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 **09.03.01 Информатика и вычислительная техника**

О Т Ч Е Т

по лабораторной работе № 3

Дисциплина: Языки интернет-программирования

Студент ИУ6-33Б Д. И. Мироненко
(Группа) (Подпись, дата) (И.О. Фамилия)

Преподаватель В. Д. Шульман
(Подпись, дата) (И.О. Фамилия)

Москва, 2023

Задание

Написать Javascript-код для вывода дерева элементов страницы, с которой этот код запущен. Отступы для отображения формировать как символ

В процессе выполнения работы реализовать следующие пункты:

1. Сформировать страницу с произвольным кодом разметки, но обеспечить уровень вложенности внутри элемента <body> не менее 3.
2. Добавить внутри элемента <body> секцию , предназначенную для вывода результата обхода дерева элементов страницы.
3. Выбрать способ активации рекурсивной программы обхода дерева элементов, реализовать и подключить эту программу.
4. При проходе по узлам разметки обеспечить отладочный вывод в консоль. Привести в отчете содержимое консоли.
5. Реализовать вывод на странице.

В отчете привести код страницы с программой обхода, отладочный вывод в консоль и примеры обхода дерева элементов страницы.

Код программы

Файл “index.html”

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Lab-3</title>
    <link rel="preconnect" href="https://fonts.googleapis.com">
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
    <link
href="https://fonts.googleapis.com/css2?family=Inter:wght@400;700&family=JetBrains+Mono:wght@400;700&display=swap"
rel="stylesheet">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/css/bootstrap.min.css" rel="stylesheet"
integrity="sha384-4bw+/aepP/YC94hEpVNVgiZdgIC5+VKNBQNGCHeKRQN+PtmoHDEXuppvnDJzQIu9" crossorigin="anonymous">
    <script defer src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/js/bootstrap.bundle.min.js"
integrity="sha384-HwwvtgBNo3bZJJLYd8oVXjrBZt8cqVSpeBNS5n7C8IVInixGAoxmnlMuBnhbgrkm" crossorigin="anonymous"></script>
    <link rel="stylesheet" type="text/css" href="style.css">
    <script src="https://cdn.jsdelivr.net/gh/google/code-prettify@master/loader/run_prettify.js"></script>
  </head>
```

```
  <body>
    <nav class="navbar navbar-expand-lg bg-light shadow-sm p-3 mb-5" data-bs-theme="light">
      <div class="container-fluid">
        <a class="navbar-brand" href="index.html">lab-3</a>
        <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarNavDropdown"
aria-controls="navbarNavDropdown" aria-expanded="false" aria-label="Toggle navigation">
          <span class="navbar-toggler-icon"></span>
        </button>
        <div class="collapse navbar-collapse" id="navbarNavDropdown">
          <ul class="navbar-nav">
            <li class="nav-item">
              <a class="nav-link active" aria-current="page" href="../../lab-2/task-2/index.html">Schedule</a>
            </li>
            <li class="nav-item">
              <a class="nav-link" href="../../lab-1/task-1/index.html">Info</a>
            </li>
          </ul>
        </div>
      </div>
    </nav>
    <main class="container">
      <div id="main">
        <h1><code class="prettyprint lang-cpp">std::vector</code></h1>
        <div class="block">
          <h2>Intro</h2>
          <p>
            The elements are stored <strong>contiguously</strong>, which means
            that elements can be accessed <em>not only through iterators</em>,
            but also using <strong>offsets</strong> to regular pointers
            to elements. This means that a pointer to an element of a vector
            <em>may be passed to any function</em> that expects a pointer
            to an element of an array.
          </p>
          <p>
            The storage of the vector is handled <strong>automatically</strong>,
            being expanded as needed. Vectors usually occupy more space than static
            arrays, because more memory is allocated to handle future growth. This
            way a vector <em>does not need to reallocate each time an element is
            inserted</em>, but only when the additional memory is exhausted. The
            total amount of allocated memory can be queried using
            <a href="https://en.cppreference.com/w/cpp/container/vector/capacity">
              <code>capacity()</code>
            </a>
            function. Extra memory <em>can be returned</em> to the system via a call to
            <a href="https://en.cppreference.com/w/cpp/container/vector/shrink_to_fit">
              <code>shrink_to_fit()</code>
            </a>
            .
          </p>
          <p>
            Reallocations are usually costly operations in terms of performance.
            The
            <a href="https://en.cppreference.com/w/cpp/container/vector/reserve">
              <code>reserve()</code>
```

```
            </a>
            function can be used to eliminate reallocations if the number of
            elements is known beforehand.
          </p>
          <p>
            The <strong>complexity</strong> (efficiency) of common operations
            on vectors is as follows:
          </p>
          <ul>
            <li>
              Random access - <em>constant</em>
              <math>
                <mo>O</mo>
                <mo>(</mo>
                  <mn>1</mn>
                <mo>)</mo>
              </math>
            </li>
            <li>
              Insertion or removal of elements at the end - <em>amortized constant</em>
              <math>
                <mo>O</mo>
                <mo>(</mo>
                <mn>1</mn>
                <mo>)</mo>
              </math>
            </li>
            <li>
              Insertion or removal of elements - <em>linear in the distance to the end
                of the vector</em>
              <math>
                <mo>O</mo>
                <mo>(</mo>
                  <mn>n</mn>
                <mo>)</mo>
              </math>
            </li>
          </ul>
        </div>
        <div class="block">
          <h2>Example</h2>
          <code class="prettyprint lang-cpp">
            #include &lt;iostream&gt;<br>
            #include &lt;vector&gt;<br>
            <br>
            int main() {<br>
                    // Create a vector containing integers<br>
                    std::vector&lt;int&gt; v = {8, 4, 5, 9};<br>
            <br>
                    // Add two more integers to vector<br>
                    v.push_back(6);<br>
                    v.push_back(9);<br>
            <br>
                    // Overwrite element at position 2<br>
```

```
                    v[2] = -1;<br>
            <br>
                    // Print out the vector<br>
                    for (int n : v)<br>
                            std::cout &lt;&lt; n &lt;&lt; ' ';<br>
                    std::cout &lt;&lt; std::endl;<br>
            }<br>
          </code>
        </div>
        <div class="block">
          <h2>DOM Tree</h2>
          <button id="dom-tree-btn" type="button" class="btn btn-primary">Generate Tree</button>
          <div id="dom-tree-container"></div>
        </div>
      </div>
    </main>
    <footer class="py-3 my-2">
      <div class="container">
        <div class="row">
          <div class="col text-center text-dark footer-label">
            © 2023 Daniil Mironenko
          </div>
        </div>
      </div>
    </footer>
    <script src="./dom_tree.js"></script>
  </body>
</html>
```

Файл “style.css”

```
body {
    background-color: #f2f1f6;
    color: #010100;
    font-size: 14px;
    font-family: 'Inter', sans-serif;
}

.navbar {
    -webkit-box-shadow: 0 8px 6px -6px #999;
    -moz-box-shadow: 0 8px 6px -6px #999;
    box-shadow: 0 8px 6px -6px #999;

    margin-bottom: 50px;
}

div#main {
    width: 50%;
    margin: auto;
}

div.block {
```

```
    margin-bottom: 40px;
    padding: 5px 40px 20px 40px;
    background-color: #ffffff;
    border-radius: 15px;
    text-align: justify;
    line-height: 1.5;
    font-size: 17px;

    -webkit-box-shadow: 3px 1px 14px 1px rgba(34, 60, 80, 0.1);
    -moz-box-shadow: 3px 1px 14px 1px rgba(34, 60, 80, 0.1);
    box-shadow: 3px 1px 14px 1px rgba(34, 60, 80, 0.1);
}

h1, h2, h3, h4, h5, h6 {
    font-weight: bold;
}

h2 {
    margin-top: 20px;
}

a {
    text-decoration: none;
}

code {
    font-family: 'JetBrains Mono', monospace;
    line-height: 0.5;
}

math {
    font-family: serif;
    font-style: italic;
    letter-spacing: 0px;
}
```

Файл “dom_tree.js”

```
const TAB_SIZE = 2;



function DOMTree(node = document.body, depth = 0) {
  if (node.tagName !== undefined) {
    console.log(" ".repeat(TAB_SIZE * depth) + node.tagName)

    p = document.createElement("p");
    p.innerHTML = " ".repeat(TAB_SIZE * depth) + node.tagName;
    DOMTreeContainer.appendChild(p);

    node.childNodes.forEach((child) => DOMTree(child, depth + 1));
  }
};
```

```
var DOMTreeBtn = document.getElementById("dom-tree-btn");
var DOMTreeContainer = document.getElementById("dom-tree-container");

DOMTreeBtn.addEventListener("click", () => {
  DOMTree();
});
```

Работа сайта

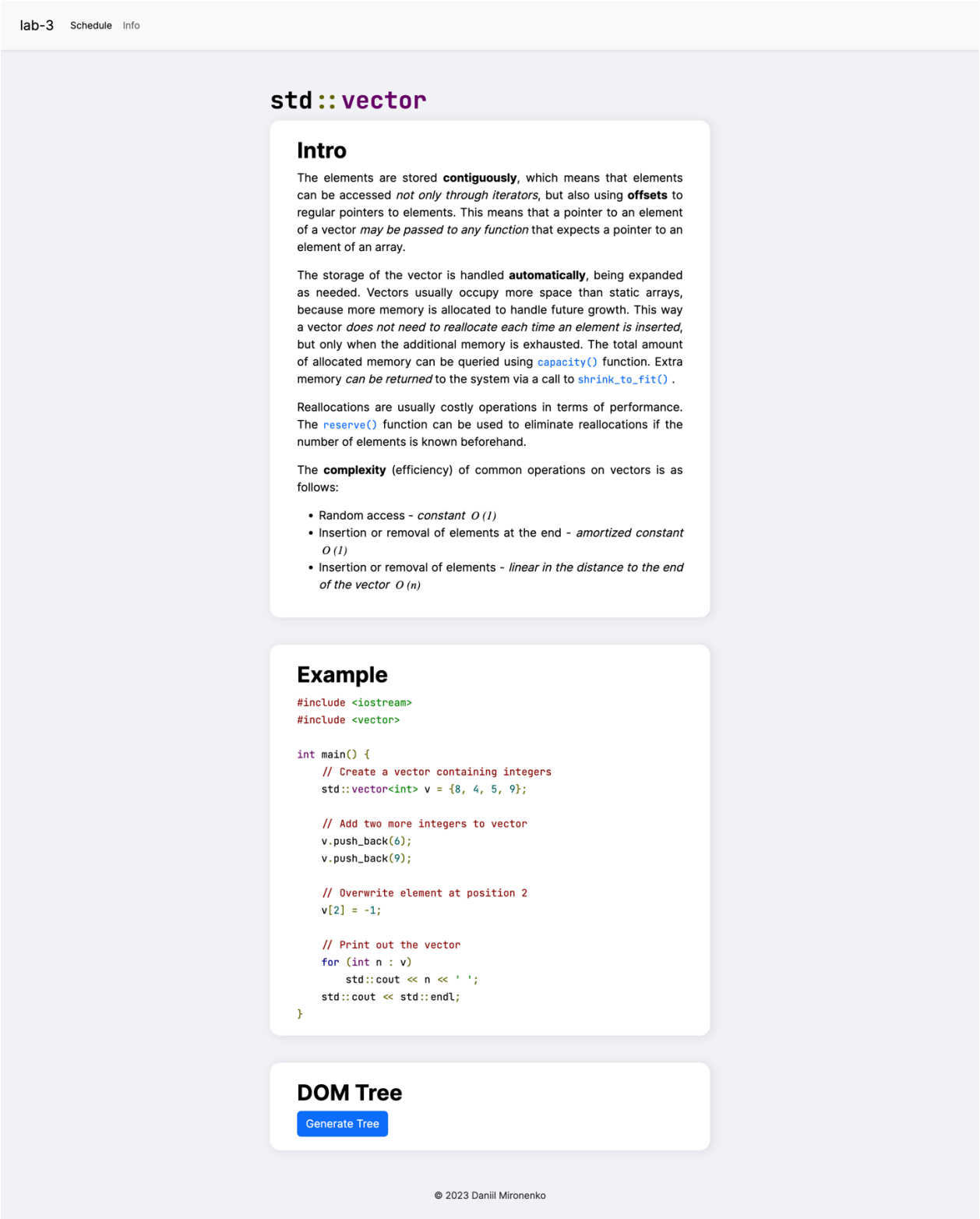

Рисунок 1

lab-3 Schedule Info

std::vector

Intro

The elements are stored **contiguously**, which means that elements can be accessed *not only through iterators*, but also using **offsets** to regular pointers to elements. This means that a pointer to an element of a vector *may be passed to any function* that expects a pointer to an element of an array.

The storage of the vector is handled **automatically**, being expanded as needed. Vectors usually occupy more space than static arrays, because more memory is allocated to handle future growth. This way a vector *does not need to reallocate each time an element is inserted*, but only when the additional memory is exhausted. The total amount of allocated memory can be queried using `capacity()` function. Extra memory *can be returned* to the system via a call to `shrink_to_fit()` .

Reallocations are usually costly operations in terms of performance. The `reserve()` function can be used to eliminate reallocations if the number of elements is known beforehand.

The **complexity** (efficiency) of common operations on vectors is as follows:

- Random access - *constant* $O(1)$
- Insertion or removal of elements at the end - *amortized constant* $O(1)$
- Insertion or removal of elements - *linear in the distance to the end of the vector* $O(n)$

Example

```
#include <iostream>
#include <vector>

int main() {
    // Create a vector containing integers
    std::vector<int> v = {8, 4, 5, 9};

    // Add two more integers to vector
    v.push_back(6);
    v.push_back(9);

    // Overwrite element at position 2
    v[2] = -1;

    // Print out the vector
    for (int n : v)
        std::cout << n << ' ';
    std::cout << std::endl;
}
```

DOM Tree

Generate Tree

BODY
NAV
DIV
A
BUTTON
SPAN
DIV
UL
LI
A
LI
A
MAIN
DIV
H1
CODE
SPAN
SPAN

Рисунок 2

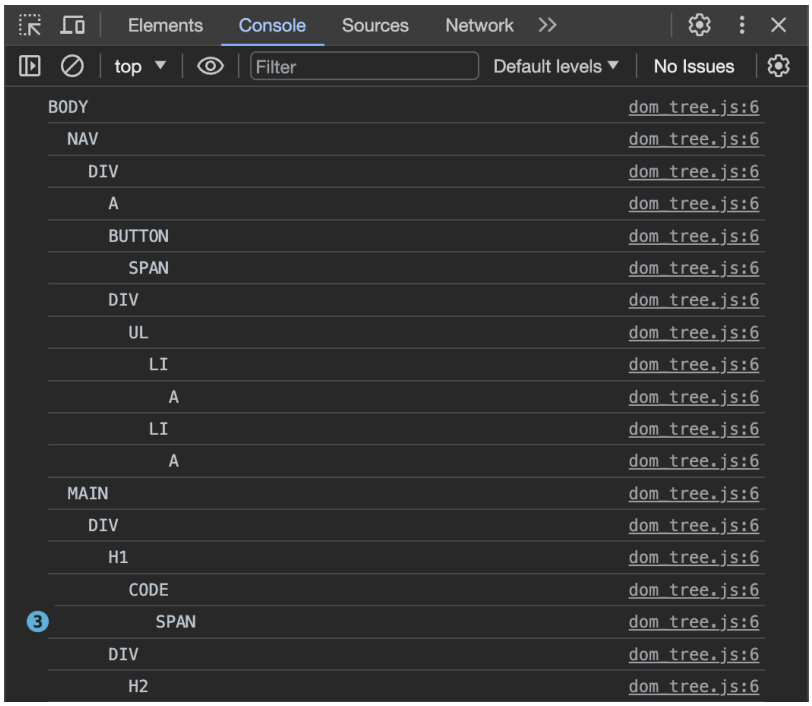

Рисунок 3

Проверка валидатором

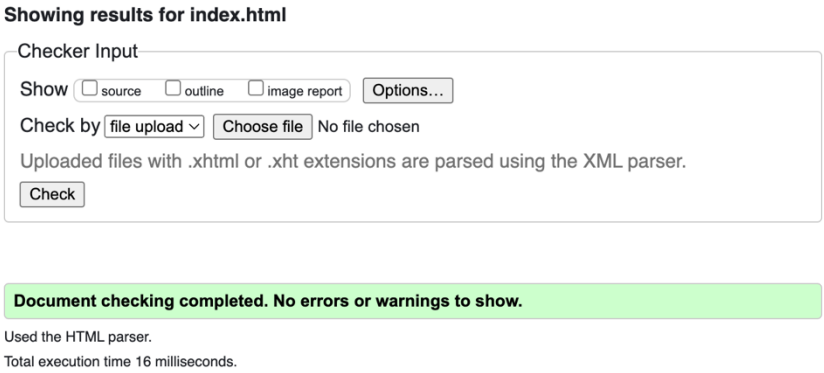

Рисунок 4

Вывод

Изучены основы языка программирования javascript, а также технология DOM для взаимодействия с элементами страницы; реализована страница, где были использованы вышесказанные технологии; выполнена проверка валидатором веб-страницы.