

# **Лабораторная работа №1**

**Julia. Установка и настройка. Основные принципы.**

Доберштейн Алина Сергеевна

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Задание</b>	<b>6</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>7</b>
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>8</b>
<b>5</b>	<b>Простейшие операции на языке Julia</b>	<b>9</b>
5.1	Основные функции Julia . . . . .	11
5.2	Функция parse() . . . . .	17
5.3	Базовые математические операции . . . . .	18
5.4	Операции над матрицами и векторами . . . . .	23
<b>6</b>	<b>Выводы</b>	<b>27</b>
	<b>Список литературы</b>	<b>28</b>

# Список иллюстраций

4.1	Julia	8
5.1	Простейшие арифметические операции	9
5.2	Пример получения информации по функции	9
5.3	Примеры определения типа числовых величин	10
5.4	Примеры приведения аргументов к одному типу	10
5.5	Примеры определения функций	10
5.6	Примеры работы с массивами	11
5.7	Примеры работы с массивами	11
5.8	Функция read	12
5.9	Примеры с функцией read	13
5.10	Функция readline	14
5.11	Примеры с функцией readline	14
5.12	Функция readlines	14
5.13	Функция readlm	15
5.14	Примеры с функцией readlm	15
5.15	Примеры с функциями print и println	16
5.16	Функция show	16
5.17	Примеры с функцией show	16
5.18	Функция write	17
5.19	Примеры с функцией write	17
5.20	Примеры с функцией write	17
5.21	Функция parse	18
5.22	Использование функции parse	18
5.23	Примеры для сложения	19
5.24	Примеры для сложения	19
5.25	Примеры для вычитания	19
5.26	Примеры для умножения	20
5.27	Примеры для деления	20
5.28	Примеры для возведения в степень	21
5.29	Примеры для извлечения корня	21
5.30	Примеры для сравнения	22
5.31	Примеры для логических операций	22
5.32	Определение матриц и векторов	23
5.33	Примеры для сложения векторов и матриц	23
5.34	Примеры для вычитания векторов и матриц	24
5.35	Примеры для скалярного произведения	24

5.36	Примеры для сложения транспонирования . . . . .	25
5.37	Примеры для умножения на скаляр . . . . .	25
5.38	Примеры для скалярного произведения . . . . .	26

# 1 Цель работы

Основная цель работы — подготовить рабочее пространство и инструментарий для работы с языком программирования Julia, на простейших примерах познакомиться с основами синтаксиса Julia.

## 2 Задание

1. Установите под свою операционную систему Julia, Jupyter.
2. Используя Jupyter Lab, повторите примеры из раздела лабораторной работы.
3. Выполните задания для самостоятельной работы.

## 3 Теоретическое введение

Julia – высокоуровневый свободный язык программирования с динамической типизацией, созданный для математических вычислений. Эффективен также и для написания программ общего назначения. Синтаксис языка схож с синтаксисом других математических языков, однако имеет некоторые существенные отличия.

Для выполнения заданий была использована официальная документация Julia.

## 4 Выполнение лабораторной работы

Запустила Julia. (рис. 4.1).



```
PS C:\Users\MI> Julia
The latest version of Julia in the 'release' channel is 1.11.6+0.x64.mingw32. You currently have '1.11.3+0.x64.mingw32' installed. Run:

    juliaup update

in your terminal shell to install Julia 1.11.6+0.x64.mingw32 and update the 'release' channel to that version.

Documentation: https://docs.julialang.org
Type "?" for help, "]" for pkg help.
Version 1.11.3 (2025-01-21)
Official https://julialang.org/ release

julia>
```

Рис. 4.1: Julia



## 5 Простейшие операции на языке Julia

Повторила простейшие примеры для знакомства с синтаксисом Julia из лабораторной работы. (рис. 5.1-5.7).

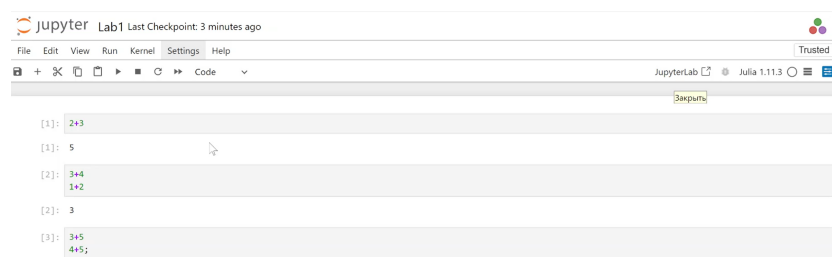


Рис. 5.1: Простейшие арифметические операции

```
[4]: ?println
      search: println print sprint pointer printstyled

[4]: println([io::IO], xs...)
      Print (using print) xs to io followed by a newline. If io is not supplied, prints to the default output stream stdout.

      See also printstyled to add colors etc.

Examples

julia> println("Hello, world")
Hello, world

julia> io = IOBuffer();

julia> println(io, "Hello", ',', " world.")

julia> String(take!(io))
"Hello, world.\n"
```

Рис. 5.2: Пример получения информации по функции

```

[5]: typeof(3), typeof(3.5), typeof(3/3.55), typeof(sqrt(3+4im)), typeof(pi)
[5]: (Int64, Float64, Float64, ComplexF64, Irrational{::N})
[6]: 1.0/0.0, 1.0/(-0.0), 0.0/0.0
[6]: (Inf, -Inf, NaN)
[7]: typeof(1.0/0.0), typeof(1.0/(-0.0)), typeof(0.0/0.0)
[7]: (Float64, Float64, Float64)
[8]: for T in [Int8, Int16, Int32, Int64, Int128, UInt8, UInt16, UInt32, UInt64, UInt128]
      println("%(lpad(T,7)): [$(typemin(T)), $(typemax(T))]\n")
    end
      Int8: [-128, 127]
      Int16: [-32768, 32767]
      Int32: [-2147483648, 2147483647]
      Int64: [-9223372036854775808, 9223372036854775807]
      Int128: [-170141183460469231731687303715884105728, 170141183460469231731687303715884105727]
      UInt8: [0, 255]
      UInt16: [0, 65535]
      UInt32: [0, 4294967295]
      UInt64: [0, 18446744073709551615]
      UInt128: [0, 340282366920938463463374607431768211455]

```

Рис. 5.3: Примеры определения типа числовых величин

```

[9]: Int(2.0), Char(2), typeof(Char(2))
[9]: (2, '\x02', Char)
[10]: typeof(promote(Int8(1), Float16(4.5), Float32(4.1)))
[10]: Tuple{Float32, Float32, Float32}

```

Рис. 5.4: Примеры приведения аргументов к одному типу

```

[12]: function f(x)
      x^2
    end
[12]: f (generic function with 1 method)
[13]: f(4)
[13]: 16
[14]: g(x)=x^2
[14]: g (generic function with 1 method)
[15]: g(8)
[15]: 64

```

Рис. 5.5: Примеры определения функций

```

[16]: a=[4 7 6]
[16]: 1×3 Matrix{Int64}:
      4  7  6
[17]: b=[1, 2, 3]
[17]: 3-element Vector{Int64}:
      1
      2
      3
[18]: a[2], b[2]
[18]: (7, 2)
[19]: a=1; b=2; c=3; d=4
      Am=[a b; c d]
[19]: 2×2 Matrix{Int64}:
      1  2
      3  4
[20]: Am[1,1], Am[1,2], Am[2,1], Am[2,2]
[20]: (1, 2, 3, 4)

```

Рис. 5.6: Примеры работы с массивами

```

[21]: aa = [1 2]
[21]: 1×2 Matrix{Int64}:
      1  2
[22]: AA = [1 2; 3 4]
[22]: 2×2 Matrix{Int64}:
      1  2
      3  4
[23]: aa*AA
[23]: 1×2 Matrix{Int64}:
      7 10
[24]: aa'
[24]: 2×1 adjoint{::Matrix{Int64}} with eltype Int64:
      1
      2
[25]: aa*AA*aa'
[25]: 1×1 Matrix{Int64}:
      27
[26]: aa, AA, aa'
[26]: ([1 2], [1 2; 3 4], [1; 2;:])

```

Рис. 5.7: Примеры работы с массивами

## 5.1 Основные функции Julia

Изучила документацию по функции `read()`. (рис. 5.8).

```
[27]: ?read
search: read read! rpad read break isready readdir Threads isreal lpad secd rem

[27]: read(io::IO, T)
Read a single value of type T from io, in canonical binary representation.

Note that Julia does not convert the endianness for you. Use ntoh or ltoh for this purpose.

read(io::IO, String)
Read the entirety of io, as a String (see also readchomp).

Examples

julia> io = IOBuffer("JuliaLang is a GitHub organization");

julia> read(io, Char)
'J': ASCII/Unicode U+004A (category Lu: Letter, uppercase)

julia> io = IOBuffer("JuliaLang is a GitHub organization");

julia> read(io, String)
"JuliaLang is a GitHub organization"

read(filename::AbstractString)
Read the entire contents of a file as a Vector{UInt8}.

read(filename::AbstractString, String)
Read the entire contents of a file as a string.

read(filename::AbstractString, args...)
Open a file and read its contents. args is passed to read: this is equivalent to open(io->read(io, args...), filename).
```

Рис. 5.8: Функция read

Примеры использования функции read(). (рис. 5.9).

```
[28]: content = read("example.txt")

[28]: 52-element Vector{UInt8}:
      0xd0
      0x9f
      0xd0
      0xb5
      0xd1
      0x80
      0xd0
      0xb2
      0xd0
      0xb0
      0xd1
      0x8f
      0x20
      ⋮
      0x81
      0xd1
      0x82
      0xd1
      0x80
      0xd0
      0xbe
      0xd0
      0xba
      0xd0
      0xb0
      0x0a

[29]: content = read("example.txt", String)

[29]: "Первая строка\nВторая строка\n"
```

Рис. 5.9: Примеры с функцией read

Функция `read()` читает содержимое файла или потока целиком в виде массива байт или другого типа, если его задать. Читает содержимое в одну строку, без переноса, с разделителями.

Изучила документацию по функции `readline()`. (рис. 5.10).

```
[30]: ?readline
search: readline readlines readlink readdir eachline readbytes! replace @inline

[30]: readline(io::IO=stdin; keep::Bool=false)
readline(filename::AbstractString; keep::Bool=false)
Read a single line of text from the given I/O stream or file (defaults to stdin). When reading from a file, the text is assumed to be encoded in UTF-8. Lines in the input end with "\n" or "\r\n" or the end of an input stream. When keep is false (as it is by default), these trailing newline characters are removed from the line before it is returned. When keep is true, they are returned as part of the line.

Return a String. See also copyline to instead write in-place to another stream (which can be a preallocated IOBuffer).

See also readuntil for reading until more general delimiters.

Examples

julia> write("my_file.txt", "JuliaLang is a GitHub organization.\nIt has many members.\n");

julia> readline("my_file.txt")
"JuliaLang is a GitHub organization."

julia> readline("my_file.txt", keep=true)
"JuliaLang is a GitHub organization.\n"

julia> rm("my_file.txt")
julia> print("Enter your name: ")
Enter your name:

julia> your_name = readline()
Logan
"Logan"
```

Рис. 5.10: Функция readline

Примеры использования функции readline(). (рис. 5.11).

```
[31]: content = readline("example.txt")

[31]: "Первая строка"
```

Рис. 5.11: Примеры с функцией readline

Эта функция читает одну строку из файла или потока. При достижении конца файла выбрасывает исключение.

Изучила документацию по функции readlines(). Примеры использования функции readlines(). (рис. 5.12).

```
[32]: ?readlines
search: readlines readline readlink readbytes! readdir eachline leading_ones

[32]: readlines(io::IO=stdin; keep::Bool=false)
readlines(filename::AbstractString; keep::Bool=false)
Read all lines of an I/O stream or a file as a vector of strings. Behavior is equivalent to saving the result of reading readline repeatedly with the same arguments and saving the resulting lines as a vector of strings. See also eachline to iterate over the lines without reading them all at once.

Examples

julia> write("my_file.txt", "JuliaLang is a GitHub organization.\nIt has many members.\n");

julia> readlines("my_file.txt")
2-element Vector{String}:
"JuliaLang is a GitHub organization."
"It has many members."

julia> readlines("my_file.txt", keep=true)
2-element Vector{String}:
"JuliaLang is a GitHub organization.\n"
"It has many members.\n"

julia> rm("my_file.txt")

[33]: content = readlines("example.txt")

[33]: 2-element Vector{String}:
"Первая строка"
"Вторая строка"
```

Рис. 5.12: Функция readlines

Эта функция читает все строки из файла и возвращает массив строк.

Изучила документацию по функции `readdlm()`. (рис. 5.13).

```
[36]: using DelimitedFiles

[37]: ?readdlm

search: readdlm readdir read real read! readchomp isreadonly readlink isreadable

[37]: readdlm(source, T::Type; options...)
The columns are assumed to be separated by one or more whitespaces. The end of line delimiter is taken as \n.
```

## Examples

```
julia> using DelimitedFiles

julia> x = [1; 2; 3; 4];

julia> y = [5; 6; 7; 8];

julia> open("delim_file.txt", "w") do io
    writedlm(io, [x y])
end;

julia> readdlm("delim_file.txt", Int64)
4x2 Matrix{Int64}:
 1  5
 2  6
 3  7
 4  8

julia> readdlm("delim_file.txt", Float64)
4x2 Matrix{Float64}:
 1.0  5.0
 2.0  6.0
 3.0  7.0
 4.0  8.0
```

Рис. 5.13: Функция `readdlm`

Примеры использования функции `readdlm()`. (рис. 5.14).

```
[41]: x = [2; 3; 2; 3];
      y = ["a"; "b"; "a"; "b"];

      open("test.txt", "w") do io
          for i in 1:length(x)
              println(io, string(x[i], " ", y[i]))
          end
      end

[42]: readdlm("test.txt")

[42]: 4x2 Matrix{Any}:
      2 "a"
      3 "b"
      2 "a"
      3 "b"
```

Рис. 5.14: Примеры с функцией `readdlm`

Эта функция читает данные из файла с разделителями (delimiter), например CSV или табличные данные. Возвращает матрицу или массив.

Примеры использования функций `print()` и `println()`. (рис. 5.15).

```
[43]: print("Hello, ")
      print("world!")

Hello, world!

[46]: println("Hello, ")
      print("world!")

Hello,
world!
```

Рис. 5.15: Примеры с функциями print и println

Функция print() выводит данные на экран без перевода строки. А println() - с переводом в конце строки.

Изучила документацию по функции show(). (рис. 5.16).

```
[47]: ?show

search: show @show chown throw Cshort chop showable ispow2

[47]: show(io::IO = stdout, x)
      Write a text representation of a value x to the output stream io. New types T should overload show(io::IO, x::T). The representation used by show generally includes Julia-specific formatting and type information, and should be parseable Julia code when possible.

repr returns the output of show as a string.

For a more verbose human-readable text output for objects of type T, define show(io::IO, ::MIME"text/plain", x::T) in addition. Checking the compact IOContext key (often checked as get(io, :compact, false)::Bool) of io in such methods is recommended, since some containers show their elements by calling this method with :compact => true.

See also print, which writes un-decorated representations.

Examples

julia> show("Hello World!")
"Hello World!"
julia> print("Hello World!")
Hello World!

show(io::IO, mime, x)
The display functions ultimately call show in order to write an object x as a given mime type to a given I/O stream io (usually a memory buffer), if possible. In order to provide a rich multimedia representation of a user-defined type T, it is only necessary to define a new show method for T, via: show(io, ::MIME"mime", x::T) = ..., where mime is a MIME-type string and the function body calls write (or similar) to write that representation of x to io. (Note that the MIME"" notation only supports literal strings; to construct MIME types in a more flexible manner use MIME{Symbol{""}}.)

For example, if you define a MyImage type and know how to write it to a PNG file, you could define a function show(io, ::MIME"image/png", x::MyImage) = ... to allow your images to be displayed on any PNG-capable AbstractDisplay (such as IJulia). As usual, be sure to import Base.show in order to add new methods to the built-in Julia function show.
```

Рис. 5.16: Функция show

Примеры использования функции show(). (рис. 5.17).

```
[48]: show([1, 2, 3])

[1, 2, 3]

[49]: show([1 2 3])

[1 2 3]
```

Рис. 5.17: Примеры с функцией show

Эта функция выводит данные в более “сыром” или структурированном виде. Часто используется для вывода объектов с отображением их внутреннего представления.

Изучила документацию по функции write(). (рис. 5.18).



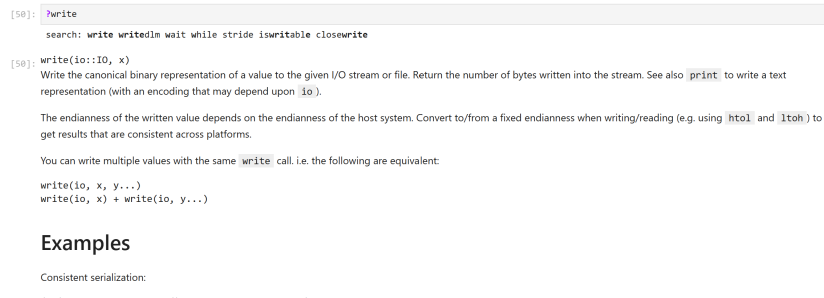


Рис. 5.18: Функция `write`

Примеры использования функции `write()`. (рис. 5.19-5.20).

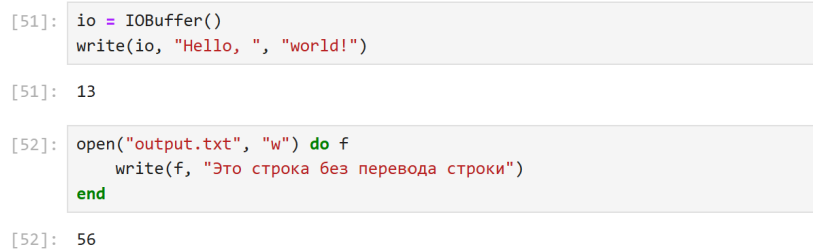


Рис. 5.19: Примеры с функцией `write`

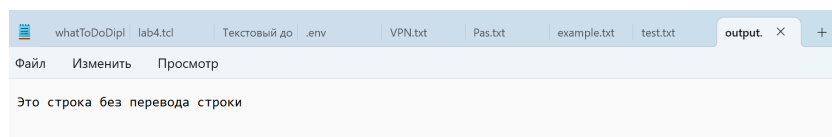


Рис. 5.20: Примеры с функцией `write`

Эта функция записывает байты или данные в файл или поток, не добавляет перевод строки, не форматирует данные.

## 5.2 Функция `parse()`

Изучила документацию функции `parse()`. (рис. 5.21).

```
[53]: ?parse
search: parse tryparse Base pairs falses parent

[53]: parse(type, str; base)
Parse a string as a number. For Integer types, a base can be specified (the default is 10). For floating-point types, the string is parsed as a decimal floating-point number.
Complex types are parsed from decimal strings of the form "R±Im" as a Complex{R,I} of the requested type; "i" or "j" can also be used instead of "im", and
"R" or "Im" are also permitted. If the string does not contain a valid number, an error is raised.

!!! compat "Julia 1.1" parse{Bool, str} requires at least Julia 1.1.

Examples

julia> parse{Int, "1234"}
1234
```

Рис. 5.21: Функция parse

Привела свои примеры её использования. (рис. 5.22).

```
[59]: a = "786"
      # typeof(a)
      b = parse{Int, a}
      typeof(b)

[59]: Int64

[61]: parse{Float64, "3.1415"}

[61]: 3.1415

[64]: parse{Bool, "true"}

[64]: true

[65]: parse{Int, "abc"}
ArgumentError: invalid base 10 digit 'a' in "abc"
Stacktrace:
 [1] tryparse_internal(::Type{Int64}, s::String, startpos::Int64, endpos::Int64, base::Int64, raise::Bool)
    @ Base ./parse.jl:143
 [2] #parse#619
    @ ./parse.jl:254 [inlined]
 [3] parse(::Type{Int64}, s::String)
    @ Base ./parse.jl:253
 [4] top-level scope
    @ In[65]:1

[66]: parse{Int, "1011"; base=2}

[66]: 11

[67]: parse{Int, "1F"; base=16}

[67]: 31
```

Рис. 5.22: Использование функции parse

Функция `parse()` в Julia используется для преобразования строки (String) в значение указанного типа. Это удобно, когда нужно конвертировать текстовые данные в числа, логические значения и другие типы.

## 5.3 Базовые математические операции

Изучила синтаксис Julia для базовых математических операций с разным типом переменных: сложение, вычитание, умножение, деление, возведение в степень, извлечение корня, сравнение, логические операции. Привела примеры с пояснениями по особенностям их применения: (рис. 5.23-5.31)

```
[69]: #сложение
sum_int = a + 5
[69]: 15
[70]: a+b
[70]: 13.5
[71]: b+c
[71]: 5.5 + 3.0im
[72]: a+e
MethodError: no method matching +(::Int64, ::String)
The function `+` exists, but no method is defined for this combination of argument types.
Closest candidates are:
+(::Any, ::Any, ::Any, ::Any...)
  @ Base operators.jl:598
+(::Real, ::Complex{Bool})
  @ Base complex.jl:322
+(::Integer, ::AbstractChar)
  @ Base char.jl:242
...
Stacktrace:
 [1] top-level scope
      @ In[72]:1
[73]: e+e
MethodError: no method matching +(::String, ::String)
The function `+` exists, but no method is defined for this combination of argument types.
String concatenation is performed with `*` (See also: https://docs.julialang.org/en/v1/manual/strings/#man-concatenation).
```

Рис. 5.23: Примеры для сложения

```
[74]: d+d
[74]: 2
[75]: true + false
[75]: 1
[76]: false+false
[76]: 0
[77]: c+d
[77]: 3 + 3im
```

Рис. 5.24: Примеры для сложения

```
[78]: # вычитание
a - 4
[78]: 6
[79]: b - a
[79]: -6.5
[80]: c - d
[80]: 1 + 3im
[81]: c - b
[81]: -1.5 + 3.0im
```

Рис. 5.25: Примеры для вычитания

```

[82]: # умножение
      a*a
[82]: 100
[83]: a*b
[83]: 35.0
[84]: b*b
[84]: 12.25
[85]: b*c
[85]: 7.0 + 10.5im
[86]: c*c
[86]: -5 + 12im
[87]: c*d
[87]: 2 + 3im
[88]: c*false
[88]: 0 + 0im
[89]: 2*e
      MethodError: no method matching *{::Int64, ::String}
      The function `*` exists, but no method is defined for this combination of argument types.

```

Рис. 5.26: Примеры для умножения

```

[90]: # деление
      a/4
[90]: 2.5
[91]: b/a
[91]: 0.35
[92]: c/b
[92]: 0.5714285714285714 + 0.8571428571428571im
[93]: c/d
[93]: 2.0 + 3.0im
[94]: true / false
[94]: Inf
[96]: # целочисленное деление
      div(10, 4)
[96]: 2
[101]: # остаток от деления
      mod(10, 5)
[101]: 0

```

Рис. 5.27: Примеры для деления

```

[102]: # возведение в степень
      a^2
[102]: 100
[103]: b^a
[103]: 275854.7353515625
[104]: c^a
[104]: -341525 - 145668im
[105]: c^d
[105]: 2 + 3im
[106]: d^0
[106]: true

```

Рис. 5.28: Примеры для возведения в степень

```

[107]: # извлечение корня
      sqrt(a)
[107]: 3.1622776601683795
[108]: sqrt(100)
[108]: 10.0
[109]: sqrt(b)
[109]: 1.8708286933869707
[110]: sqrt(c)
[110]: 1.6741492280355401 + 0.8959774761298381im
[111]: sqrt(d)
[111]: 1.0

```

Рис. 5.29: Примеры для извлечения корня

```

[114]: # равенство
      a == 10

[114]: true

[115]: # не равно
      a != 10

[115]: false

[118]: a > b, a, b

[118]: (true, 10, 3.5)

[119]: a < b

[119]: false

[121]: a >= b

[121]: true

[128]: a <= d

[128]: false

```

Рис. 5.30: Примеры для сравнения

```

[129]: true && true

[129]: true

[130]: true && false

[130]: false

[131]: true || false

[131]: true

[132]: true || true

[132]: true

[133]: !true

[133]: false

[134]: !false

[134]: true

```

Рис. 5.31: Примеры для логических операций

## 5.4 Операции над матрицами и векторами

Определила две матрицы и два вектора (вектор-строка и вектор-столбец).(рис. 5.32)

```
[142]: # матрицы и векторы
a = [1 2 3]
c = [1 1 1]

[142]: 1x3 Matrix{Int64}:
 1  1  1

[143]: b = [4, 5, 6]
d = [1, 2, 3]

[143]: 3-element Vector{Int64}:
 1
 2
 3

[139]: m1 = [1 1 1; 2 2 2; 1 1 1]

[139]: 3x3 Matrix{Int64}:
 1  1  1
 2  2  2
 1  1  1

[140]: m2 = [2 2 2; 1 1 1; 2 2 2]

[140]: 3x3 Matrix{Int64}:
 2  2  2
 1  1  1
 2  2  2
```

Рис. 5.32: Определение матриц и векторов

Привела примеры с пояснениями с операциями над матрицами и векторами: сложение, вычитание, скалярное произведение, транспонирование, умножение на скаляр (рис. 5.33-5.38)

```
[144]: # сложение векторов и матриц
sum1 = a+c

[144]: 1x3 Matrix{Int64}:
 2  3  4

[145]: sum2 = b+d

[145]: 3-element Vector{Int64}:
 5
 7
 9

[147]: sum3 = m1 + m2

[147]: 3x3 Matrix{Int64}:
 3  3  3
 3  3  3
 3  3  3
```

Рис. 5.33: Примеры для сложения векторов и матриц

```
[155]: # вычитание векторов и матриц  
diff1 = a-c
```

```
[155]: 1x3 Matrix{Int64}:  
0 1 2
```

```
[151]: diff2 = b-d
```

```
[151]: 3-element Vector{Int64}:  
3  
3  
3
```

```
[153]: diff3 = m1-m2
```

```
[153]: 3x3 Matrix{Int64}:  
-1 -1 -1  
1 1 1  
-1 -1 -1
```

```
[154]: diff4 = m2-m1
```

```
[154]: 3x3 Matrix{Int64}:  
1 1 1  
-1 -1 -1  
1 1 1
```

Рис. 5.34: Примеры для вычитания векторов и матриц

```
[158]: # скалярное произведение  
using LinearAlgebra  
dot_prod = dot(a, c)
```

```
[158]: 6
```

```
[159]: dot(a, b)
```

```
[159]: 32
```

```
[160]: dot(b, d)
```

```
[160]: 32
```

```
[161]: dot(m1, m2)
```

```
[161]: 18
```

Рис. 5.35: Примеры для скалярного произведения



```
[163]: transpose(a)

[163]: 3×1 transpose(::Matrix{Int64}) with eltype Int64:
      1
      2
      3

[164]: a'

[164]: 3×1 adjoint(::Matrix{Int64}) with eltype Int64:
      1
      2
      3

[165]: m1'

[165]: 3×3 adjoint(::Matrix{Int64}) with eltype Int64:
      1  2  1
      1  2  1
      1  2  1

[166]: transpose(m1)

[166]: 3×3 transpose(::Matrix{Int64}) with eltype Int64:
      1  2  1
      1  2  1
      1  2  1

[167]: b'

[167]: 1×3 adjoint(::Vector{Int64}) with eltype Int64:
      4  5  6
```

Рис. 5.36: Примеры для сложения транспонирования

```
[168]: scalar = 2
      scalar * m1

[168]: 3×3 Matrix{Int64}:
      2  2  2
      4  4  4
      2  2  2

[169]: scalar * a

[169]: 1×3 Matrix{Int64}:
      2  4  6

[170]: scalar * b

[170]: 3-element Vector{Int64}:
      8
     10
     12
```

Рис. 5.37: Примеры для умножения на скаляр

```

[171]: a*b
[171]: 1-element Vector{Int64}:
       32
[173]: a*m1
[173]: 1×3 Matrix{Int64}:
       8  8  8
[176]: m1*b
[176]: 3-element Vector{Int64}:
       15
       30
       15
[177]: m1*m2
[177]: 3×3 Matrix{Int64}:
       5  5  5
      10 10 10
       5  5  5

```

Рис. 5.38: Примеры для скалярного произведения

## 6 Выводы

В результате выполнения данной лабораторной работы я подготовила рабочее пространство и инструментарий для работы с языком программирования Julia, на простейших примерах познакомилась с основами синтаксиса Julia

## **Список литературы**