Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Мануал по работе с ClosureScript

Москва — 2023 г.

Содержание

Введение

Что такое ClosureScript?

ClosureScript [1] — компилятор для Closure [2], выдающий в результате код на JavaScript.

Что такое Closure?

Closure — диалект языка LISP, являющийся динамическим компилируемым языком программирования, поддерживающий доступ к фреймворкам, написанным на Java. Из-за своего родства с LISP поддерживает функциональное программирование и использование макросов.

Что нужно для того, чтобы начать писать на Closure?

Во-первых, среда разработки или текстовый редактор для Closure — подходящих несколько, например Emacs, Intellij IDEA, VS Code. В рамках данного мануала будет рассмотрено использование текстового редактора VS Code для работы с Closure.

Во-вторых, сам Closure — он доступен для установки под MacOS, Linux и Windows.

В-третьих,

Установка

VS Code и Calva

Установить VS Code под свою платоформу можно по ссылке: `https://code.visualstudio.com/Download`

Для тех, кто не имеет опыта использования IDE или желает научиться использовать VS Code, рекомендуются к прочтению следующие статьи:

1) `https://habr.com/ru/post/490754/` — статья на русском языке

2) `https://code.visualstudio.com/docs/introvideos/basics` — статья на английском языке

Далее необходимо установить Calva — расширение для VS Code, поддерживающее ClojureScript и помогающее разрабатывать ПО на Clojure. Для этого в левом меню VS Code необходимо перейти в раздел "Расширения" (см. скриншот 1), сделать поиск "Calva" и установить найденное расширение (см. скриншот 2).

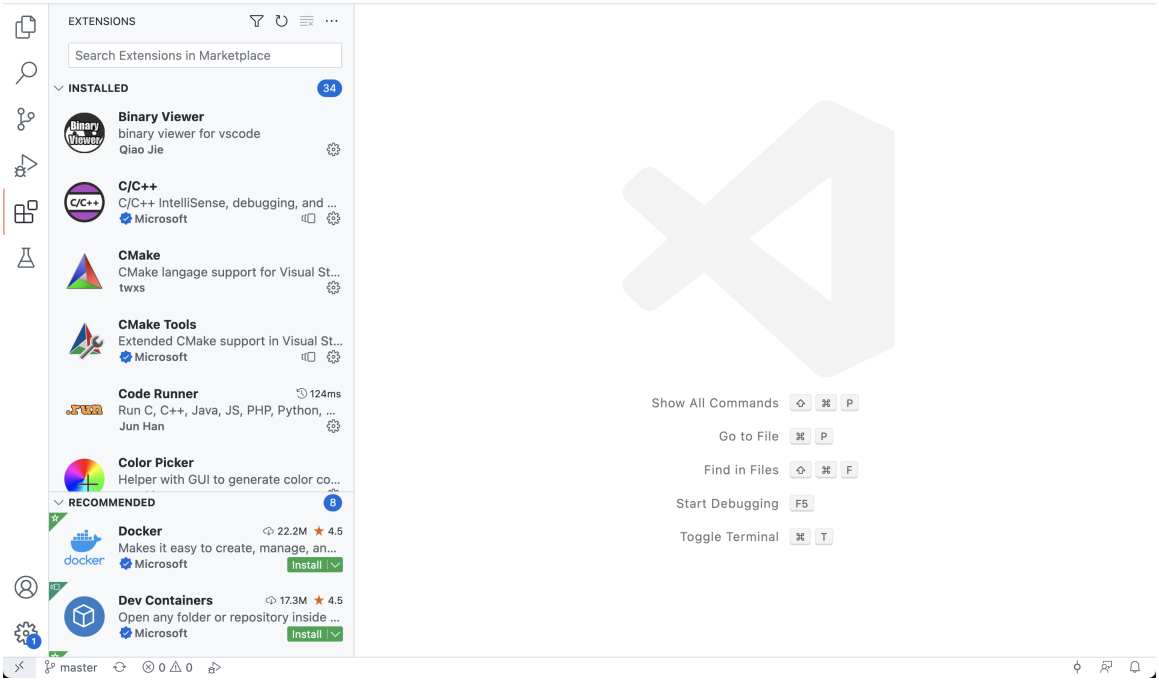

Рисунок 1 – Раздел "Расширения" VS Code

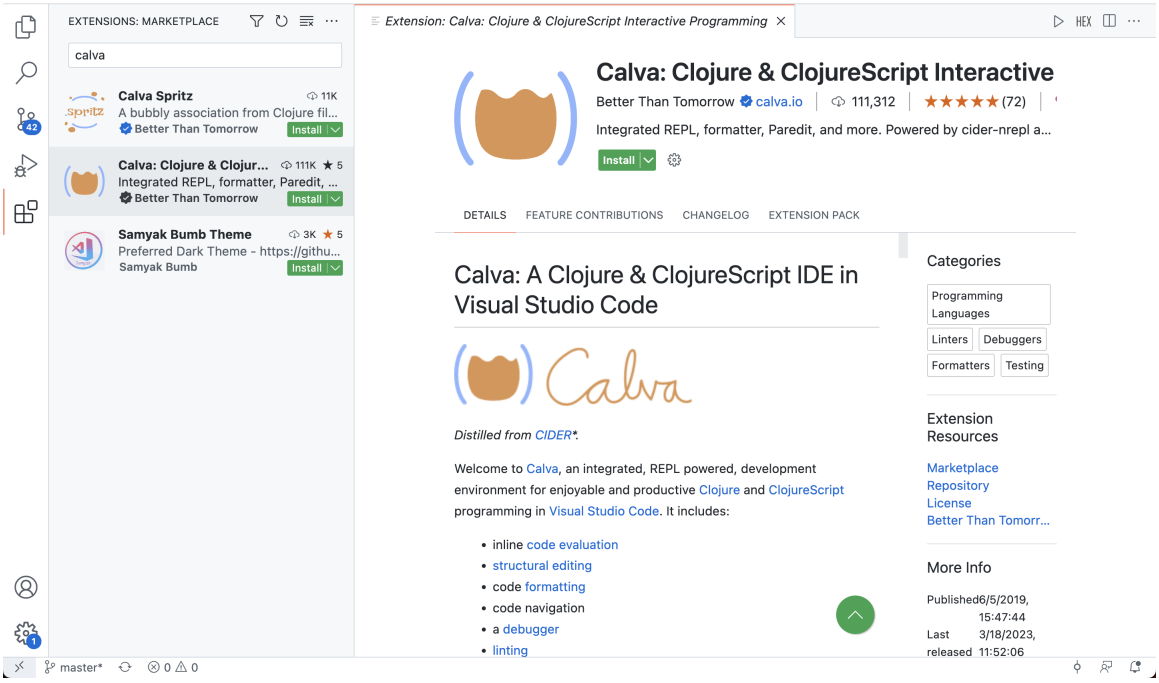

Рисунок 2 – Расширение Calva

Clojure

Для того, чтобы установить Clojure, воспользуемся инструкциями с официального сайта: `https://clojure.org/guides/install_clojure`.

В приведённой статье (на английском языке) присутствую инструкции по установке Clojure на ОС MacOS, Linux-подобные ОС (Ubuntu, Debian), а также Windows. В данном мануале будет рассмотрена установка Clojure на Linux-подобные ОС посредством установщика пакетов brew.

Для установки необходимо открыть терминал и выполнить в нём следующие команды:

Листинг 1 – Установка brew

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/
    Homebrew/install/HEAD/install.sh)"
(echo; echo 'eval "$(/home/linuxbrew/.linuxbrew/bin/brew shellenv)"'
    ) >> /home/<YOURUSERNAME>/.profile
eval "$(/home/linuxbrew/.linuxbrew/bin/brew shellenv)"
sudo apt-get install build-essentials
```

После установки brew необходимо установить и сам Clojure:

Листинг 2 – Установка Clojure

```
1 brew install clojure/tools/clojure
```

LISP

Поскольку Clojure является диалектом LISP, стоит также установить и базовый интерпретатор LISP, например, Common Lisp. Для ОС семейства Linux это можно сделать следующей командой:

Листинг 3 – Установка Common Lisp

```
1 sudo apt-get install sbcl
```

Начало работы и примеры

Hello World, Clojure

Откроем VS Code и нажмём Ctrl + Shift + P (данное сочетание клавиш откроет паллет управления в VS Code) и введём следующий текст: Calva: Fire up the Getting Started REPL Среди появившихся результатов выберем единственный:

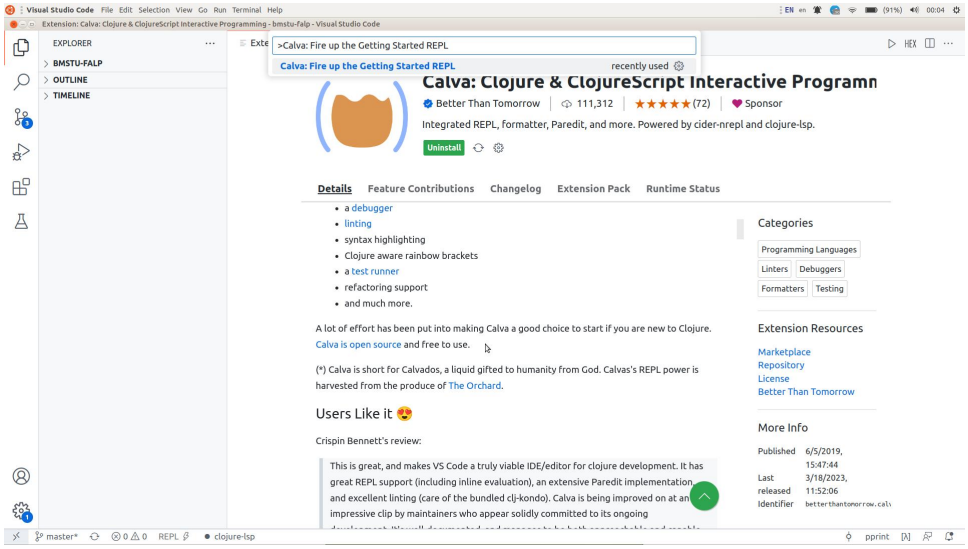

Рисунок 3 – Запуск Calva

После запуска Calva мы увидим примерно следующее:

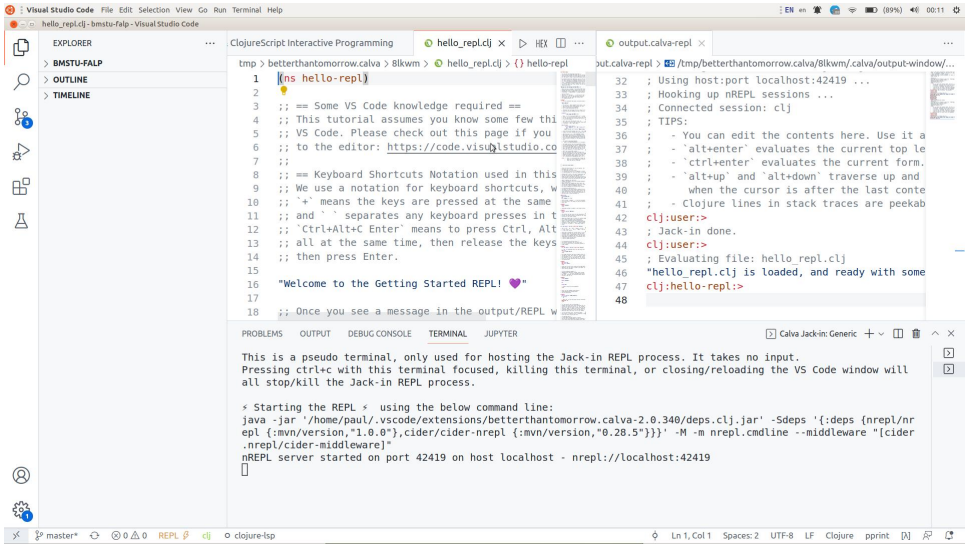

Рисунок 4 – После запуска Calva

Очистим содержимое левого файла и напишем в него ”Hello, World!”:

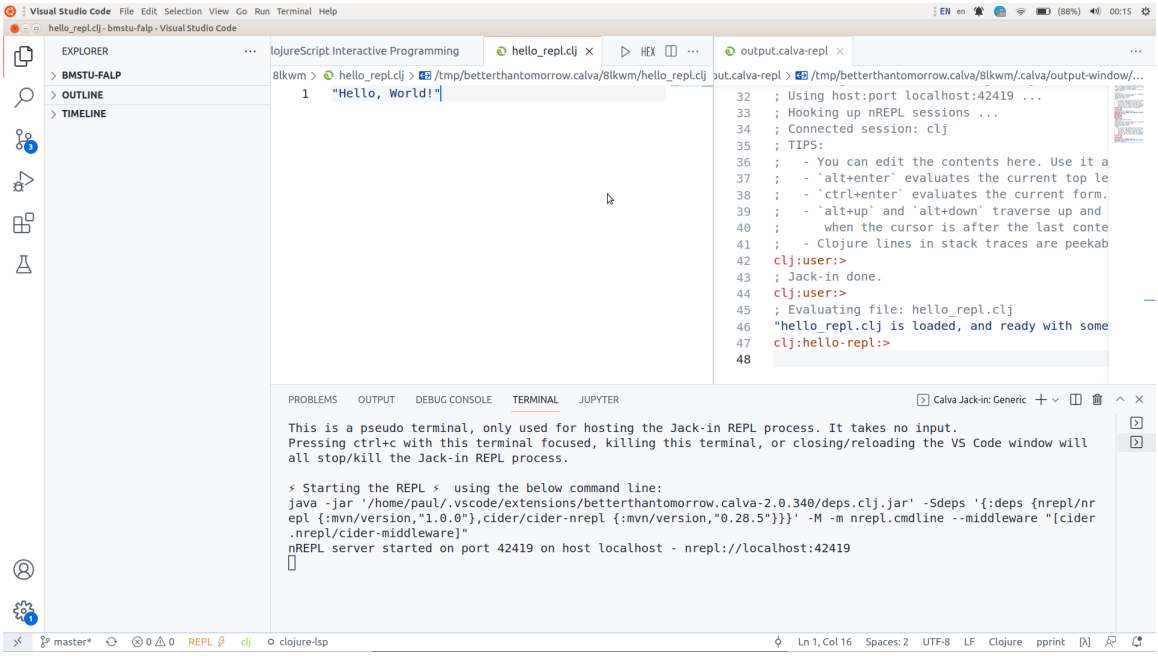

Рисунок 5 – Напишем ”Hello, World!”

Нажмём Alt + Enter и увидим результат:

Рисунок 6 – Hello, World!

Hello World, ClojureScript

Чтобы запустить Hello World с использованием ClojureScript, мы будем выполнять действия по следующему гайду:

`https://clojurescript.org/guides/quick-start`

Для этого создадим папку hello-world и настроим её содержимое следующим образом:

Листинг 4 – ClojureScirpt

```
hello-world          # Our project folder
    src              # The CLJS source code for our project
        hello_world  # Our hello_world namespace folder
            core.cljs # Our main file
    cljs.jar         # (Windows only) The standalone Jar you
      downloaded earlier
    deps.edn         # (macOS/Linux only) A file for listing our
      dependencies
```

Содержимое deps.edn:

Листинг 5 – Содержимое deps.edn

```
{:deps {org.clojure/clojurescript {:mvn/version "1.11.54"}}}
```

Содержимое src/hello_world/core.cljs:

Листинг 6 – Содержимое src/hello_world/core.cljs

```
(ns hello-world.core)

(println "Hello world!")
```

После чего запустим в терминал следующую команду:

Листинг 7 – Запуск в терминале

```
clj -M --main cljs.main --compile hello-world.core --repl
```

В браузере при этом должна открыться следующая страница:

После этого в терминале, из которого произходил запуск, должно появиться сообщение Hello World!

Попробуем добавить несколько функций. Для этого изменим содержимое src/hello_world/core.cljs:

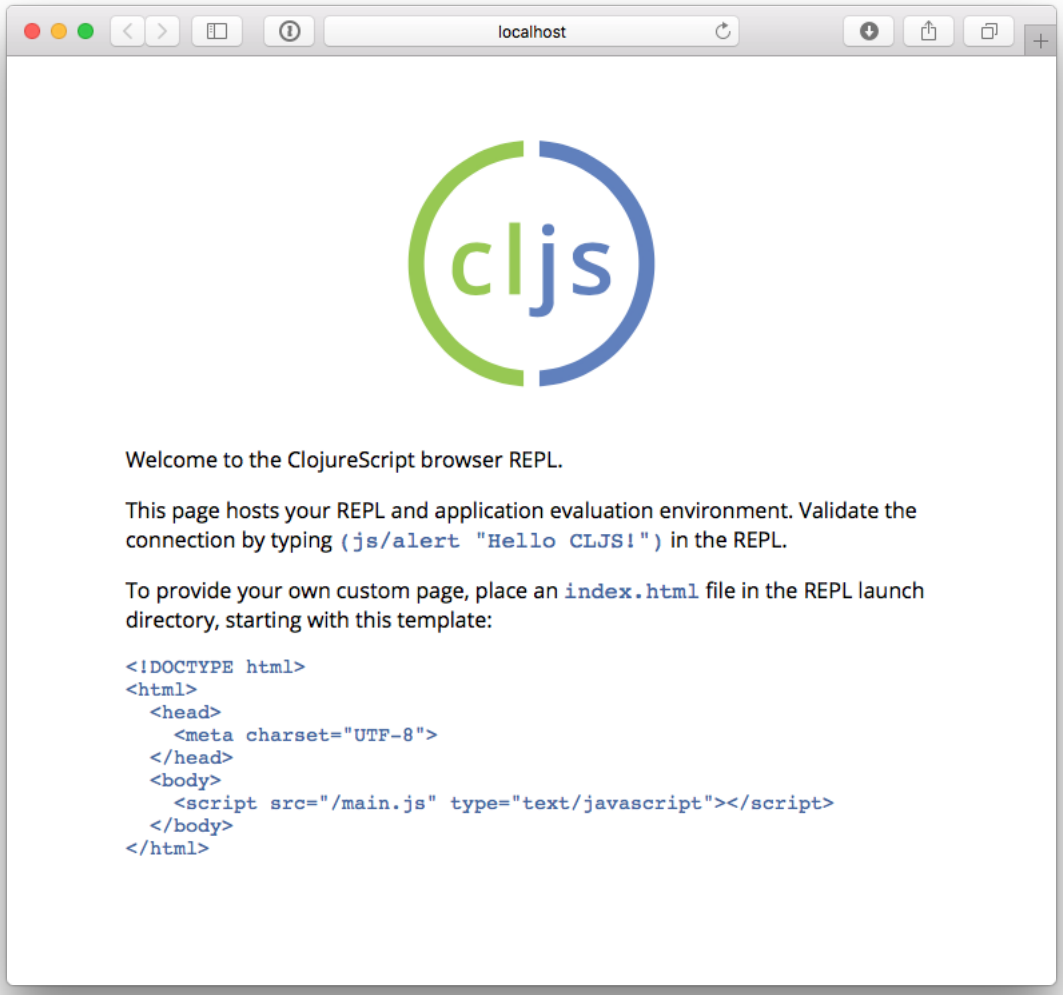

Рисунок 7 – Hello, World!

Листинг 8 – Содержимое src/hello_world/core.cljs

```
1  (ns hello-world.core)
2
3  (println "Hello world!")
4
5  ;; ADDED
6  (defn average [a b]
7      (/ (+ a b) 2.0))
8
9  (defn plus [a b]
10     (+ a b))
```

Перекомпилируем рабочее пространство, выполнив в терминале следующие команды:

Листинг 9 – Обновление страницы проверка новых функций

```
(require '[hello-world.core :as hello] :reload)
(hello/average 20 13)
(hello/plus 1 21)
```

В терминале должны отобразиться результаты работы функций - числа 16.5 и 22.

Отдельно отметим, что в папке out можно найти скомпилированный JavaScript код. Исследовав её содержимое, можно придти к выводу, что полученный код является ни слишком оптимизированным. Для того, чтобы получить более оптимизированную версию кода, необходимо запустить ClojureScript компилятор со следующими опциями:

Листинг 10 – Получение оптимизированной сборки

```
clj -M -m cljs.main --optimizations advanced -c hello-world.core
```

Оптимизированная сборка (со значением advanced) может быть использована для получения итоговой сборки, которая будет использована не разработчиками, а пользователями.

Нахождение обратной матрицы, ClojureScript

Как более сложный пример программы на ClojureScript предлагается рассмотреть функции нахождения для квадратной матрицы обратной матрицы и детерминанта:

Листинг 11 – Нахождение обратной матрицы

```
(ns hello-repl)

(defn matMake
  [rows cols val]
  (with-local-vars [result [], i 0, j 0, tmp []]
    (while (< @i rows)
      (var-set tmp [])
      (var-set j 0)
```

```
9         (while (< @j cols)
10          (var-set tmp (conj @tmp val))
11          (var-set j (inc @j)))
12        (var-set result (conj @result @tmp))
13        (var-set i (inc @i)))
14      @result))
15
16 (defn vecMake
17   [n val]
18   (with-local-vars [result [], i 0]
19     (while (< @i n)
20       (var-set result (conj @result val))
21       (var-set i (inc @i)))
22     @result))
23
24 (defn matDecompose
25   [matrix n lum perm result]
26   (with-local-vars [tmp [], toggle 1, i 0, j 0, k 0, mx 0.0, piv 0,
        xij 0.0, ij 0.0, t 0.0]
27     (while (< @i n)
28       (var-set j 0)
29       (var-set tmp [])
30       (while (< @j n)
31         (var-set tmp (conj @tmp ((matrix @i) @j)))
32         (var-set j (inc @j)))
33       (var-set lum (assoc @lum @i @tmp))
34       (var-set i (inc @i)))
35
36     (var-set i 0)
37     (var-set tmp [])
38     (while (< @i n)
39       (var-set tmp (conj @tmp @i))
40       (var-set i (inc @i)))
41     (var-set perm (vec @tmp))
42
43     (var-set i 0)
44     (var-set j 0)
45     (while (< @j (- n 1))
46       (var-set mx (abs ((@lum @j) @j)))
47       (var-set piv @j)
48       (var-set i (inc @j))
```

```
49
50        (while (< @i n)
51          (var-set ij (abs ((@lum @i) @j)))
52          (if (> @ij @mx)
53            (do
54              (var-set mx @ij)
55              (var-set piv @i)
56              true))
57
58          (var-set i (inc @i)))
59
60        (if-not (= @piv @j)
61          (do (var-set t (@lum @piv))
62              (var-set lum (assoc @lum @piv (@lum @j)))
63              (var-set lum (assoc @lum @j @t))
64
65              (var-set t (@perm @piv))
66              (var-set perm (assoc @perm @piv (@perm @j)))
67              (var-set perm (assoc @perm @j @t))
68
69              (var-set toggle (* -1 @toggle))
70              true))
71
72        (var-set xij ((@lum @j) @j))
73        (if-not (= @xij 0)
74          (do
75            (var-set i (inc @j))
76
77            (while (< @i n)
78              (var-set tmp (@lum @i))
79              (var-set ij (/ ((@lum @i) @j) @xij))
80              (var-set tmp (assoc @tmp @j @ij))
81              (var-set k (inc @j))
82              (var-set lum (assoc @lum @i @tmp))
83
84              (while (< @k n)
85
86                (var-set tmp (assoc @tmp @k (- (@tmp @k) (* ((@lum @j)
                      @k) @ij))))
87                (var-set k (inc @k)))
88
```

```
89              (var-set lum (assoc @lum @i @tmp))
90
91              (var-set i (inc @i)))
92
93            true))
94        (var-set j (inc @j)))
95
96
97      (var-set result @toggle)))
98
99  (defn matDeterminant
100   [matrix n]
101   (with-local-vars [result 0, lum (matMake n n 0.0), perm (vecMake n
          0.0), i 0]
102     (matDecompose matrix n lum perm result)
103     (while (< @i n)
104       (var-set result (* @result ((@lum @i) @i)))
105       (var-set i (inc @i)))
106     @result))
107
108 (defn myReduce
109   [n lum b x]
110   (with-local-vars [i 0, j 0, tmp [], sum 0.0]
111
112     (var-set i 0)
113     (var-set tmp [])
114     (while (< @i n)
115       (var-set tmp (conj @tmp (@b @i)))
116       (var-set i (inc @i)))
117     (var-set x (vec @tmp))
118
119     (var-set i 1)
120     (while (< @i n)
121       (var-set sum (@x @i))
122       (var-set j 0)
123
124       (while (< @j @i)
125         (var-set sum (- @sum (* ((@lum @i) @j) (@x @j))))
126         (print @sum)
127         (print "\n")
128         (var-set j (inc @j)))
```

```
129
130       (var-set x (assoc @x @i @sum))
131
132       (var-set i (inc @i)))
133     (var-set x (assoc @x (- n 1) (/ (@x (- n 1)) ((@lum (- n 1)) (-
          n 1)))))
134
135     (var-set i (- n 2))
136
137     (while (>= @i 0)
138       (var-set sum (@x @i))
139       (var-set j (inc @i))
140       (while (< @j n)
141         (var-set sum (- @sum (* ((@lum @i) @j) (@x @j))))
142         (print @sum)
143         (print "\n")
144         (var-set j (inc @j)))
145       (var-set x (assoc @x @i (/ @sum ((@lum @i) @i))))
146
147       (var-set i (dec @i)))
148     @x))
149
150 (defn matInverse
151   [matrix n]
152   (with-local-vars [result (matMake n n 0.0), lum (matMake n n 0.0),
        perm (vecMake n 0.0), i 0, j 0, tmp [], det 0.0, b (vecMake n
       0.0), x (vecMake n 0.0)]
153
154     (while (< @i n)
155       (var-set j 0)
156       (var-set tmp [])
157       (while (< @j n)
158         (var-set tmp (conj @tmp ((matrix @i) @j)))
159         (var-set j (inc @j)))
160       (var-set result (assoc @result @i @tmp))
161       (var-set i (inc @i)))
162
163     (matDecompose matrix n lum perm det)
164
165     (var-set i 0)
166     (while (< @i n)
```

```
167       (var-set j 0)
168       (while (< @j n)
169         (if (= @i (@perm @j))
170           (var-set b (assoc @b @j 1.0))
171           (var-set b (assoc @b @j 0.0)))
172         (var-set j (inc @j)))
173
174       (myReduce n lum b x)
175
176       (var-set j 0)
177       (while (< @j n)
178         (var-set tmp (@result @j))
179         (var-set tmp (assoc @tmp @i (@x @j)))
180         (var-set result (assoc @result @j @tmp))
181         (var-set j (inc @j)))
182
183       (var-set i (inc @i)))
184
185
186     @result))
187
188 ;; (matDeterminant [[1.0 2.0] [3.0 4.0]] 2)
189 ;; (matDeterminant [[1.0 2.0 3.0] [4.0 5.0 6.0] [7.0 8.0 9.0]] 3)
190 ;; (matInverse [[1.0 2.0] [3.0 4.0]] 2)
191 ;; (matInverse [[3.0 7.0 2.0 5.0] [4.0 0.0 1.0 1.0] [1.0 6.0 3.0
    0.0] [2.0 8.0 4.0 3.0]] 4)
```

Список использованных источников

1. ClosueScript [Электронный ресурс]. Режим доступа: `https://clojurescript.org/` (дата обращения: 22.03.2023).

2. Closure [Электронный ресурс]. Режим доступа: `https://clojure.org/` (дата обращения: 22.03.2023).

3. Баварин И. И. Высшая математика: учебник по естественно-научным направлениям и специальностям. – М.: Гуманит. изд. центр ВЛАДОС, 2003. С. 45–49.

4. Kakaradov B. Ultra-Fast Matrix Multiplication: An Empirical Analysis of Highly Optimized Vector Algorithms, 2004 [Электронный ресурс]. Режим досступа: `https://cs.stanford.edu/people/boyko/pubs/MatrixMult_SURJ_2004.pdf` (дата обращения: 04.10.2022).