` (если) – `else` (иначе) устроен следующим образом.

```
if логическое_выражение
    действия_1
else
    действия_2
end
```

Если `логическое_выражение` принимает значение `true` (истина), то выполняется последовательность действий `действия_1`. Если `логическое_выражение` принимает значение `false` (ложь), то выполняется последовательность действий `действия_2`. Завершается условная конструкция обязательным ключевым слово `end`.

Например, сравним два числа `x` и `y`. Если они равны, то на экран выводится сообщение: “`x` равно `y`”. В противном случае (если `x` не равен `y`) выводится сообщение: “`x` не равно `y`”. Фрагмент кода, выполняющий эти действия, представлен ниже:

```
if x == y
    println("x равно y")
else
    println("x не равно y")
end
```

Если выражение условия `x == y` имеет значение `true`, то соответствующий блок вычисляется; в противном случае вычисляется блок `else`. Вот как это работает на практике. Создадим функцию `compare(x,y)`, которая сравнивает два аргумента `x` и `y`:

[ ]:
```
function compare(x, y)
    if x == y
        println("x равно y")
    else
        println("x не равно y")
    end
end
```

Вызовем эту функцию с аргументами `(1,2)` и `(3,3)`:

[ ]:
```
compare(1,2)
```

[ ]:
```
compare(3,3)
```

✏Задание 1

Создайте функцию even_odd, которая принимает на вход целое число n и возвращает текстовое сообщение о том, четное это число или нечетное. Затем вызовите эту функцию со значениями аргумента, равными 4, 5 -4 и -5.

[ ]:

Подсказка Напомним, что четность числа можно проверить следующим образом: если остаток от деления числа на 2 равен нулю, то число четное; в противном случае – нечетное. Остаток от деления числа n на 2 находится так: `mod(n,2)`. В заголовке функции не забудьте задать тип входных данных (`Int`): `function even_odd(n::Int)`.

Решение

[ ]:
```
function even_odd(n::Int)
    if mod(n,2) == 0
        println(n, " четное")
    else
       println(n, " нечетное")
    end
end
```

[ ]:
```
even_odd(4)
```

[ ]:
```
even_odd(5)
```

[ ]:
```
even_odd(-4)
```

[ ]:
```
even_odd(-5)
```

Оператор elseif

Еще один оператор условной конструкции `elseif` может быть использован после `if` для описания дополнительных условий для выполнения следующих блоков кода.

```
if логическое_выражение_1
    действия_1
elseif логическое_выражение_2
    действия_2
else
    действия_3
end
```

Модифицируем функцию `compare` из предыдущего примера. Сравним два числа x и y. Если они равны, то на экран выводится сообщение: “x равно y”. В противном случае, если `x > y`, то выводится сообщение: “x больше y”. И, наконец, если

последнее условие не выполняется (т.е. если x < y), то выводится сообщение: “x меньше y”.

[ ]:
```
function compare(x, y)
    if x == y
        println("x равно y")
    elseif x > y
        println("x больше y")
    else
        println("x меньше y")
    end
end
```

Проверим работу этой функции:

[ ]:
```
compare(4,4)
```

[ ]:
```
compare(6,5)
```

[ ]:
```
compare(7,8)
```

Блоки `elseif` и `else` являются необязательными. В одной условной конструкции может быть сколько угодно блоков `elseif`. Выражения условия в конструкции `if` – `elseif`– `else` вычисляются до тех пор, пока для одного из них не будет получено значение `true`, после чего вычисляется соответствующий блок. Последующие выражения условия или блоки не вычисляются.

✏Задание 2

Создайте функцию `translate` с одним входным аргументом `s` текстового типа, который принимает одно из следующих значений: one, two, three, four или five. Функция возвращает перевод введенного слова на русский язык. Если аргумент принимает какое-либо значение, отличное от заданных выше, то функция возвращает текстовое сообщение: “Не могу перевести”. Затем проверьте работу этой функции, вызвав функцию со значениями аргумента, равными one, three и six.

[ ]:
```

```

Подсказка В заголовке функции задайте тип входных данных (`String`): `function translate(s::String)`. Внутри функции вам необходимо будет проверить пять условий (равенство переменной `s` каждому из пяти слов, подлежащих переводу). Поэтому вам потребуется одно ключевое слово `if` и четыре ключевых слова `elseif`. Кроме того, последнее ключевое слово `else` внутри условной конструкции будет обрабатывать ситуацию, когда переменная `s` не принимает ни одного из пяти заданных значений.

Решение

[ ]:
```
function translate(s::String)
    if s == "one"
        println("один")
    elseif s == "two"
        println("два")
    elseif s == "three"
        println("три")
    elseif s == "four"
        println("четыре")
    elseif s == "five"
        println("пять")
    else
        println("Не могу перевести")
    end
end
```

[ ]:
```
translate("one")
```

[ ]:
```
translate("three")
```

[ ]:
```
translate("six")
```

Условные конструкции являются “открытыми”, то есть не образуют локальную область. Это означает, что новые переменные, определенные внутри условной конструкции, можно использовать и после нее, даже если они не были определены ранее. Поэтому приведенную выше функцию `compare` можно было бы определить так:

[ ]:
```
function compare(x,y)
    if x == y
        relation = "равно"
    elseif x > y
        relation = "больше"
    else
        relation = "меньше"
    end
    println("x ", relation, " y")
end
```

Переменная `relation` объявлена внутри условной конструкции, но используется вне ее. Однако при использовании такой возможности значение переменной должно быть определено для каждого возможного пути выполнения кода.

Тернарный оператор ?:

Так называемый **тернарный оператор** `?:` тесно связан с синтаксисом `if` – `elseif` – `else`, но применяется тогда, когда в зависимости от условия требуется выбрать значение одного из выражений, а не выполнить блок кода. Называется он так потому, что это единственный оператор, принимающий три операнда:

`a ? b : c`

Выражение a перед `?` – это выражение условия. Если условие a равно `true`, то вычисляется выражение b перед `:`, а если оно равно `false`, то вычисляется выражение c после `:`. Обратите внимание, что пробелы вокруг `?` и `:` обязательны: выражение `a?b:c` не является допустимым тернарным выражением.

Пример.

[ ]:
```
x = 1
y = 2
println(x == y ? "x равно y" : "x не равно y")
```

Если выражение `x == y` равно `true`, то результатом всего тернарного выражения является строка “x равно y”; в противном случае результатом будет строка “x не равно y”.

Чтобы воспроизвести ситуацию из исходного примера с выбором из трех вариантов, потребуется построить цепочку из тернарных операторов:

[ ]:
```
compare(x, y) = println(x == y ? "x равно y"    :
                            x > y ? "x больше y" : "x меньше y")
```

[ ]:
```
compare(1,1)
```

[ ]:
```
compare(2,3)
```

[ ]:
```
compare(5,4)
```

[ ]:
```
compare(9,10)
```

✏Задание 3

С помощью функции `rand()` сгенерируйте случайное число, равномерно распределенное в интервале от 0 до 1, и присвойте его переменной `x`. С помощью тернарного оператора `?:` выполните следующие действия: если `x < 0.5`, то переменная y должна принимать значение 100, в противном случае переменная y должна принимать значение 200. Запустите код несколько раз, убедившись в случайности возвращаемых результатов.

[ ]:
```

```

Подсказка В тернарном операторе сначала записывается проверяемое условие. В нашем случае это `x < 0.5`. Затем, после символа `?`, записывается действие, которое выполяется, если условие равно `true`. В нашем случае это `y = 100`. Затем, после символа `:` записывается действие, которое выполняется, если условие равно `false`. В нашем случае это `y = 200`.

Решение

[ ]:
```
x = rand()
x < 0.5 ? y = 100 : y = 200
```

Практические примеры использования условных операторов

В качестве примера рассмотрим **функцию для вычисления факториала**, использующую условный оператор. На вход функции подается целое число n. Далее проверяются два условия. Если $n < 0$, то факториал отрицательного числа не может быть вычислен и на экран выводится сообщение об ошибке. Если $n = 0$, то $n! = 1$. В противном случае факториал вычисялется по формуле $n! = n \cdot (n-1)$.

[ ]:
```
function fact(n::Int)
    if n < 0
        return "n должно быть неотрицательным"
    elseif n == 0
        return n = 1
    else
        return n*fact(n-1)
    end
end
```

Проверим работу этой функции, задавая ее аргумент равным -1, 0 и 5:

[ ]:
```
fact(-1)
```

[ ]:
```
fact(0)
```

[ ]:
```
fact(5)
```

✏Задание 4

Создайте функцию `right`, которая принимает на вход длины сторон треугольника `a`, `b` и `c` и возвращает текстовое сообщение о том, является ли этот треугольник прямоугольным. Затем проверьте работу этой функции, вызывая ее со значениями аргументов, равными (2,6,7) и (5,3,4).

[ ]:
```

```

Подсказка Для прямоугольного треугольника выполняется теорема Пифагора: $a^2 + b^2 = c^2$. Мы не знаем, какая из сторон является гипотенузой, а какие –

катетами, поэтому проверить выполнение этого равенства можно, поочередно принимая в качестве гипотенузы каждую из трех сторон треугольника. Если хотя бы в одном случае равенство выполняется, то треугольник прямоугольный. Все три условия удобнее всего объединить в одно с помощью логической операции ИЛИ. Тогда условие будет выглядеть так:

```
if a^2+b^2 == c^2 || b^2+c^2 == a^2 || a^2+c^2 == b^2
```

Если это условие не выполняется, то треугольник не прямоугольный. Следовательно, в условной конструкции вам понадобятся только операторы `if` и `else`.

Решение

[ ]:
```
function right(a,b,c)
    if a^2+b^2 == c^2 || b^2+c^2 == a^2 || a^2+c^2 == b^2
        println("треугольник прямоугольный")
    else
        println("треугольник не прямоугольный")
    end
end
```

[ ]:
```
right(2,6,7)
```

[ ]:
```
right(5,3,4)
```

Тест для получения сертификата

Пройти тест по теме **Условные операторы**.