Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Методы трансляции

ОТЧЁТ

к лабораторной работе №2

на тему

ЛЕКСИЧЕСКИЙ АНАЛИЗ

Выполнил студент гр.153502 Толстой Д.В.

Проверил ассистент кафедры информатики Гриценко Н.Ю.

Минск 2024

**СОДЕРЖАНИЕ**

[1 Формулировка задачи 3](#_Toc158758016)

[2 Описание функций программы 4](#_Toc158758017)

[Приложение А (обязательное) Листинг исходного кода 6](#_Toc158758018)

1 ФОРМУЛИРОВКА ЗАДАЧИ

Целью выполнения лабораторной работы является разработка лексического анализатора подмножества языка программирования, определенного в лабораторной работе 1. Определяются лексические правила. Выполняется перевод потока символов в поток лексем.

В качестве задачи необходимо определить лексические правила, выполнить перевод потока символов в поток лексем, а также показать скриншоты нахождения четырех лексических ошибок разработанным анализатором.

2 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

Программа реализует перевод потока символов в поток лексем. При успешном выполнении программа выводит в консоль поток лексем. Вывод успешно выполненной программы предоставлен на рисунке 1.

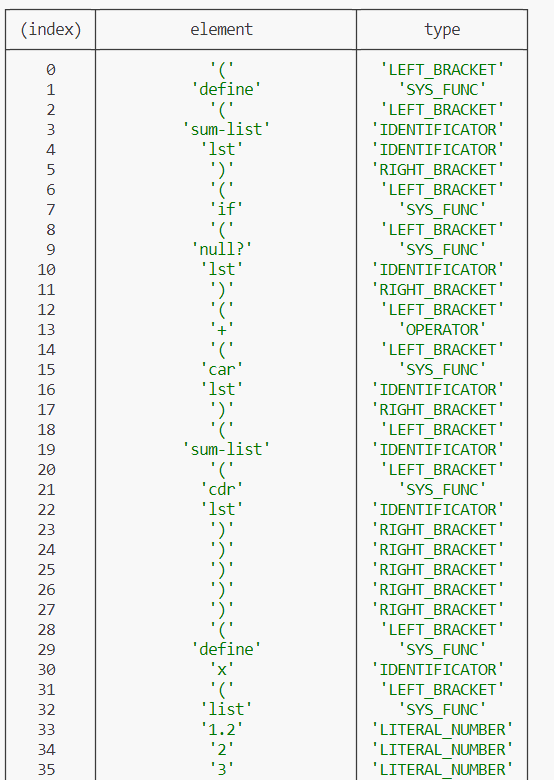


Рисунок 1 – Вывод программы

На рисунке 2 предоставлена реакция программы на неправильное написание числа.

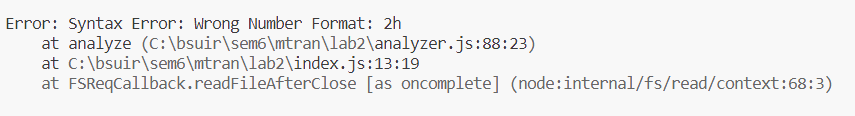


Рисунок 2 – Реакция программы на неправильное написание числа

На рисунке 3 предоставлена реакция программы на неправильное написание логического значения.

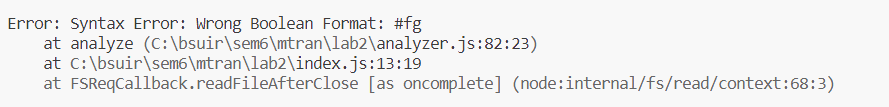


Рисунок 3 – Реакция программы на неправильное написание логического значения

На рисунке 4 предоставлена реакция программы на незакрытую строку.

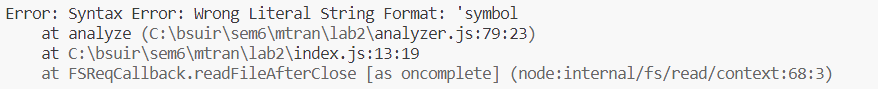


Рисунок 4 – Реакция программы на незакрытую строку

На рисунке 5 предоставлена реакция программы на неожидаемый символ в названии переменной.

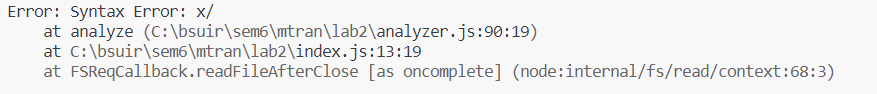


Рисунок 5 – Реакция программы на неожидаемый символ в названии переменной

Таким образом в ходе лабораторной работы был разработан лексический анализатор, который способен выводить в консоль таблицу имен, поток лексем, а также найденные лексические ошибки.

ПРИЛОЖЕНИЕ А

**(обязательное)**

**Листинг исходного кода**

Файл index.js

const spaceText = require('./spacing');

const fs = require('fs');

const analyze = require('./analyzer');

fs.readFile('code.txt', 'utf8', (err, data) => {

    if (err) {

      console.error(err);

      return;

    }

    const spacedText = spaceText(data);

    listOfElements = splitIgnoringQuotes(spacedText, ' ');

    console.table(analyze(listOfElements));

  });

  function splitIgnoringQuotes(str, delimiter) {

    let elements = [];

    let currentElement = '';

    let insideQuotes = false;

    for (let i = 0; i < str.length; i++) {

        const char = str[i];

        if (char === '"') {

            insideQuotes = !insideQuotes;

        }

        if (char === delimiter && !insideQuotes) {

            elements.push(currentElement);

            currentElement = '';

        } else {

            currentElement += char;

        }

    }

    elements.push(currentElement); // Добавляем последний элемент

    return elements;

}

Файл service.js

function isValidIdentifier(identifier) {

    // Проверяем, что идентификатор не пустой

    if (identifier.length === 0) {

        return false;

    }

    // Проверяем, что первый символ является буквой

    const firstChar = identifier[0];

    if (!((firstChar >= 'a' && firstChar <= 'z') || (firstChar >= 'A' && firstChar <= 'Z'))) {

        return false;

    }

    // Проверяем каждый символ идентификатора

    for (let i = 1; i < identifier.length; i++) {

        const char = identifier[i];

        // Разрешаем буквы, цифры, дефисы и знаки подчеркивания

        if (!((char >= 'a' && char <= 'z') || (char >= 'A' && char <= 'Z') || (char >= '0' && char <= '9') || char === '-' || char === '\_')) {

            return false;

        }

    }

    // Идентификатор прошел все проверки

    return true;

}

module.exports = isValidIdentifier;

Файл spacing.js

function spaceText(text) {

    const brackets = ['(', ')', '[', ']', '{', '}'];

    // Преобразуем строку в массив символов

    const characters = text.split('');

    // Проходим по каждому символу

    for (let i = 0; i < characters.length; i++) {

        const char = characters[i];

        // Если текущий символ является скобкой

        if (brackets.includes(char)) {

            // Проверяем, есть ли пробел перед текущей скобкой

            if (i > 0 && characters[i - 1] !== ' ') {

                // Вставляем пробел перед скобкой

                characters.splice(i, 0, ' ');

                i++; // Увеличиваем индекс, чтобы пропустить вставленный пробел

            }

            // Проверяем, есть ли пробел после текущей скобки

            if (i < characters.length - 1 && characters[i + 1] !== ' ') {

                // Вставляем пробел после скобки

                characters.splice(i + 1, 0, ' ');

                i++; // Увеличиваем индекс, чтобы пропустить вставленный пробел

            }

        }

    }

    // Преобразуем массив обратно в строку

    return characters.join('');

}

module.exports = spaceText;

Файл analyzer.js

const test = require('./service');

function analyze(listOfElements) {

    const answer = [];

    normal\_answer = [];

    const categoryMappings = {

        '(': 'LEFT\_BRACKET',

        ')': 'RIGHT\_BRACKET',

        'nil': 'CONSTANT',

        't': 'CONSTANT',

        '#t': 'CONSTANT',

        '#f': 'CONSTANT',

        'defun': 'SYS\_FUNC',

        'define': 'SYS\_FUNC',

        'null?': 'SYS\_FUNC',

        'list?': 'SYS\_FUNC',

        'display': 'SYS\_FUNC',

        'eval': 'SYS\_FUNC',

        'cond': 'SYS\_FUNC',

        'if': 'SYS\_FUNC',

        'newline': 'SYS\_FUNC',

        'quote': 'SYS\_FUNC',

        'list': 'SYS\_FUNC',

        'cons': 'SYS\_FUNC',

        'cdr': 'SYS\_FUNC',

        'car': 'SYS\_FUNC',

        'typep': 'SYS\_FUNC',

        'else': 'KEYWORD',

        'number': 'KEYWORD',

        'rational': 'KEYWORD',

        'float': 'KEYWORD',

        'complex': 'KEYWORD',

        'integer': 'KEYWORD',

        'sym': 'KEYWORD',

        'ratio': 'KEYWORD',

        'fixnum': 'KEYWORD',

        'bignum': 'KEYWORD',

        'short-float': 'KEYWORD',

        'single-float': 'KEYWORD',

        'double-float': 'KEYWORD',

        'long-float': 'KEYWORD',

        'lambda': 'KEYWORD',

        '+': 'OPERATOR',

        '-': 'OPERATOR',

        '/': 'OPERATOR',

        '\*': 'OPERATOR',

        '=': 'OPERATOR',

        '<=': 'OPERATOR',

        '>=': 'OPERATOR',

        '<': 'OPERATOR',

        '>': 'OPERATOR',

    };

    for (element of listOfElements) {

        if (element.includes('\r') || element.includes('\n') || element === '') {

            continue;

        } else if (categoryMappings[element]) {

            answer.push({'element': element, 'type': categoryMappings[element]});

            normal\_answer.push({element: element, type: element, type: categoryMappings[element]});

        } else if (!isNaN(Number(element))) {

            answer.push({'element': element, 'type': 'LITERAL\_NUMBER'});

            normal\_answer.push({element: element, type: 'LITERAL\_NUMBER'});

        } else if ((element[0] === '\'' || element[0] === '"') && (element[element.length - 1] === '\'' || element[element.length - 1] === '"')) {

            answer.push({'element': element, 'type': 'LITERAL\_STRING'});

            normal\_answer.push({element: element, type: 'LITERAL\_STRING'});

        } else if (element[0] === ':') {

            answer.push({'element': element, 'type': 'CONSTANT'}); // Символы, начинающиеся с двоеточия

            normal\_answer.push({element: element, type: 'CONSTANT'});

        } else if (element.startsWith('macro')) {

            answer.push({'element': element, 'type': 'MACRO'}); // Макросы

            normal\_answer.push({element: element, type: 'MACRO'});

        } else if (element.startsWith(';')) {

            answer.push({'element': element, 'type': 'COMMENT'}); // Комментарии

            normal\_answer.push({element: element, type: 'COMMENT'});

        } else if (test(element) && answer[element] === undefined) {

            answer.push({'element': element, 'type': 'IDENTIFICATOR'});

            normal\_answer.push({element: element, type: 'IDENTIFICATOR'});

        } else if (answer[element] === undefined) {

            if (element.includes('\'') || element.includes("\"")) {

                throw Error("Syntax Error: Wrong Literal String Format: " + element);

            }

            if (element.includes('#')) {

                throw Error("Syntax Error: Wrong Boolean Format: " + element);

            }

            if (element.includes('\\')) {

                throw Error("Syntax Error: Wrong Variable Name Format: " + element);

            }

            if (!((element[0] >= 'a' && element[0] <= 'z') || (element[0] >= 'A' && element[0] <= 'Z'))) {

                throw Error("Syntax Error: Wrong Number Format: " + element);

            }

            throw Error("Syntax Error: " + element);

        }

    }

    return answer;

}

module.exports = analyze;