

# **Лабораторная работа № 1**

**Дисциплина: Информационная безопасность**

Покрас Илья Михайлович

# Содержание

1	Постановка задачи:	4
2	Выполнение работы	5
3	Код Программы	10
4	Выводы:	13

## Список иллюстраций

2.1	Код выполнения пункта 1 с результатом вывода . . . . .	6
2.2	Код выполнения пункта 2 с результатом вывода . . . . .	7
2.3	Код выполнения пункта 3 с результатом вывода . . . . .	8
2.4	Код выполнения пункта 4 с результатом вывода . . . . .	9

# 1 Постановка задачи:

Целью данной работы является подготовка рабочего пространства и инструментария для работы с языком программирования Julia, ознакомления с основами синтаксиса Julia.

## 2 Выполнение работы

1. Я изучил документацию по основным функциям Julia для чтения / записи / вывода информации на экран: `read()`, `readline()`, `readlines()`, `readdlm()`, `print()`, `println()`, `show()`, `write()`. И написал следующий код(2.1).

```

using DelimitedFiles

println(read("example.txt", String))
foo = readline("example.txt")
println("Var's(readline) type - ", typeof(foo))
println("Var(readline) - ", foo)
foo = readlines("example.txt")
println("Var's(readlines) type - ", typeof(foo))
println("Var(readlines) - ", foo)
println("Matrix via readlm: \n", readlm("example.txt", String))
println("2nd file - ", readlines("example2.txt"))
foo = "test3"
write("example2.txt", foo)
println("edited 2nd file - ", readlines("example2.txt"))
y = 3
x = -2

struct MyData
    x::Int
    y::Int
end

function Base.show(io::IO, data::MyData)
    println(io, "x = $(data.x)")
    println(io, "y = $(data.y)")
end

data = MyData(x, y)
println(data)

test1 test1
test2 test2
Var's(readline) type - String
Var(readline) - test1 test1
Var's(readlines) type - Vector{String}
Var(readlines) - ["test1 test1", "test2 test2"]
Matrix via readlm:
["test1" "test1"; "test2" "test2"]
2nd file - ["test3"]
edited 2nd file - ["test3"]
x = -2
y = 3

```

Рис. 2.1: Код выполнения пункта 1 с результатом вывода

Readline() читает одну строку текста из файла или стандартного ввода и возвра-

щает ее в виде строки. Если файл содержит несколько строк, при каждом вызове функции будет прочитана следующая строка.

`Readlines()` читает все строки из файла или стандартного ввода и возвращает их в виде массива строк. Каждая строка файла представлена отдельным элементом массива.

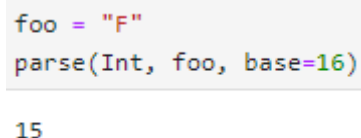
`Read()` читает содержимое файла или стандартного ввода в виде одной большой строки и возвращает ее. Она объединяет все строки файла в одну строку.

`Println` и `print` являются функциями в языке программирования Julia, которые используются для вывода информации в консоль. `Println` выводит переданную ей информацию и переводит курсор на новую строку. Функция `print` также выводит информацию, но не добавляет символ новой строки после вывода.

`Write()` используется для записи данных в файл или другой поток вывода.

`Show()` используется для вывода объектов на экран и предназначена для представления объектов в пользовательском формате, и она автоматически вызывается при вызове функций вывода, таких как `println()`, `print()`.

2. Я изучил документацию по функции `parse()` и написал код(2.2).



```
foo = "F"
parse{Int, String}(foo, base=16)
```

15

Рис. 2.2: Код выполнения пункта 2 с результатом вывода

`Parse()` преобразует строку как число. Если тип является целочисленным, то можно указать систему счисления. Если тип переменная имеет тип `Float`, строка анализируется как десятичное число с плавающей запятой.

3. Я изучил синтаксис Julia для базовых математических операций с разным типом переменных: сложение, вычитание, умножение, деление, возведение в степень, извлечение корня, сравнение, логические операции. И написал код с их применения(2.3).

```

x = +x
println("id op: ", x)
x=-x
println("add inv map: ", x)
println("add: ", x+y)
println("sub: ", x-y)
println("mul: ", x*y)
println("div: ", x/y)
println("int div: ", x÷y)
println("inv div: ", x\y)
println("pow: ", x^y)
println("rem: ", x%y)
println("bitwise NOT - ", ~x)
println("bitwise AND - ", x & y)
println("bitwise OR - ", x | y)
println("bitwise XOR - ", x ⊕ y)
println("bitwise NAND - ", x ⊞ y)
println("bitwise NOR - ", x ∇ y)

a = true
b = false
println("logical AND - ", a && b)
println("logical OR - ", a || b)
println("logical NOT - ", ! a)

```

```

id op: -2
add inv map: 2
add: 5
sub: -1
mul: 6
div: 0.6666666666666666
int div: 0
inv div: 1.5
pow: 8
rem: 2
bitwise NOT - -3
bitwise AND - 2
bitwise OR - 3
bitwise XOR - 1
bitwise NAND - -3
bitwise NOR - -4
logical AND - false
logical OR - true
logical NOT - false

```

Рис. 2.3: Код выполнения пункта 3 с результатом вывода



4. Я написал код нескольких примеров с операциями над матрицами и векторами: сложение, вычитание, скалярное произведение, транспонирование, умножение на скаляр(2.4).

```
A = [1 2; 4 5]
B = [7 8; 10 11]
C = [10 ; 11]
println("matrix add: ", A + B)
println("matrix sub: ", A - B)
println("transposed matrix: ", transpose(A))
println("matrix mul: ", A * 2)
println("matrix mul: ", A * C)

matrix add: [8 10; 14 16]
matrix sub: [-6 -6; -6 -6]
transposed matrix: [1 4; 2 5]
matrix mul: [2 4; 8 10]
matrix mul: [32, 95]
```

Рис. 2.4: Код выполнения пункта 4 с результатом вывода

### **3 Код Программы**

In [1]: `using DelimitedFiles`

```
println(read("example.txt", String))
foo = readline("example.txt")
println("Var's(readline) type - ", typeof(foo))
println("Var(readline) - ", foo)
foo = readlines("example.txt")
println("Var's(readlines) type - ", typeof(foo))
println("Var(readlines) - ", foo)
println("Matrix via readdlm: \n", readdlm("example.txt", String))
println("2nd file - ", readlines("example2.txt"))
foo = "test3"
write("example2.txt", foo)
println("edited 2nd file - ", readlines("example2.txt"))
y = 3
x = -2

struct MyData
    x::Int
    y::Int
end

function Base.show(io::IO, data::MyData)
    println(io, "x = $(data.x)")
    println(io, "y = $(data.y)")
end

data = MyData(x, y)
println(data)
```

```
test1 test1
test2 test2
Var's(readline) type - String
Var(readline) - test1 test1
Var's(readlines) type - Vector{String}
Var(readlines) - ["test1 test1", "test2 test2"]
Matrix via readdlm:
["test1" "test1"; "test2" "test2"]
2nd file - ["test3"]
edited 2nd file - ["test3"]
x = -2
y = 3
```

In [2]: `foo = "F"`  
`parse{Int, foo, base=16}`

Out[2]: 15

In [3]: `x = +x`  
`println("id op: ", x)`  
`x=-x`  
`println("add inv map: ", x)`  
`println("add: ", x+y)`  
`println("sub: ", x-y)`  
`println("mul: ", x*y)`  
`println("div: ", x/y)`  
`println("int div: ", x÷y)`

```

println("inv div: ", x\y)
println("pow: ", x^y)
println("rem: ", x%y)
println("bitwise NOT - ", ~x)
println("bitwise AND - ", x & y)
println("bitwise OR - ", x | y)
println("bitwise XOR - ", x ⊕ y)
println("bitwise NAND - ", x ⊞ y)
println("bitwise NOR - ", x ⊃ y)

a = true
b = false
println("logical AND - ", a && b)
println("logical OR - ", a || b)
println("logical NOT - ", ! a)

```

```

id op: -2
add inv map: 2
add: 5
sub: -1
mul: 6
div: 0.6666666666666666
int div: 0
inv div: 1.5
pow: 8
rem: 2
bitwise NOT - -3
bitwise AND - 2
bitwise OR - 3
bitwise XOR - 1
bitwise NAND - -3
bitwise NOR - -4
logical AND - false
logical OR - true
logical NOT - false

```

```

In [16]: A = [1 2; 4 5]
          B = [7 8; 10 11]
          C = [10 ; 11]
          println("matrix add: ", A + B)
          println("matrix sub: ", A - B)
          println("transposed matrix: ", transpose(A))
          println("matrix mul: ", A * 2)
          println("matrix mul: ", A * C)

```

```

matrix add: [8 10; 14 16]
matrix sub: [-6 -6; -6 -6]
transposed matrix: [1 4; 2 5]
matrix mul: [2 4; 8 10]
matrix mul: [32, 95]

```

In [ ]:

## 4 Выводы:

Я подготовил рабочее пространство и инструментария для работы с языком программирования Julia, ознакомления с основами синтаксиса Julia.