



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 КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.04** Программное обеспечение ЭВМ и
информационные технологии

О Т Ч Е Т

по лабораторной работе № 0 4

Название: *Взаимодействие серверов. Введение в Prolog.*

Дисциплина: *Архитектура ЭВМ*

Студент

ИУ7И-56Б

(Группа)

Нгуен Ф. С.

(Подпись, дата)

(И.О. Фамилия)

Преподаватель

Попов А. Ю.

(Подпись, дата)

(И.О. Фамилия)

Москва, 2020

Цель работы:

Ознакомиться с взаимодействием серверов в Node.js.

Ознакомиться с особенностями и базовыми принципами программирования на Prolog.

❖ Задание 1

Создать сервер **А**. На стороне сервера хранится файл с содержимым в формате **JSON**. При получении запроса на **/insert/record** идёт добавление записи в файл. При получении запроса на **/select/record** идёт получение записи из файла. Каждая запись хранит информацию о машине (*название* и *стоимость*).

Создать сервер **Б**. На стороне сервера хранится файл с содержимым в формате **JSON**. Каждая запись в файле хранит информацию о складе и массиве машин, находящихся на данном складе. То есть каждая запись хранит в себе название склада (*строку*) и массив названий машин (*массив строк*). При получении запроса на **/insert/record** идёт добавление записи в файл. При получении запроса на **/select/record** идёт получение записи из файла.

Создать сервер **С**. Сервер выдаёт пользователю страницы с формами для ввода информации. При этом сервер взаимодействует с серверами **А** и **Б**. Реализовать для пользователя функции:

- создание нового типа машины
- получение информации о стоимости машины по её типу
- создание нового склада с находящимися в нём машинами
- получение информации о машинах на складе по названию склада

Реализовать удобный для пользователя интерфейс взаимодействия с системой (использовать поля ввода и кнопки).

Код программы:

```
1. "use strict";
2.
3. class ServerA {
4.     static fs = require("fs");
5.     static express = require("express");
6.
7.     constructor(port) {
8.         this.app = ServerA.express();
9.         this.port = port;
10.
11.         try {
12.             this.app.listen(this);
13.             console.log(` Starting server on port ${this.port}... `);
14.         } catch (error) {
15.             console.log(" Failure while starting server!");
16.             throw new Error(' Port is unavalible!');
17.         }
18.     }
19. }
```

```

18.
19.     this.app.use(this.getHeaders);
20.     this.app.use(ServerA.express.static(__dirname + '/static'));
21.     this.app.post('/insert/record', this.insertRecord);
22.     this.app.post('/select/record', this.selectRecord);
23.     console.log(" Server started succesfully!");
24. }
25.
26. getHeaders(request, response, next) {
27.     response.header("Cache-Control", "no-cache, no-store, must-revalidate");
28.     response.header("Access-Control-Allow-Headers", "Origin, X-Requested-
With, Content-Type, Accept");
29.     response.header("Access-Control-Allow-Origin", "*");
30.     next();
31. }
32.
33. insertRecord(request, response) {
34.     function loadBody(request, callback) {
35.         let body = [];
36.         request.on('data', (chunk) => {
37.             body.push(chunk);
38.         }).on('end', () => {
39.             body = Buffer.concat(body).toString();
40.             callback(body);
41.         });
42.     }
43.
44.     loadBody(request, function(body) {
45.         const obj = JSON.parse(body);
46.         const name = obj.name;
47.         const price = obj.price;
48.
49.         const storage_path = "data/cars.json";
50.         const fd = ServerA.fs.readFileSync(storage_path, "utf8")
51.         let storage = fd.length ? new Map(JSON.parse(fd)) : new Map();
52.
53.         const name_exists = storage.has(name);
54.
55.         let added = false;
56.
57.         if (!name_exists) {
58.             added = true;
59.             storage.set(name, price);
60.             ServerA.fs.writeFileSync(storage_path, JSON.stringify([...storage]));
61.         }
62.
63.         response.end(JSON.stringify({answer: added}));
64.     });
65. }
66.
67. selectRecord(request, response) {
68.     function loadBody(request, callback) {
69.         let body = [];
70.         request.on('data', (chunk) => {
71.             body.push(chunk);
72.         }).on('end', () => {
73.             body = Buffer.concat(body).toString();
74.             callback(body);
75.         });
76.     }
77.
78.     loadBody(request, function(body) {
79.         const obj = JSON.parse(body);
80.         const name = obj.name;
81.
82.         const storage_path = "data/cars.json";
83.         const fd = ServerA.fs.readFileSync(storage_path, "utf8")
84.         let storage = fd.length ? new Map(JSON.parse(fd)) : new Map();
85.

```

```

86.         let found = false;
87.         let price;
88.
89.         if (storage.has(name)) {
90.             found = true;
91.             price = storage.get(name);
92.         }
93.
94.         response.end(JSON.stringify({answer: found,
95.                                     price: price}));
96.     });
97. }
98. }
99.
100. class ServerB {
101.     static fs = require("fs");
102.     static express = require("express");
103.
104.     constructor(port) {
105.         this.app = ServerB.express();
106.         this.port = port;
107.
108.         try {
109.             this.app.listen(this);
110.             console.log(` Starting server on port ${this.port}... `);
111.         } catch (error) {
112.             console.log(" Failure while starting server!");
113.             throw new Error(' Port is unavalible!');
114.         }
115.
116.         this.app.use(this.getHeaders);
117.         this.app.use(ServerB.express.static(__dirname + '/static'));
118.         this.app.post('/insert/record' , this.insertRecord);
119.         this.app.post('/select/record', this.selectRecord);
120.         console.log(" Server started succesfully!");
121.     }
122.
123.     getHeaders(request, response, next) {
124.         response.header("Cache-Control", "no-cache, no-store, must-
revalidate");
125.         response.header("Access-Control-Allow-Headers", "Origin, X-Requested-
With, Content-Type, Accept");
126.         response.header("Access-Control-Allow-Origin", "*");
127.         next();
128.     }
129.
130.     loadBody(request, callback) {
131.         let body = [];
132.         request.on('data', (chunk) => {
133.             body.push(chunk);
134.         }).on('end', () => {
135.             body = Buffer.concat(body).toString();
136.             callback(body);
137.         });
138.     }
139.
140.     insertRecord(request, response) {
141.         function loadBody(request, callback) {
142.             let body = [];
143.             request.on('data', (chunk) => {
144.                 body.push(chunk);
145.             }).on('end', () => {
146.                 body = Buffer.concat(body).toString();
147.                 callback(body);
148.             });
149.         }
150.
151.         console.log(1);
152.         loadBody(request, function(body) {

```

```

153.         const obj = JSON.parse(body);
154.         const name = obj.name;
155.         const cars = obj.cars;
156.
157.         const storage_path = "data/storage.json";
158.         const fd = ServerB.fs.readFileSync(storage_path, "utf8")
159.         let storage = fd.length ? new Map(JSON.parse(fd)) : new Map();
160.
161.         console.log(name, cars);
162.         console.log(storage);
163.
164.         const name_exists = storage.has(name);
165.
166.         let added = false;
167.
168.         if (!name_exists) {
169.             added = true;
170.             storage.set(name, cars);
171.             ServerB.fs.writeFileSync(storage_path, JSON.stringify([...storag
172. e]));
173.         }
174.         response.end(JSON.stringify({answer: added}));
175.     });
176. }
177.
178. selectRecord(request, response) {
179.     function loadBody(request, callback) {
180.         let body = [];
181.         request.on('data', (chunk) => {
182.             body.push(chunk);
183.         }).on('end', () => {
184.             body = Buffer.concat(body).toString();
185.             callback(body);
186.         });
187.     }
188.
189.     loadBody(request, function(body) {
190.         const obj = JSON.parse(body);
191.         const name = obj.name;
192.
193.         const storage_path = "data/storage.json";
194.         const fd = ServerB.fs.readFileSync(storage_path, "utf8")
195.         let storage = fd.length ? new Map(JSON.parse(fd)) : new Map();
196.
197.         let found = false;
198.         let cars;
199.
200.         if (storage.has(name)) {
201.             found = true;
202.             cars = storage.get(name);
203.         }
204.
205.         response.end(JSON.stringify({answer: found,
206.                                     cars: cars}));
207.     });
208. }
209. }
210.
211. class ServerC {
212.     static fs = require("fs");
213.     static express = require("express");
214.
215.     constructor(port) {
216.         this.app = ServerC.express();
217.         this.port = port;
218.
219.         try {
220.             this.app.listen(this);

```

```

221.         console.log(` Starting server on port ${this.port}... `);
222.     } catch (error) {
223.         console.log(" Failure while starting server!");
224.         throw new Error(' Port is unavalible!');
225.     }
226.
227.     this.app.use(this.getHeaders);
228.     this.app.use(ServerC.express.static(__dirname + "/static"));
229.     this.app.post('/add_car', this.addCar);
230.     this.app.get('/get_car', this.getCar);
231.     this.app.post('/add_storage', this.addStorage);
232.     this.app.get('/get_storage', this.getStorage);
233.     this.app.get('/Task71', this.startTask7);
234.     console.log(" Server started succesfully!");
235. }
236.
237. getHeaders(request, response, next) {
238.     response.header("Cache-Control", "no-cache, no-store, must-
revalidate");
239.     response.header("Access-Control-Allow-Headers", "Origin, X-Requested-
With, Content-Type, Accept");
240.     response.header("Access-Control-Allow-Origin", "*");
241.     next();
242. }
243.
244. startTask7(request, response) {
245.     const nameString = "./C/71C.html";
246.     if (fs.existsSync(nameString)) {
247.         const contentString = fs.readFileSync(nameString, "utf8");
248.         response.end(contentString);
249.     }
250.     response.end(JSON.stringify({result: "page not found"}));
251. }
252. };
253.
254.
255. addCar(request, response) {
256.     const name = request.query.name;
257.     const price = request.query.price;
258.
259.     function sendPost(url, body, callback) {
260.         const headers = {};
261.         const requests = require("request");
262.
263.         headers["Cache-Control"] = "no-cache, no-store, must-revalidate";
264.         headers["Connection"] = "close";
265.
266.         requests.post({
267.             url: url,
268.             body: body,
269.             headers: headers
270.         }, function(error, response, body) {
271.             if (error) {
272.                 callback(null);
273.             } else {
274.                 callback(body);
275.             }
276.         });
277.     }
278.
279.     sendPost("http://localhost:5015/insert/record",
280.         JSON.stringify({name: name,
281.             price: price
282.         }), function(answerString) {
283.             response.end(answerString);
284.         });
285. }
286.
287. getCar(request, response) {

```

```

288.         const name = request.query.name;
289.
290.         function sendPost(url, body, callback) {
291.             const headers = {};
292.             const requests = require("request");
293.
294.             headers["Cache-Control"] = "no-cache, no-store, must-revalidate";
295.             headers["Connection"] = "close";
296.
297.             requests.post({
298.                 url: url,
299.                 body: body,
300.                 headers: headers
301.             }, function(error, response, body) {
302.                 if (error) {
303.                     callback(null);
304.                 } else {
305.                     callback(body);
306.                 }
307.             });
308.         }
309.
310.         sendPost("http://localhost:5015/select/record",
311.             JSON.stringify({name: name}),
312.             function(answerString) {
313.                 response.end(answerString);
314.             });
315.     }
316.
317.     addStorage(request, response) {
318.         const name = request.query.name;
319.         const cars = request.query.cars;
320.
321.         function sendPost(url, body, callback) {
322.             const headers = {};
323.             const requests = require("request");
324.             headers["Cache-Control"] = "no-cache, no-store, must-revalidate";
325.             headers["Connection"] = "close";
326.
327.             requests.post({
328.                 url: url,
329.                 body: body,
330.                 headers: headers
331.             }, function(error, response, body) {
332.                 if (error) {
333.                     callback(null);
334.                 } else {
335.                     callback(body);
336.                 }
337.             });
338.         }
339.
340.         sendPost("http://localhost:5020/insert/record",
341.             JSON.stringify({name: name,
342.                 cars: cars}),
343.             function(answerString) {
344.                 response.end(answerString);
345.             });
346.     }
347.
348.     getStorage(request, response) {
349.         const name = request.query.name;
350.         const requests = require("request");
351.
352.         function sendPost(url, body, callback) {
353.             const headers = {};
354.             headers["Cache-Control"] = "no-cache, no-store, must-revalidate";
355.             headers["Connection"] = "close";
356.

```



```

357.         requests.post({
358.             url: url,
359.             body: body,
360.             headers: headers
361.         }, function(error, response, body) {
362.             if (error) {
363.                 callback(null);
364.             } else {
365.                 callback(body);
366.             }
367.         });
368.     }
369.
370.     sendPost("http://localhost:5020/select/record",
371.             JSON.stringify({name: name}),
372.             function(answerString) {
373.                 response.end(answerString);
374.             });
375.     }
376. }
377.
378. let serverA = new ServerA(5000);
379. let serverB = new ServerB(5002);
380. let serverC = new ServerC(5004);

```

./static/html/71C.html

```

1. <!DOCTYPE html>
2. <html>
3. <head>
4.     <meta charset="UTF-8">
5.     <title> Task 7.1</title>
6.     <link rel="stylesheet" type="text/css" href="
7.         /stylesheets/style.css" />
8. </head>
9. <body>
10.    <h1>Task 7.1</h1>
11.
12.    <button onclick="btn1Clicked()">Добавление машины</button>
13.    <button onclick="btn2Clicked()">Получение машины</button>
14.    <button onclick="btn3Clicked()">Добавление склада</button>
15.    <button onclick="btn4Clicked()">Получение склада</button>
16.
17.    <script>
18.        function btn1Clicked() {
19.            location.replace("http://localhost:5025/html/add_car.html")
20.        }
21.        function btn2Clicked() {
22.            location.replace("http://localhost:5025/html/get_car.html")
23.        }
24.        function btn3Clicked() {
25.            location.replace("http://localhost:5025/html/add_storage.html")
26.        }
27.        function btn4Clicked() {
28.            location.replace("http://localhost:5025/html/get_storage.html")
29.        }
30.    </script>
31.
32. </body>
33. </html>

```

/static/html/add_car.html

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4.   <meta charset="UTF-8">
5.   <title>Добавить машину</title>
6.   <link rel="stylesheet" type="text/css" href="
7.     /stylesheets/style.css" />
8. </head>
9. <body>
10.  <h1>Добавление машины</h1>
11.
12.  <p>Введите название машины</p>
13.  <input id="name_input" type="text" spellcheck="false" autocomplete="off">
14.
15.  <p>Введите стоимость машины</p>
16.  <input id="price_input" type="text" spellcheck="false" autocomplete="off">
17.
18.  <br><br>
19.
20.  <div id="add_btn" class="btn-class">Добавить машину</div>
21.
22.  <br><br>
23.
24.  <div id="back_btn" class="btn-class">Back</div>
25.
26.  <br><br>
27.
28.  <h5 id="result_label"></h5>
29.  <script src="/scripts/add_car.js"></script>
30. </body>
31. </html>
```

/static/html/get_car.html

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4.   <meta charset="UTF-8">
5.   <title>Найти машину</title>
6.   <link rel="stylesheet" type="text/css" href="
7.     /stylesheets/style.css" />
8. </head>
9. <body>
10.  <h1>Получение машины</h1>
11.
12.  <p>Введите название машины</p>
13.  <input id="name_input" type="text" spellcheck="false" autocomplete="off">
14.
15.  <br><br>
16.
17.  <div id="add_btn" class="btn-class">Найти машину</div>
18.
19.  <br><br>
20.
21.  <div id="back_btn" class="btn-class">Back</div>
22.
23.  <br><br>
24.
25.  <h5 id="result_label"></h5>
26.  <script src="/scripts/get_car.js"></script>
27. </body>
28. </html>
```

./static/html/add_storage.html

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4.   <meta charset="UTF-8">
5.   <title>Добавить склад</title>
6.   <link rel="stylesheet" type="text/css" href="
7.     /stylesheets/style.css" />
8. </head>
9. <body>
10.  <h1>Добавление склада</h1>
11.
12.  <p>Введите название склада</p>
13.  <input id="name_input" type="text" spellcheck="false" autocomplete="off">
14.
15.  <p>Введите список машин</p>
16.  <input id="cars_input" type="text" spellcheck="false" autocomplete="off">
17.
18.  <br><br>
19.
20.  <div id="add_btn" class="btn-class">Добавить склад</div>
21.
22.  <br><br>
23.
24.  <div id="back_btn" class="btn-class">Back</div>
25.
26.  <br><br>
27.
28.  <h5 id="result_label"></h5>
29.  <script src="/scripts/add_storage.js"></script>
30. </body>
31. </html>
```

./static/html/get_storage.html

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4.   <meta charset="UTF-8">
5.   <title>Найти склад</title>
6.   <link rel="stylesheet" type="text/css" href="
7.     /stylesheets/style.css" />
8. </head>
9. <body>
10.  <h1>Получение склада</h1>
11.
12.  <p>Введите название склада</p>
13.  <input id="name_input" type="text" spellcheck="false" autocomplete="off">
14.
15.  <br><br>
16.
17.  <div id="add_btn" class="btn-class">Найти склад</div>
18.
19.  <br><br>
20.
21.  <div id="back_btn" class="btn-class">Back</div>
22.
23.  <br><br>
24.
25.  <h5 id="result_label"></h5>
26.  <script src="/scripts/get_storage.js"></script>
27. </body>
28. </html>
```

./static/css/style.css

```
1. body {
2.     padding: 30px;
3.     background: burlywood;
4.     font-family: Geneva, Arial, Helvetica, sans-serif;
5. }
6.
7. .btn-class {
8.     padding: 6px;
9.     background-color: #4CAF50;
10.    color: white;
11.    cursor: pointer;
12.    display: inline-block;
13.    border-radius: 12px;
14. }
15.
16. button {
17.    background-color: #4CAF50;
18.    border: none;
19.    color: white;
20.    padding: 15px 32px;
21.    text-align: center;
22.    text-decoration: none;
23.    display: inline-block;
24.    font-size: 16px;
25.    border-radius: 12px;
26. }
```

http://localhost:5004/html/71C.html

Task 7.1

Добавление машины

Получение машины

Добавление склада

Получение склада

Добавление машины

Введите название машины

Введите стоимость машины

Добавить машину

Back

Машина **MyCar** с ценой **1000** добавлена!

Получение машины

Введите название машины

Найти машину

Back

Машина **MyCar** с ценой **1000**!

Добавление склада

Введите название склада

Введите список машин

Добавить склад

Back

Склад **K0** добавлена!

Получение склада

Введите название склада

Найти склад

Back

Склад **K0** с машинами **Car1, Car2, Car3** !

❖ Задание 2

Написать скрипт, который принимает на вход число и считает его факториал. Скрипт должен получать параметр через **process.argv**.

Написать скрипт, который принимает на вход массив чисел и выводит на экран факториал каждого числа из массива. Скрипт принимает параметры через **process.argv**.

При решении задачи вызывать скрипт вычисления факториала через **execSync**.

Код программы

Fact.js

```
1. "use strict";
2.
3. const nStr = "" + process.argv[2];
4. const nInt = parseInt(nStr);
5.
6. let s = 1;
7. for (let i = 1; i <= nInt; i++)
8.     s *= i;
9.
10. console.log("" + s);
```

Task72_1.js

```
1. "use strict";
2.
3. const execSync = require('child_process').execSync;
4.
5. function useCmd(s) {
6.     const options = {encoding: 'utf8'};
7.     const cmd = s.toString();
8.     const answer = execSync(cmd, options);
9.     return answer.toString();
10. }
11.
12.
13.
14. // получаем параметры скрипта
15. const nStr = process.argv[2];
16.
17. const factCommand = `node fact.js ${nStr}`;
18. let fact = useCmd(factCommand);
19. fact = parseInt(fact);
20. console.log(`Result: ${nStr}! = ${fact}`);
```

Task72_2.js

```
1. "use strict";
2.
3. const execSync = require('child_process').execSync;
4.
5. function useCmd(s) {
6.     const options = {encoding: 'utf8'};
7.     const cmd = s.toString();
8.     const answer = execSync(cmd, options);
9.     return answer.toString();
10. }
11.
12.
13. const len = process.argv.length;
14. var arr = [];
15. for (let i = 2; i < len; i++){
16.     const nStr = process.argv[i];
17.     const factCommand = `node fact.js ${nStr}`;
18.     let fact = useCmd(factCommand);
19.     fact = parseInt(fact);
20.     console.log(`${nStr}! = ${fact}`);
21. }
```

Результаты тестирования

```
PS E:\EVM_Ky_5\Lab_07> node .\Task72_1.js 5
Result: 5! = 120
PS E:\EVM_Ky_5\Lab_07>
```

```
PS E:\EVM_Ky_5\Lab_07> node .\Task72_2.js 1 2 3 4 5 6 7
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
```


❖ Задание 3

С клавиатуры считываются числа **A** и **B**. Необходимо вывести на экран все **числа Фибоначчи**, которые принадлежат отрезку от **A** до **B**.

Код программы

```
1. ok.
2.
3. writeNumber(X) :-
4.     X_NEW is X,
5.     write(X_NEW),
6.     write("\n").
7.
8. writetmp(X, A, B) :-
9.     X >= A,
10.    X <= B,
11.    writeNumber(X).
12.
13. writetmp(X, A, B):- ok.
14.
15. fibo(A, B, S) :-
16.    findfibo(S, Res),
17.    Res <= B,
18.    writetmp(Res, A, B),
19.    S_NEW is S + 1,
20.    fibo(A, B, S_NEW); ok.
21.
22.
23. fib(A, B) :- fibo(A, B, 0); ok.
24.
25. findfibo(0, 1) :- !.
26. findfibo(1, 1) :- !.
27. findfibo(N, F) :-
28.    N > 1,
29.    N1 is N-1,
30.    N2 is N-2,
31.    findfibo(N1, F1),
32.    findfibo(N2, F2),
33.    F is F1+F2.
34.
35. start :- write("Number 1: "), nl,
36.    read(A), nl,
37.    write("Number 2: "), nl,
38.    read(B), nl,
39.    fib(A, B), nl.
```

```
?- start.
Number 1:
|: 0.
```

```
Number 2:
|: 50.
```

```
1
1
2
3
5
8
13
21
34
```

```
true .
```

```
?- █
```

Результаты тестирования

❖ Задание 4

С клавиатуры считываются числа **A** и **B**. Необходимо вывести на экран все числа, **квадратный корень которых является целым числом**. При этом, необходимо вывести только числа, которые принадлежат отрезку от **A** до **B**.

Код программы

```
1. ok.
2.
3. writeNumber(X) :-
4.     X_NEW is X,
5.     write(X_NEW),
6.     write("\n").
7.
8. start :-
9.     write("Number 1: "), nl,
10.    read(A), nl,
11.    write("Number 2: "), nl,
12.    read(B), nl,
13.    A >= 0,
14.    B >= A,
15.    find(A, B), nl.
16.
17. find_square(A, B) :-
18.    Tmp is A * A,
19.    writeNumber(Tmp),
20.    A < B,
21.    A_NEW is A + 1,
22.    find_square(A_NEW, B); ok.
23.
24. find(A, B) :-
25.    A1 is ceiling(sqrt(A)),
26.    B1 is floor(sqrt(B)),
27.    find_square(A1, B1); ok.
28.
```

Результаты тестирования

```
?- start.
Number 1:
|: 0
|: .

Number 2:
|: 200.

0
1
4
9
16
25
36
49
64
81
100
121
144
169
196

true .
?- ■
```

❖ Задание 5

С клавиатуры считываются числа **A**, **B** и **C**. Необходимо вывести на экран все числа, которые принадлежат отрезку от **A** до **B** и делятся на **C** без остатка. Также надо вывести на экран **сумму** полученных чисел.

Код программы

```
1. ok.
2.
3. writeNumber(X) :-
4.   X_NEW is X,
5.   write(X_NEW),
6.   write("\n").
7.
8. writecmp(S, A, B) :-
9.   A =:= B,
10.  write("Sum = "),
11.  writeNumber(S).
12.
13. writecmp(S, A, B) :- ok.
14.
15. start :- write("Number 1: "), nl,
16.  read(A), nl,
17.  write("Number 2: "), nl,
18.  read(B), nl,
19.  write("Number 3: "), nl,
20.  read(C), nl,
21.
22.  find(A, B, C), nl.
23.
24. find_sum(A, B, C, SUM) :-
25.  Tmp is A * C,
26.  writeNumber(Tmp),
27.  SUM_NEW is SUM + Tmp,
28.  writecmp(SUM_NEW, A, B),
29.
30.  A_NEW is A + 1,
31.  A_NEW <= B,
32.  find_sum(A_NEW, B, C, SUM_NEW);ok.
33.
34. find(A, B, C) :-
35.  A1 is ceiling(A / C),
36.  B1 is floor(B / C),
37.  find_sum(A1, B1, C, 0); ok.
38.
39.
```

?- start.
Number 1:
|: 1.

Number 2:
|: 50.

Number 3:
|: 6.

Результаты тестирования

6
12
18
24
30
36
42
48
Sum = 216

true .
?- ■

