

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«Московский технологический университет»**

**МГУПИ**

Институт комплексной безопасности и специального приборостроения

**Кафедра КБ-3 Управления и моделирования систем**

**Разработка проблемно-ориентированных транслирующих средств**

**Отчет**

Студент 1 курса

заочного отделения

кафедра КБ-3

направление 09.04.04

Матросов Кирилл Дмитриевич

Преподаватель каф. КБ-3

«Управления и моделирования систем»:

Корягин С.В.

2017

Содержание

# Введение

Целью работы является реализовать транслятор по БНФ форме.

Трансляция – преобразование программы, представленной на одном из языков программирования, в программу на другом языке. Транслятор обычно выполняет также диагностику ошибок, формирует словари идентификаторов, выдает для печати тексты программы и т.д.

Этапы трансляции:

1. Лексический анализ – процесс аналитического разбора входной последовательности символов (например, код на одном из языков программирования) с целью получения на выходе последовательности символов, называемых «токенами» (подобно группировке букв в словах). Группа символов входной последовательности, идентифицируемая на выходе процесса как токен называется лексемой. В процессе лексического анализа производится распознавание и выделение лексем из входной последовательности символов.  
   Токен можно представить в виде структуры, содержащий идентификатор токена и если нужно, последовательность символов лексемы, выделенной из входного потока (строка, число и т.д.).
2. Синтаксический анализ (парсинг) – это процесс сопоставления последовательности лексем (слов, токенов) языка с его формальной грамматикой результатом обычно является дерево разбора (синтаксическое дерево). Обычно применяется совместно с лексическим анализом.
3. Преобразование дерева разбора в другой язык программирования, либо интерпретирование дерева разбора.

# 1. Цель работы

По заданной БНФ разработать синтаксически-управляемый транслятор, который будет выполнять следующие функции:

1. Анализ входного текста и определение его принадлежности к данной БНФ.
2. Определение места и типа первой ошибки во входном тексте.
3. Расчет выражений и вывод значений переменных.

# 2. Форма Бэкуса-Наура

Lan = "Program Defaults Interval Step Method Operations"

Defaults = "parametrs" var "=" num | "values" var "=" num

Interval = "interval" [real, real]

Step = "step" num

Method = "method" "eiler"|"runge2"|"runge4"

Operation = label "=" expression

expression = part <maths part>

part = expression | func"("var|num|part")

maths = a&s|m&d|exp

exp = "^"

m&d = "\*", "/"

a&s = "+", "-"

funcs = "cos"|"sin"

num = int|real

var = l<symbol\*>

label = symbol+"'"

symbol = l|n

real = int"."int

int = n...n

n = ["0"-"9"]

l = ["a"-"z"]

# 3. Список ошибок

1. Программа должна начинаться с оператора “Program”.
2. Отсутствуют дифференциальные уравнения.
3. Отсутствуют значения по умолчанию.
4. Оператор «parametrs» должен быть указан до оператора «values».
5. Оператор «values» должен быть указан до оператора «interval».
6. Оператор «interval» должен быть указан до оператора «step».
7. Оператор «step» должен быть указан до оператора «method».
8. Отсутствует значение по интервалу/шагу.
9. Правая часть в блоке «parametrs» должна быть числом.
10. За оператором «method» должен быть указан из методов: eiler, runge2, runge4
11. Пропущен знак «=».
12. Дублируется знак «=».
13. Не число
14. Количество открывающих скобок не соответствует количеству закрывающих скобок
15. Нет левой части
16. Нет правой части
17. Знак после открывающей скобки
18. Знак до закрывающей скобки
19. Повторяющий знак подряд n раз
20. Знак в конце строки

# 4. Блок-схемы

**Lan = "Program Defaults Interval Step Method Operations"**

|  |  |
| --- | --- |
|  | this.text = this.text.split('\n').filter(line => line !== '').join('\n');  this.text = this.text.split('\n');  for (let i = 0, len = this.text.length; i < len; i++) {  this.text[i] = this.text[i].trim();  }  this.text = this.text.join('\n');  const index = this.text.search('Program');  if (index === -1) {  throw new this.exceptions.programBlock()  }  let last\_block = {};  ['parametrs', 'values', 'interval', 'step', 'method'].forEach((block, i) => {  const index = this.text.search(block);  if (index === -1) {  throw new this.exceptions.missBlock(block)  }  if (i && index < last\_block.index) {  throw new this.exceptions.orderBlock(block, last\_block.block)  }  last\_block = {block, index};  });  this.getFunctions(this.text.split(/method.\*\n/gi)[1]);  this.getStep(this.text.match(/step\s\*(\d+\.?\d\*)\n/));  this.getInterval(this.text.match(/interval.\*\[(.+),(.+)\]\n/));  this.getMethod(this.text.match(/method(.\*)/));  this.build(); |

**Operation = label "=" expression (описание дифференциального уравнения)**

|  |  |
| --- | --- |
|  | if (!text)  {  throw new this.exceptions.noLines('functions');  }  let fs = text.trim().split('\n');  let \_\_functions = [];  fs.forEach((func) => {  this.checkSigns(func, '=');  let parts = func.split('=');  const name = parts[0].split(this.exprSign)[0].trim(), f = parts[1].trim();  if (name === '') {  throw new this.exceptions.noPart(func);  } else if (f === '') {  throw new this.exceptions.noPart(func, true);  }  else {  \_\_functions.push({name, f});  }  });  if (\_\_functions.length === 0)  {  throw new this.exceptions.noLines('functions');  }  \_\_functions.forEach(this.checkExpression);  this.functions = \_\_functions; |

**Step = "step" num**

|  |  |
| --- | --- |
|  | this.getStep(this.text.match(/step\s\*(\d+\.?\d\*)\n/));  getStep(text){  if (!text)  {  throw new this.exceptions.missValue('step');  }  this.isNumber(text ? text[1] : NaN);  this.step = parseFloat(text[1]);  } |

**Interval = "interval" [real, real]**

|  |  |
| --- | --- |
|  | this.getInterval(this.text.match(/interval.\*\[(.+),(.+)\]\n/));  getInterval(text){    if (!text)  {  throw new this.exceptions.missValue('interval');  }  this.isNumber(text[1]);  this.isNumber(text[2]);  this.interval = [parseFloat(text[1]), parseFloat(text[2])];  } |

**Method = "method" "eiler"|"runge2"|"runge4"**

|  |  |
| --- | --- |
|  | const METHODS = ['eiler', 'runge2', 'runge4'];  this.method = text[1].trim().toLowerCase();  if (!METHODS.includes  (this.method)) {  throw new this.exceptions.  noMethod(this.method);  } |

**"parametrs" var "=" num**

|  |  |
| --- | --- |
|  | this.getParams(this.text.match(/parametrs(\n.\*)+values/));  getParams(ps){  ps = ps[0].split('parametrs\n')[1];  ps = ps.split('\nvalues')[0];  ps = ps.split('\n');  ps.forEach((\_ps) => {  this.checkSigns(\_ps, '=');  const p = \_ps.split('='), left = p[0].trim(), right = p[1].trim();  if (left === '') {  throw new this.exceptions.noPart(\_ps);  } else if (right === '') {  throw new this.exceptions.noPart(\_ps, true);  }  else {  this.isNumber(right);  this.params[left] = parseFloat(right);  }  })} |

**"values" var "=" num**

|  |  |
| --- | --- |
|  | let values = this.text.split('values\n')[1].split('inter')[0].split('\n').filter(v => v !== '');  this.getDefaults(values);  getDefaults(ds){    if (ds.length === 0)  {  throw new this.exceptions.noLines('values');  }  if (this.functions.length !== ds.length) {  throw new this.exceptions.countDefault();  }  for (let i = 0, len = ds.length; i < len; i++) {  this.checkSigns(ds[i], '=');  let d = ds[i].split('='), left = d[0].trim(), right = d[1].trim();  let func = this.functions.find(f => f.name === left);  if (func) {  if (left === '') {  throw new this.exceptions.noPart(ds);  } else if (right === '') {  throw new this.exceptions.noPart(ds, true);  }  else {  this.isNumber(right);  func.def = parseFloat(right);  console.log(func.def)  }  }  }  } |

**label = symbol+"'"**

|  |  |
| --- | --- |
|  | let parts = func.split('=');  const name = parts[0].split(this.exprSign)[0].trim(), f = parts[1].trim();  if (name === '') {  throw new this.exceptions.noPart(func);  } else if (f === '') {  throw new this.exceptions.noPart(func, true);  } |

**a&s = "+", "-"**

|  |  |
| --- | --- |
|  | function *parseAddSubtract* (res, token, last) { var node, operators, name, fn, params;  node = *parseMultiplyDivide*();  operators = { '+': 'add', '-': 'subtract' }; while (operators.hasOwnProperty(token)) { name = token; fn = operators[name];  *getTokenSkipNewline*(); params = [node, *parseMultiplyDivide*()]; node = new OperatorNode(name, fn, params); }  res.calc(node, last); } |

**m&d = "\*", "/"**

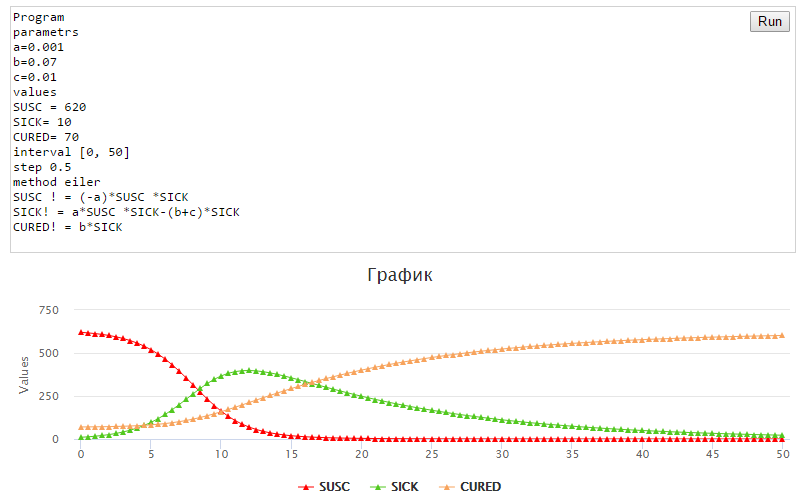
|  |  |
| --- | --- |
|  | function parseMultiplyDivide (node) { var last, operators, name, fn;  last = node;  operators = { '\*': 'multiply',  '/': 'divide' };  while (true) { if (operators.hasOwnProperty(token)) { name = token; fn = operators[name];  getTokenSkipNewline();  last = parseUnary(); node = new OperatorNode(name, fn, [node, last]); } else if ((token\_type == TOKENTYPE.SYMBOL) || (token == 'in' && (node && node.isConstantNode)) || (token\_type == TOKENTYPE.NUMBER && !last.isConstantNode && (!last.isOperatorNode || last.op === '!')) || (token == '(')) {  last = parseUnary(); node = new OperatorNode('\*', 'multiply', [node, last], true ); } else { break; } }  return node; } |

# 5. Результаты работы программы

1. Распространение инфекционного заболевания (эпидемии), где, для восстановления иммунитета, мы учитываются три переменные состояния:

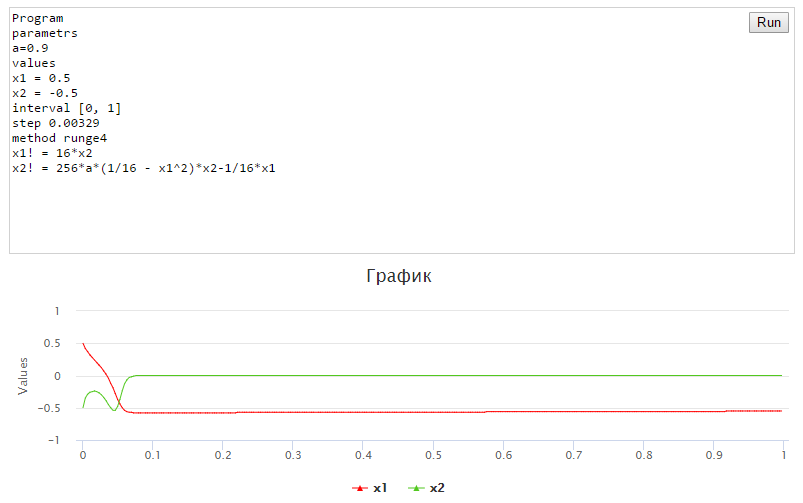
SUSC – люди, которые могут заболеть;

SICK – больные в данный момент;

CURED – люди, перенёсшие болезнь; 

2. Система дифференциальных уравнений, описывающая решение уравнения Ван-дер-Поля.

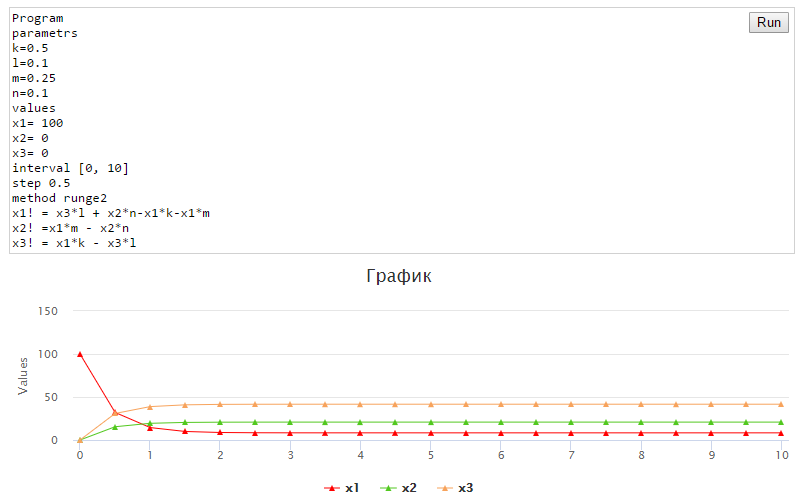
|  |  |
| --- | --- |
|  |  |



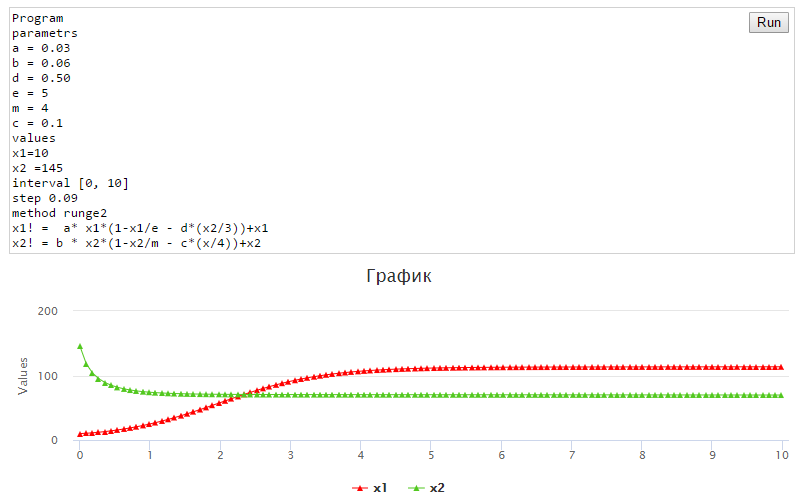
3. Система дифференциальных уравнений, описывающая кинетику химического процесса при изотермических условиях.

Данная система уравнений имеет следующий вид:

|  |  |
| --- | --- |
|  |  |

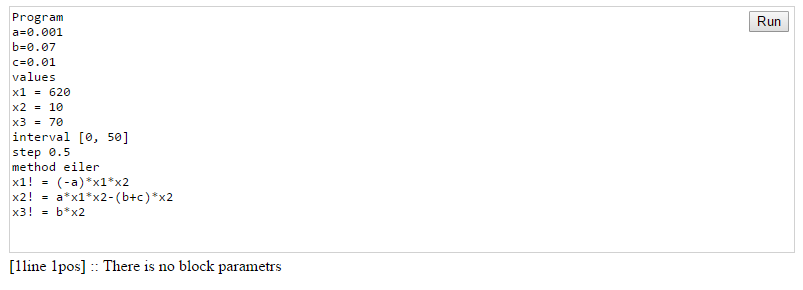


4. Мутация клетки.



# 6. Примеры ошибок

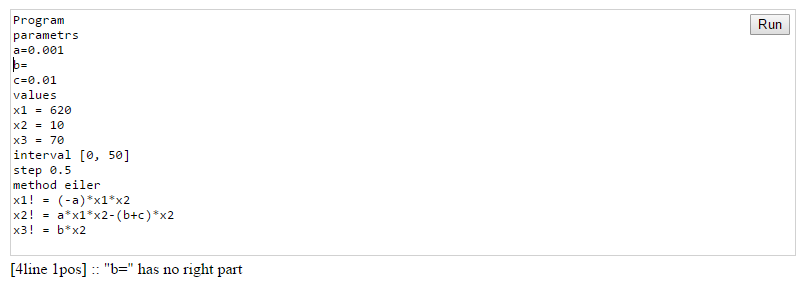
Отсутствие блока



Параметр должен быть числовым



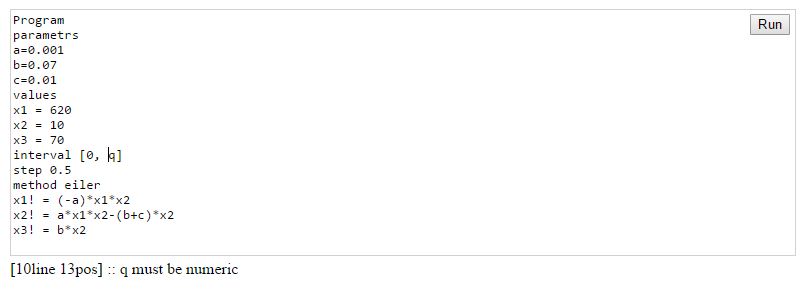
Отсутствует правая часть определения параметра



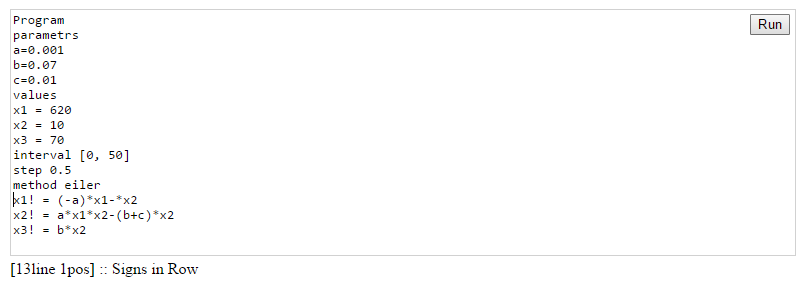
Два знака «равно» в определении значения по умолчанию



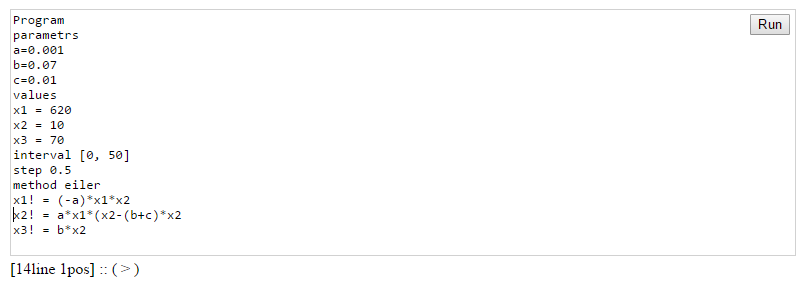
Интервал нечисловой



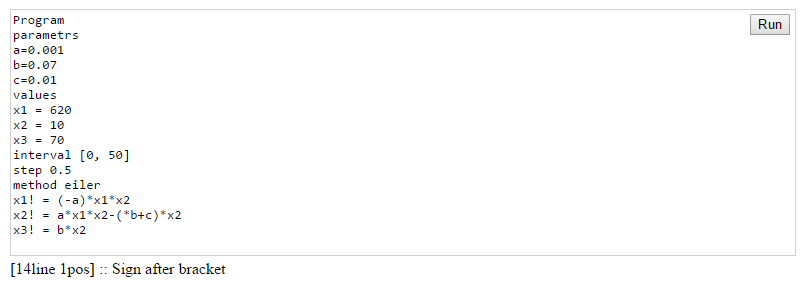
Несколько знаков подряд



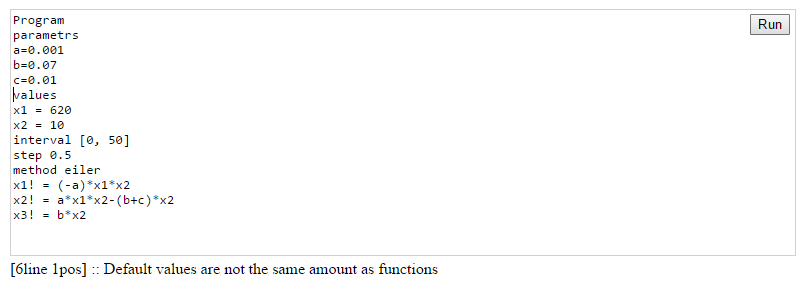
Разное количество открывающих и закрывающих скобок



Математический знак после скобки



Разное количество значений по умолчанию и дифференциальных уравнений



# Заключение

В данной работе была поставлена задача разработки синтаксически управляемого транслятора по БНФ форме.

В ходе решения задачи была разработана программа с графическим интерфейсом. Основные классы программы:

1. **VueInstance** – класс реализует предварительный анализ. Проверяет синтаксическую целостность и корректную разметку полученных данных, производит лексический анализ введенного текста, производит формирование узлов, для последующего синтаксического анализа.

2. **Parser** – класс производит синтаксический анализ, отвечает за корректную последовательность введенных данных. Производит формирование конечного лексического дерева принимаемого классами, отвечающими за математическую обработку.

3. **Math** — класс отвечает за конечную проверку лексического дерева, проверяет целостность данных и их корректную последовательность и производит вычисление выражений.

Программа производит анализ исходного текста и выдает сообщения об ошибках, если они есть. Задача была реализована на языке Javascript с применением IDE WebStorm. Графический интерфейс реализован средствами HTML и CSS. В ходе тестирования реализованного программного обеспечения было выявлено, что программа решает задачу корректно и устойчиво работает на тестовом наборе данных.

# Список литературы

1. Никлаус Вирт. Алгоритмы и структуры данных – М.: «МИР», 2011. – 360 с.
2. Кнут Д. Исскусство программирования, том 3. Сортировка и поиск – 2-е изд. – М.: «Вильямс», 2007. -824 с.
3. Эккель Б. Философия C++. Практическое программирование. – М.:«Питер», 2004. – 608 с.
4. Макконелл, Дж. Основы современных алгоритмов. 2-е дополнительное издание – М.:Техносфера, 2004. – 368 с.
5. Ахо А., Хопкрофт Дж., Ульман Дж. Построение и анализ вычислительных алгоритмов. – М.Ж Мир, 1979. – 384 с
6. Никлаус Вирт. Построение компиляторов – М.: «ДМК Пресс»,2010. – 192 с.

# Приложение А. Листинг программы

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Title</title>

</head>

<link rel="stylesheet" href="./style.css">

<script src="https://unpkg.com/vue/dist/vue.js"></script>

<script src="http://cdnjs.cloudflare.com/ajax/libs/mathjs/3.11.4/math.min.js"></script>

<script src="https://code.highcharts.com/highcharts.js"></script>

<body>

<div id="app">

<div class="container">

<div class="container\_\_text">

<textarea class="text" v-model="text" ref='input'></textarea>

</div>

<div class="container\_\_bnf">

<textarea class="bnf" readonly v-model="bnf"></textarea>

<button @click="main" class="run">Run</button>

</div>

<div class="container\_\_chart"></div>

<div class="container\_\_error">

{{error}}

</div>

</div>

</div>

<div id="chart"></div>

<script>

new Vue({

el: '#app',

data: {

header: 'Транслятор',

projects: [],

text: `

Program

parametrs

a=0.001

b=0.07

c=0.01

values

x1 = 620

x2 = 10

x3 = 70

interval [0, 50]

step 0.5

method eiler

x1! = (-a)\*x1\*x2

x2! = a\*x1\*x2-(b+c)\*x2

x3! = b\*x2`,

toch: 4,

functions: [],

params: {},

method: '',

step: 0,

interval: [0, 0],

exprSign: '!',

bnf: `Lan = "Program Defaults Interval Step Method Operations"

Defaults = "parametrs" var "=" num | "values" var "=" num

Interval = "interval" [real, real]

Step = "step" num

Method = "method" "eiler"|"runge2"|"runge4"

Operation = label "=" expression

expression = part <maths part>

part = expression | func"("var|num|part")

maths = a&s|m&d|exp

exp = "^"

m&d = "\*", "/"

a&s = "+", "-"

funcs = "cos"|"sin"

num = int|real

var = l<symbol\*>

label = symbol+"'"

symbol = l|n

real = int"."int

int = n...n

n = ["0"-"9"]

l = ["a"-"z"]`,

exceptions: {},

error: '',

},

created(){

let self = this;

this.exceptions = {

programBlock: function () {

this.toString = function () {

return 'Therу must be block Program in the beginning';

};

},

missSign: function (sign, place) {

self.getCaretPos(place);

this.value = sign;

this.message = "There is no sign ";

this.toString = function () {

return this.message + this.value;

};

},

dublicateSign: function (sign, place) {

self.getCaretPos(place);

this.value = sign;

this.message = "There is repeated sign ";

this.toString = function () {

return this.message + '"' + this.value + '"';

};

},

missBlock: function (block) {

self.getCaretPos('');

this.value = block;

this.message = "There is no block ";

this.toString = function () {

return this.message + this.value;

};

},

orderBlock: function (X, Y) {

self.getCaretPos(X);

this.toString = function () {

return `Block '${X}' must be before Block '${Y}'`;

};

},

notNumber: function (num) {

self.getCaretPos(num);

this.value = num;

this.message = " must be numeric ";

this.toString = function () {

return this.value + this.message;

};

},

noMethod: function (method) {

self.getCaretPos(method);

this.toString = function () {

return `There is no method ${method}`;

};

},

bracketCount: function (expr, more) {

self.getCaretPos(expr.replace("\\(", "\\(").replace("\\)", "\\)"));

this.toString = function () {

return '( ' + (more ? '>' : '<') + ' )'

};

},

noPart: function (expr, right) {

self.getCaretPos(expr);

this.value = `"${expr.trim()}"`;

this.message = ' has no ' + (right ? 'right' : 'left') + ' part';

this.toString = function () {

return this.value + this.message;

};

},

countDefault: function () {

self.getCaretPos('values');

this.toString = function () {

return 'Default values are not the same amount as functions';

};

},

singBracket: function (expr, before) {

self.getCaretPos(expr);

this.toString = function () {

return 'Sign ' + (before ? 'before' : 'after') + ' bracket';

};

},

signsInRow: function (expr) {

self.getCaretPos(expr);

this.toString = function () {

return 'Signs in Row';

};

},

signInTheEnd: function (expr) {

self.getCaretPos(expr);

this.toString = function () {

return 'Sign in the end';

};

},

notFullScope: function () {

self.getCaretPos('parametrs');

this.toString = function () {

return 'Not all parametrs';

};

},

noLines: function(text){

self.getCaretPos(text);

this.toString = function () {

return 'Not filled ' + text;

};

},

missValue: function(text){

self.getCaretPos(text);

this.toString = function () {

return 'No value for ' + text;

};

}

}

},

methods: {

main(){

this.error = '';

try {

console.log(this.text);

this.text = this.text.split('\n').filter(line => line !== '').join('\n');

this.text = this.text.split('\n');

for (let i = 0, len = this.text.length; i < len; i++) {

this.text[i] = this.text[i].trim();

}

this.text = this.text.join('\n');

const index = this.text.search('Program');

if (index === -1) {

throw new this.exceptions.programBlock()

}

let last\_block = {};

['parametrs', 'values', 'interval', 'step', 'method'].forEach((block, i) => {

const index = this.text.search(block);

if (index === -1) {

throw new this.exceptions.missBlock(block)

}

if (i && index < last\_block.index) {

throw new this.exceptions.orderBlock(block, last\_block.block)

}

last\_block = {block, index};

});

this.getFunctions(this.text.split(/method.\*\n/gi)[1]);

this.getStep(this.text.match(/step\s\*(\d+\.?\d\*)\n/));

this.getInterval(this.text.match(/interval.\*\[(.+),(.+)\]\n/));

this.getMethod(this.text.match(/method(.\*)/));

this.build();

} catch (ex) {

let pos = this.pos;

for (let key in this.exceptions) {

if (this.exceptions.hasOwnProperty(key) && Object.keys(ex).length && ex instanceof this.exceptions[key]) {

console.log(key);

}

}

this.setCaretToPos(this.$refs.input, pos);

this.error = (this.error\_pos || '') + ex.toString();

this.error\_pos = '';

}

},

getFunctions(text){

if (!text)

{

throw new this.exceptions.noLines('functions');

}

let fs = text.trim().split('\n');

let \_\_functions = [];

fs.forEach((func) => {

this.checkSigns(func, '=');

let parts = func.split('=');

const name = parts[0].split(this.exprSign)[0].trim(), f = parts[1].trim();

if (name === '') {

throw new this.exceptions.noPart(func);

} else if (f === '') {

throw new this.exceptions.noPart(func, true);

}

else {

\_\_functions.push({name, f});

}

});

if (\_\_functions.length === 0)

{

throw new this.exceptions.noLines('functions');

}

\_\_functions.forEach(this.checkExpression);

this.functions = \_\_functions;

let values = this.text.split('values\n')[1].split('inter')[0].split('\n').filter(v => v !== '');

this.getDefaults(values);

this.getParams(this.text.match(/parametrs(\n.\*)+values/));

},

getDefaults(ds){

if (ds.length === 0)

{

throw new this.exceptions.noLines('values');

}

if (this.functions.length !== ds.length) {

throw new this.exceptions.countDefault();

}

for (let i = 0, len = ds.length; i < len; i++) {

this.checkSigns(ds[i], '=');

let d = ds[i].split('='), left = d[0].trim(), right = d[1].trim();

let func = this.functions.find(f => f.name === left);

if (func) {

if (left === '') {

throw new this.exceptions.noPart(ds);

} else if (right === '') {

throw new this.exceptions.noPart(ds, true);

}

else {

this.isNumber(right);

func.def = parseFloat(right);

console.log(func.def)

}

}

}

},

getParams(ps){

ps = ps[0].split('parametrs\n')[1];

ps = ps.split('\nvalues')[0];

ps = ps.split('\n');

ps.forEach((\_ps) => {

this.checkSigns(\_ps, '=');

const p = \_ps.split('='), left = p[0].trim(), right = p[1].trim();

if (left === '') {

throw new this.exceptions.noPart(\_ps);

} else if (right === '') {

throw new this.exceptions.noPart(\_ps, true);

}

else {

this.isNumber(right);

this.params[left] = parseFloat(right);

}

})

},

getStep(text){

if (!text)

{

throw new this.exceptions.missValue('step');

}

this.isNumber(text ? text[1] : NaN);

this.step = parseFloat(text[1]);

},

getInterval(text){

if (!text)

{

throw new this.exceptions.missValue('interval');

}

this.isNumber(text[1]);

this.isNumber(text[2]);

this.interval = [parseFloat(text[1]), parseFloat(text[2])];

},

getMethod(text){

const METHODS = ['eiler', 'runge2', 'runge4'];

this.method = text[1].trim().toLowerCase();

if (!METHODS.includes(this.method)) {

throw new this.exceptions.noMethod(this.method);

}

},

calc(f, scope){

return math.eval(f, scope);

},

build(){

try {

let i = this.interval[0] + this.step;

let series = [];

let y = {};

this.functions.forEach(func => {

series.push({

name: func.name,

data: [[this.interval[0], func.def]]

});

y[func.name] = func.def;

});

while (i <= this.interval[1]) {

let next ={};

this.functions.forEach((func) => {

let s = series.find(ser => ser.name === func.name);

next[func.name] = this[this.method](y, i, this.step, func.f, func.name);

s.data.push([i, +next[func.name].toFixed(2)]);

});

Object.assign(y, next);

i += this.step;

}

this.buildChart(series);

} catch (ex) {

}

},

eiler(y, x, dx, f, name)

{

return y[name] + dx \* this.calc(f, Object.assign(y, this.params, {x}));

},

runge2(y, x, dx, f, name){

let s1 = this.calc(f, Object.assign(y, this.params, {x})),

s2 = this.calc(f, Object.assign(y, this.params, {x: x + dx, [name]: y[name] + dx \* s1}));

return y[name] + dx \* (s1 + s2) / 2;

},

runge4(y, x, dx, f, name)

{

let k1 = dx \* this.calc(f, Object.assign(y, this.params, {x})),

k2 = dx \* this.calc(f, Object.assign(y, this.params, {

x: x + dx / 2.0,

[name]: y[name] + k1 / 2.0

})),

k3 = dx \* this.calc(f, Object.assign(y, this.params, {

x: x + dx / 2.0,

[name]: y[name] + k2 / 2.0

})),

k4 = dx \* this.calc(f, Object.assign(y, this.params, {x: x + dx, [name]: y[name] + k3}));

return y[name] + (k1 + 2.0 \* k2 + 2.0 \* k3 + k4) / 6.0;

},

buildChart(series)

{

Highcharts.chart('chart', {

title: {

text: 'График'

},

credits: {

enabled: false

},

xAxis: {

crosshair: true,

legend: {

enabled: false

}

},

legend: {

enabled: true

},

colors: ['#ff0000', '#52c81f', '#f7a35c', '#8085e9', '#90ed7d',

'#f15c80', '#e4d354', '#2b908f', '#f45b5b', '#91e8e1', '#99C1F7'],

plotOptions: {

series: {

lineWidth: 1,

states: {

lineWidth: 1,

hover: {

lineWidth: 1

}

},

marker: {

radius: 3,

symbol: 'triangle',

states: {

hover: {

radius: 3

}

}

}

}

}, tooltip: {

shared: true,

pointFormat: '<span style="color:{point.color}">\u25CF</span> {series.name}: <b>{point.y}</b><br/>'

},

series: series

});

},

checkSigns(str, sign){

const match = str.match(new RegExp(sign, 'g'));

if (match === null) {

throw new this.exceptions.missSign(sign, str)

}

if (match.length > 1) {

throw new this.exceptions.dublicateSign(sign, str.split('=')[0])

}

},

isNumber(n)

{

if (!isNaN(parseFloat(n)) && isFinite(n)) {

} else {

throw new this.exceptions.notNumber(n);

}

},

checkExpression(expr){

const signs = '[\+|\-|\\*|\/|^]';

const signsinrow = /([\*|+|\-|\/|^])\s\*([\*|+|\-|\/|^])/g;

const bracket = {

open: "\\(",

close: "\\)"

};

const LIST\_OF\_ERRORS = [

//

];

const f = expr.f.split('').filter(v => v !== ' ').join('');

const open = f.match(new RegExp(bracket.open, "g")) || [],

close = f.match(new RegExp(bracket.close, "g")) || [];

if (open.length !== close.length) {

throw new this.exceptions.bracketCount(expr.name + this.exprSign, open.length > close.length);

}

const afterOpen = f.match(new RegExp(bracket.open + signs, "g")) || [],

beforeClose = f.match(new RegExp(signs + bracket.close, "g")) || [];

if (beforeClose.length || afterOpen.length) {

throw new this.exceptions.singBracket(expr.name + this.exprSign, beforeClose.length > 0);

}

const inrow = f.match(signsinrow) || [];

if (inrow.length) {

throw new this.exceptions.signsInRow(expr.name + this.exprSign);

}

const signinthend = f.match(new RegExp(signs + '$')) || [];

if (signinthend.length) {

throw new this.exceptions.signInTheEnd(expr.name + this.exprSign);

}

},

setSelectionRange(input, selectionStart, selectionEnd) {

if (input.setSelectionRange) {

input.focus();

input.setSelectionRange(selectionStart, selectionEnd);

}

else if (input.createTextRange) {

let range = input.createTextRange();

range.collapse(true);

range.moveEnd('character', selectionEnd);

range.moveStart('character', selectionStart);

range.select();

}

},

setCaretToPos (input, pos) {

this.setSelectionRange(input, pos, pos);

},

getCaretPos(place)

{

place || (place = '');

this.pos = this.text.search(place.trim());

let check = false;

this.text.split('\n').forEach((l, i) => {

let index = l.search(place);

if (!check && index !== -1) {

check = true;

this.error\_pos = `[${i + 1}line ${index + 1}pos] :: `

}

});

}

}

})