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 №4

на тему

СЕМАНТИЧЕСКИЙ АНАЛИЗАТОР

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 семантического анализатора подмножества языка программирования, определенного в лабораторной работе 1. В процессе семантического анализа проверяется наличие семантических ошибок в исходной программе и накапливается информация о типах для следующей стадии – генерации кода. При семантическом анализе используются иерархические структуры, полученные во время синтаксического анализа для идентификации операторов и операндов выражений и инструкций.

2 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

Программа проверяет наличие инициализации используемых переменных в текущей области видимости. Код участка кода в программе предоставлен на рисунке 1.

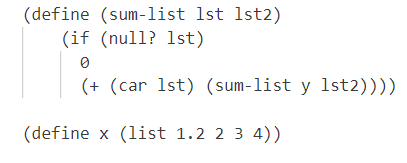


Рисунок 1 – Код неправильного участка кода

Вывод ошибки при использовании неинициализированной переменной предоставлен на рисунке 2.



Рисунок 2 – Вывод ошибки

На рисунке 3 предоставлен участок кода с сложением переменных с разными типами данных.

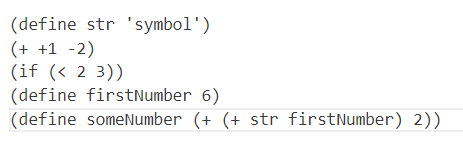


Рисунок 3 – Участок кода с сложением переменных с разными типами данных

На рисунке 4 предоставлена реакция программы на сложение переменных с разными типами.

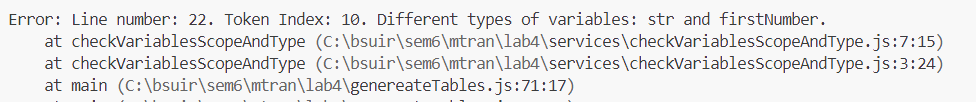


Рисунок 4 – Реакция программы на сложение переменных с разными типами

На рисунке 5 предоставлен участок кода с использованием функции с неверным количеством переменных.

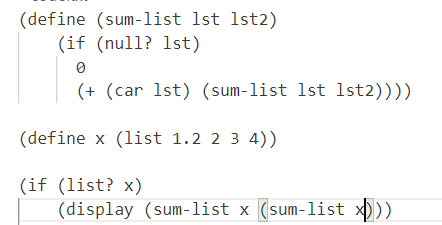


Рисунок 5 – Участок кода с сложением переменных с разными типами данных

На рисунке 6 предоставлена реакция программы на использование функции с неверным количеством переменных.

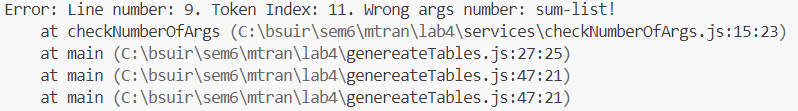


Рисунок 6 – Реакция программы на использование функции с неверным количеством переменных

ПРИЛОЖЕНИЕ А

**(обязательное)**

**Листинг исходного кода**

Файл index.js

const tokenize = require('../lab2/analyzer');

const buildSyntaxTree = require('../lab3/syntaxAnalyzer');

const generateTables = require('./genereateTables');

const fs = require('fs');

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

if (err) {

console.error(err);

return;

}

let [answer, normalAnswer] = tokenize(data);

let ast = buildSyntaxTree(normalAnswer);

generateTables(ast);

//checkScopes(ast);

fs.writeFile('tree.txt', JSON.stringify(ast, null, '\t'), function(error){

if(error){

return console.log(error);

}

console.log("Файл успешно записан");

});

});

Файл generateTables.js

const findFirstVariable = require('./services/findFirstVariable');

const findFirstLiteral = require('./services/findFirstLiteral');

const checkLiteralTypes = require('./services/checkLiteralTypes');

const checkVariablesScopeAndType = require('./services/checkVariablesScopeAndType');

const checkNumberOfArgs = require('./services/checkNumberOfArgs');

const checkVariableScope = require('./services/checkVariableScope');

function generateTables(node) {

let stackOfTables = [];

function main(node) {

stackOfTables.push([]);

if (node.type === 'Program') {

main(node.body);

}

if (node.name !== 'define') {

if (node.type === 'CallExpression') {

let variableFound = null;

for (let scopeIndex = stackOfTables.length - 1; scopeIndex >= 0; scopeIndex--) {

const scope = stackOfTables[scopeIndex];

for (const variable of scope) {

if (variable.name === node.name) {

variableFound = true;

break;

}

}

if (variableFound) {

checkNumberOfArgs(node, stackOfTables);

}

}

}

if (node.type === 'Operator') {

if (!checkLiteralTypes(node.params[1])) {

throw new Error(`Line number: ${node.lineNumber}. Token Index: ${node.tokenIndex}. Different types of literals!`)

}

checkVariablesScopeAndType(node.params[1], stackOfTables);

checkVariableScope(node, stackOfTables);

}

if (node.body !== undefined) {

for (let expr of node.body) {

main(expr);

}

}

if (node.params === undefined) {

return;

} else if (Array.isArray(node.params)) {

for (let param of node.params) {

main(param);

}

} else {

main(node.params);

}

} else {

if (node.params.length !== 2) {

throw new Error(`Line number: ${node.lineNumber}. Token Index: ${node.tokenIndex}. Wrong definition of variable or function!`)

}

if (node.params[0].type !== 'CallExpression' && node.params[0].type !== 'Variable') {

throw new Error(`Semantic error. Line number: ${node.lineNumber}. Token Index: ${node.tokenIndex}. Can't define not variable or function: ${node.params[0].name}`)

}

let variable;

variable = {name: node.params[0].name};

if (node.params[0].type === 'CallExpression') {

variable.type = 'Function';

variable.numberOfArgs = node.params[0].params.length;

for (let param of node.params[0].params) {

stackOfTables[stackOfTables.length - 2].push({name: param.name, type: 'Any'});

}

} else if (node.params[1].type === 'Operator') {

if (!checkLiteralTypes(node.params[1])) {

throw new Error(`Line number: ${node.lineNumber}. Token Index: ${node.tokenIndex}. Different types of literals!`)

}

checkVariablesScopeAndType(node.params[1], stackOfTables);

let variableOrNull = findFirstVariable(node.params[1]);

if (variableOrNull !== null) {

let index = stackOfTables.length - 1;

let founded = false;

for (; index >= 0; index--) {

for (let j = 0; j < stackOfTables[index].length; j++) {

if (stackOfTables[index][j].name === variableOrNull) {

variable.type = stackOfTables[index].type;

founded = true;

}

}

}

if (!founded) {

throw new Error(`Line number: ${node.lineNumber}. Token Index: ${node.tokenIndex}. Undefined variable: ${variableOrNull}!`)

}

} else {

let literal = findFirstLiteral(node.params[1]);

if (!isNaN(Number(literal))) {

variable.type = 'Number';

variable.value = node.params[1].value;

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

variable.type = 'String';

} else {

variable.type = 'Boolean';

}

}

} else if (node.params[1].name === 'list') {

variable.type = 'List';

} else if (node.params[1].value === 't' || node.params[1].value === 'nil' || node.params[1].value === '#t' || node.params[1].value === '#f') {

variable.type = 'Boolean';

variable.value = node.params[1].value;

} else if (!isNaN(Number(node.params[1].value))) {

variable.type = 'Number';

variable.value = node.params[1].value;

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

variable.type = 'String';

}

stackOfTables[stackOfTables.length - 2].push(variable);

main(node.params[1]);

}

if (stackOfTables.length !== 0) {

if (stackOfTables[stackOfTables.length - 1].length !== 0) {

node.tableOfVariables = stackOfTables.pop();

}

else {

stackOfTables.pop();

}

}

return;

}

main(node);

}

module.exports = generateTables;