Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

Лабораторная работа № 3

по дисциплине: «Конструирование компиляторов»

Тема: «Синтаксический разбор с использованием метода рекурсивного спуска»

Выполнил студент группы ИУ7-21М

Констандогло Александр

Москва
17 апреля 2020

Цель работы: приобретение практических навыков реализации синтаксического разбора с использованием метода рекурсивного спуска.

Постановка задачи

Вариант 7.

Дополнить грамматику блоком, состоящим из последовательности операторов присваивания.

Вариант в стиле Алгол-Паскаль.

```
<программа> ->
     <блок>
<блок> ->
     begin <список операторов> end
<список операторов> ->
     <оператор> | <список операторов> ; <оператор>
<оператор> ->
     <идентификатор> = <выражение>
```

Вариант содержит левую рекурсию, которая должна быть устранена. Точка с запятой (;) ставится между операторами. Вариант содержат цепное правило <программа> -> <блок>. Можно начальным символом грамматики назначить нетерминал <блок>. Нетерминал <выражение> определяется в грамматике G2.

Грамматика G2

```
<выражение> ->
     <арифметическое выражение> <операция отношения> <арифметическое
     выражение> |
     <арифметическое выражение>
<арифметическое выражение> ->
     <арифметическое выражение> <операция типа сложения> <терм> |
     <терм>
<терм> ->
     <терм> <операция типа умножения> <фактор> |
     <фактор>
<фактор> ->
     <идентификатор> |
     <константа> |
     ( <арифметическое выражение> )
<операция отношения> ->
     < | <= | = | <> | > | >=
<операция типа сложения> ->
     + | -
<операция типа умножения> ->
     * | /
```

Замечания.

1. Нетерминалы <идентификатор> и <константа> – это лексические единицы (лексемы), которые оставлены неопределенными, а при выполнении лабораторной работы можно либо рассматривать их как терминальные символы, либо определить их по своему усмотрению и добавить эти определения.
2. Терминалы ( ) – это разделители и символы пунктуации.
3. Терминалы < <= = <> > >= + - * / – это знаки операций.
4. Нетерминал <выражение> – это начальный символ грамматики.

Если между символом присваивания (=) и символом операции отношения (=) возникает конфликт, то можно для любого из них ввести новое изображение, например, :=, <-, == и т. п.

Для модифицированной грамматики написать программу нисходящего синтаксического анализа с использованием метода рекурсивного спуска.

Модифицированная грамматика G0

В данной грамматике была устранена левая рекурсия и проведена левая факторизация. Начальным символом грамматики назначен нетерминал <блок>. В качестве символа присваивания примем :=. Все терминальные символы разделяются одним пробелом. Нетерминалы <идентификатор> и <константа> рассматриваются как терминальные символы, для этого введём следующие правила:

```
<идентификатор> -> identifier
<константа> -> const
```

Грамматика G0

```
<блок> ->
    begin <список операторов> end
<список операторов> ->
    identifier := <выражение> <список операторов 1>
<список операторов 1> ->
    ; <оператор> <список операторов 1> | ε
<оператор> ->
    identifier := <выражение>
<выражение> ->
    const <выражение 4> |
    identifier <выражение 4> |
    ( <арифметическое выражение> ) <выражение 4>
<выражение 1> ->
    <операция отношения> <арифметическое выражение> | ε
<выражение 3> ->
    <операция отношения> <арифметическое выражение> |
    <арифметическое выражение 1> <выражение 1> |
    ε
<выражение 4> ->
```

```
     <операция отношения> <арифметическое выражение> |
     <арифметическое выражение 1> <выражение 1> |
     <терм 1> <выражение 3> |
     ε
<арифметическое выражение> ->
     <терм> <арифметическое выражение 2>
<арифметическое выражение 1> ->
     <операция типа сложения> <терм> <арифметическое выражение 2>
<арифметическое выражение 2> ->
     <арифметическое выражение 1> | ε
<терм> ->
     const <терм 2> |
     identifier <терм 2> |
     ( <арифметическое выражение> ) <терм 2>
<терм 1> ->
     <операция типа умножения> <фактор> <терм 2>
<терм 2> ->
     <терм 1> | ε
<фактор> ->
     identifier |
     const |
     ( <арифметическое выражение> )
<операция отношения> ->
     < | <= | = | <> | > | >=
<операция типа сложения> ->
     + | -
<операция типа умножения> ->
     * | /
```

Текст программы, проводящей нисходящий синтаксический анализ

Программа написана на языке C++. Для ее корректной работы в операционной системе должна быть установлена утилита dot.

parser.h

```
#pragma once
#include <map>
#include <set>
#include <vector>
#include <string>
#include <fstream>

class Symbol {
         std::string sym;
         bool terminal;
public:
         Symbol(std::string sym, bool terminal);
         bool is_terminal();
         std::string get_sym();
         bool get_type();
};

typedef std::map<std::pair<std::string, std::string>, std::vector<Symbol>> Table;

class PredTable {
```

```
        const std::set<std::string> ralation_operator_syms = std::set<std::string>({ "<", "<=", "=", ">=", "<>", ">" });
        const std::set<std::string> addition_syms = std::set<std::string>({ "+", "-" });
        const std::set<std::string> multiplication_syms = std::set<std::string>({ "*", "/" });
        Table tbl;
public:
        PredTable();
        Table get_table();
};

class Node {
        int num;
        std::string name;
        bool terminal;
        bool error;
        std::vector<Node*> children;
public:
        Node(int num, std::string name, bool terminal);
        void set_error();
        void add_child(Node *child);
        int get_num();
        std::string get_name();
        bool get_terminal();
        bool get_error();
        std::vector<Node*> get_children();
        ~Node();
};

class Storage {
        std::ifstream text;
        std::vector<Node*> tree;
        int current_node;
        Table tbl;
        std::vector<std::string> symbols;
        void parse(std::string nonterminal, std::string first, Node *root);
        void make_graph();
public:
        Storage(const char* filename);
        bool parse();
        bool handle();
        ~Storage();
};
```

parser.cpp

```
#include "parser.h"
#include <algorithm>
#include <fstream>
#include <regex>

#define ADD(K, V) insert(pair<pair<string, string>, vector<Symbol>>(pair<string, string>K, vector<Symbol>V))

using namespace std;

Symbol::Symbol(string sym, bool terminal) : sym(sym), terminal(terminal) {}

bool Symbol::is_terminal() {
        return terminal;
}

string Symbol::get_sym() {
        return sym;
}

bool Symbol::get_type() {
        return terminal;
}

PredTable::PredTable() {
        tbl.ADD(("blok", "begin"), ({ Symbol("begin", true), Symbol("operators", false), Symbol("end", true) }));
        tbl.ADD(("operators", "identifier"), ({ Symbol("identifier", true), Symbol(":=", true), Symbol("expression", false),
Symbol("operators_1", false) }));
        tbl.ADD(("operators_1", ";"), ({ Symbol(";", true), Symbol("operator", false), Symbol("operators_1", false) }));
        tbl.ADD(("operators_1", ""), ({ Symbol("", false) }));
        tbl.ADD(("operator", "identifier"), ({ Symbol("identifier", true), Symbol(":=", true), Symbol("expression", false) }));
        tbl.ADD(("expression", "const"), ({ Symbol("const", true), Symbol("expression_4", false) }));
        tbl.ADD(("expression", "identifier"), ({ Symbol("identifier", true), Symbol("expression_4", false) }));
        tbl.ADD(("expression", "("), ({ Symbol("(", true), Symbol("arithmetic_expression", false), Symbol(")", true),
Symbol("expression_4", false) }));
        tbl.ADD(("expression_1", ""), ({ Symbol("", false) }));
        tbl.ADD(("expression_3", ""), ({ Symbol("", false) }));
        tbl.ADD(("expression_4", ""), ({ Symbol("", false) }));
        tbl.ADD(("arithmetic_expression", "const"), ({ Symbol("term", false), Symbol("arithmetic_expression_2", false) }));
        tbl.ADD(("arithmetic_expression", "identifier"), ({ Symbol("term", false), Symbol("arithmetic_expression_2", false) }));
```

```
	tbl.ADD(("arithmetic_expression", "("), ({ Symbol("term", false), Symbol("arithmetic_expression_2", false) }));
	tbl.ADD(("arithmetic_expression_2", ""), ({ Symbol("", false) }));
	tbl.ADD(("term", "const"), ({ Symbol("const", true), Symbol("term_2", false) }));
	tbl.ADD(("term", "identifier"), ({ Symbol("identifier", true), Symbol("term_2", false) }));
	tbl.ADD(("term", "("), ({ Symbol("(", true), Symbol("arithmetic_expression", false), Symbol(")", true), Symbol("term_2",
false) }));
	tbl.ADD(("term_2", ""), ({ Symbol("", false) }));
	tbl.ADD(("faktor", "identifier"), ({ Symbol("identifier", true) }));
	tbl.ADD(("faktor", "const"), ({ Symbol("const", true) }));
	tbl.ADD(("faktor", "("), ({ Symbol("(", true), Symbol("arithmetic_expression", false), Symbol(")", true) }));
	for (string sym : ralation_operator_syms)
	{
		tbl.ADD(("expression_1", sym), ({ Symbol(sym, true), Symbol("arithmetic_expression", false) }));
		tbl.ADD(("expression_3", sym), ({ Symbol(sym, true), Symbol("arithmetic_expression", false) }));
		tbl.ADD(("expression_4", sym), ({ Symbol(sym, true), Symbol("arithmetic_expression", false) }));
	}
	for (string sym : addition_syms)
	{
		tbl.ADD(("expression_3", sym), ({ Symbol("arithmetic_expression_1", false), Symbol("expression_1", false) }));
		tbl.ADD(("expression_4", sym), ({ Symbol("arithmetic_expression_1", false), Symbol("expression_1", false) }));
		tbl.ADD(("arithmetic_expression_1", sym), ({ Symbol(sym, true), Symbol("term", false),
Symbol("arithmetic_expression_2", false) }));
		tbl.ADD(("arithmetic_expression_2", sym), ({ Symbol("arithmetic_expression_1", false) }));
	}
	for (string sym : multiplication_syms)
	{
		tbl.ADD(("expression_4", sym), ({ Symbol("term_1", false), Symbol("expression_3", false) }));
		tbl.ADD(("term_1", sym), ({ Symbol(sym, true), Symbol("faktor", false), Symbol("term_2", false) }));
		tbl.ADD(("term_2", sym), ({ Symbol("term_1", false) }));
	}
}

Table PredTable::get_table() { return tbl; }

Node::Node(int num, string name, bool terminal) : num(num), name(name), terminal(terminal), error(false)
{ }

void Node::set_error()
{
	error = true;
}

void Node::add_child(Node *child)
{
	children.push_back(child);
}

int Node::get_num()
{
	return num;
}

string Node::get_name()
{
	return name;
}

bool Node::get_terminal()
{
	return terminal;
}

bool Node::get_error()
{
	return error;
}

vector<Node*> Node::get_children()
{
	return children;
}

Node::~Node()
{
	children.clear();
}
```

```
void Storage::parse(string nonterminal, string first, Node *root)
{
	if (first != *symbols.rbegin())
		if (tbl.find(pair<string, string>(nonterminal, "")) != tbl.end())
		{
			tree.push_back(new Node(current_node, "", true));
			root->add_child(tree[current_node]);
			current_node++;
			return;
		}
	string err;
	int num;
	for (Symbol sym : tbl[pair<string, string>(nonterminal, first)])
	{
		num = 0;
		err.clear();
		if (sym.get_type())
		{
			do {
				if (symbols.size() > 0)
					if (sym.get_sym() == symbols[symbols.size() - 1 - num])
					{
						if (num != 0)
						{
							if (err.size() > 0)
								err.pop_back();
							tree.push_back(new Node(current_node, err, sym.get_type()));
							root->add_child(tree[current_node]);
							tree[current_node]->set_error();
							root->set_error();
							current_node++;
							symbols.erase(symbols.end() - num, symbols.end());
							err.clear();
							num = 0;
						}
						tree.push_back(new Node(current_node, sym.get_sym(), sym.get_type()));
						root->add_child(tree[current_node]);
						current_node++;
						symbols.pop_back();
					}
					else
					{
						err = err + symbols[symbols.size() - 1 - num] + " ";
						num++;
					}
				else
					root->set_error();
			} while (0 < num && num < symbols.size());
		}
		else
		{
			do {
				if (symbols.size() > 0 && tbl.find(pair<string, string>(sym.get_sym(),
symbols[symbols.size() - 1 - num])) == tbl.end() && tbl.find(pair<string, string>(sym.get_sym(), "")) == tbl.end())
				{
					err = err + symbols[symbols.size() - 1 - num] + " ";
					num++;
				}
				else
				{
					if (num != 0)
					{
						if (err.size() > 0)
							err.pop_back();
						tree.push_back(new Node(current_node, err, true));
						root->add_child(tree[current_node]);
						tree[current_node]->set_error();
						root->set_error();
						current_node++;
						symbols.erase(symbols.end() - num, symbols.end());
						err.clear();
						num = 0;
					}
					tree.push_back(new Node(current_node, sym.get_sym(), sym.get_type()));
					root->add_child(tree[current_node]);
					current_node++;
					if (symbols.size() == 0)
					{
```

```
                                                        if (tbl.find(pair<string, string>(sym.get_sym(), "")) == tbl.end())
                                                                tree[current_node - 1]->set_error();

                                                        return;
                                                }
                                                else
                                                {
                                                        parse(sym.get_sym(), *symbols.rbegin(), tree[current_node - 1]);
                                                }
                                        }
                                } while (0 < num && num < symbols.size());
                        }
                        if (num > 0)
                                root->set_error();
                }
        }

void Storage::make_graph()
{
        ofstream tree_dot("tree.dot");
        if (!tree_dot.is_open())
                throw runtime_error("Не удалось открыть файл tree.dot для записи\n");
        tree_dot << "digraph G{" << endl << "node[shape=rectangle style=filled fillcolor=white fontsize=12];" << endl;
        for (Node* node : tree)
        {
                tree_dot << node->get_num() << "[label=\"" << node->get_name() << "\"";
                if (node->get_terminal())
                        tree_dot << " fillcolor=lightgrey shape=ellipse";
                if (node->get_error())
                        tree_dot << " fillcolor=coral";
                tree_dot << "];" << endl;
                for (Node* child : node->get_children())
                        tree_dot << node->get_num() << "->" << child->get_num() << ";" << endl;
        }
        tree_dot << "}" << endl;
        tree_dot.close();
        system("dot -Tsvg tree.dot -o tree.svg");
}

Storage::Storage(const char* filename)
{
        tbl = PredTable().get_table();
        text.open(filename);
        if (!text.is_open())
                throw runtime_error("Не удалось открыть файл для чтения\n");
        text.seekg(0, ios::end);
        size_t size = text.tellg();
        string fstr(size + 1, ' ');
        text.seekg(0);
        text.read(&fstr[0], size);
        regex exp("[ \t\r\n]+");
        string str = regex_replace(fstr, exp, " ");
        if (str[0] == ' ')
                str.erase(0, 1);
        string delimiter = " ";
        size_t pos = 0;
        while ((pos = str.find(delimiter)) != string::npos) {
                symbols.push_back(str.substr(0, pos));
                str.erase(0, pos + delimiter.length());
        }
        reverse(symbols.begin(), symbols.end());
        current_node = 0;
}

bool Storage::parse()
{
        string err;
        size_t i, num, size = symbols.size();
        for (i = size - 1; i > -1; i--)
                if (symbols[i] != "begin")
                        err = err + symbols[i] + " ";
        tree.push_back(new Node(current_node, "blok", false));
        current_node++;
        if (err.size() > 0)
        {
                tree.push_back(new Node(current_node, err, true));
                tree[0]->add_child(tree[current_node]);
                tree[0]->set_error();
```

```
                    tree[current_node]->set_error();
                    current_node++;
                    err.clear();
                    num = size - i - 1;
                    symbols.erase(symbols.end() - num, symbols.end());
          }
          parse("blok", "begin", tree[0]);
          while (symbols.size() > 0)
          {
                    err = err + *symbols.rbegin() + " ";
                    symbols.pop_back();
          }
          if (err.size() > 0)
          {
                    err.pop_back();
                    tree.push_back(new Node(current_node, err, true));
                    tree[0]->add_child(tree[current_node]);
                    tree[0]->set_error();
                    tree[current_node]->set_error();
                    current_node++;
          }
          err.clear();
          make_graph();
          for (Node* node : tree)
                    if (node->get_error())
                              return false;
          return true;
}

bool Storage::handle()
{
          bool ans = parse();
          make_graph();
          return ans;
}

Storage::~Storage()
{
          text.close();
          for (auto i = 0; i < tree.size(); i++)
                    delete tree[i];
          tree.clear();
}
```

Source.cpp

```
#include <iostream>
#include "parser.h"

using namespace std;

int main(int argc, char* argv[])
{
          setlocale(LC_ALL, "Russian");
          try {
                    Storage s(argv[1]);
                    bool ans = s.handle();
                    if (ans)
                              cout << "Ошибок нет" << endl;
                    else
                              cout << "Текст содержит ошибки" << endl;
          }
          catch (const exception& err) {
                    cerr << err.what() << endl;
          }
          return 0;
}
```

Результаты тестирования

Файл `UnitTest.cpp` содержит тесты для проверки исходных текстов. Далее приведены тесты.

UnitTest.cpp

```
#include "pch.h"
#include "CppUnitTest.h"
#include "../parser/parser.h"
```

```
using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace UnitTest
{
	TEST_CLASS(UnitTest)
	{
	public:

		TEST_METHOD(Test1)
		{
			try {
				const char* inp = "..\\inps\\inp0.txt";
				Storage s(inp);
				Assert::IsFalse(s.parse());
			}
			catch (const std::exception& err) {
				Assert::IsTrue(false);
			}
		}

		TEST_METHOD(Test2)
		{
			try {
				const char* inp = "..\\inps\\inp1.txt";
				Storage s(inp);
				Assert::IsTrue(s.parse());
			}
			catch (const std::exception& err) {
				Assert::IsTrue(false);
			}
		}

		TEST_METHOD(Test3)
		{
			try {
				const char* inp = "..\\inps\\inp2.txt";
				Storage s(inp);
				Assert::IsFalse(s.parse());
			}
			catch (const std::exception& err) {
				Assert::IsTrue(false);
			}
		}

		TEST_METHOD(Test4)
		{
			try {
				const char* inp = "..\\inps\\inp3.txt";
				Storage s(inp);
				Assert::IsFalse(s.parse());
			}
			catch (const std::exception& err) {
				Assert::IsTrue(false);
			}
		}

		TEST_METHOD(Test5)
		{
			try {
				const char* inp = "..\\inps\\inp4.txt";
				Storage s(inp);
				Assert::IsFalse(s.parse());
			}
			catch (const std::exception& err) {
				Assert::IsTrue(false);
			}
		}

		TEST_METHOD(Test6)
		{
			try {
				const char* inp = "..\\inps\\inp5.txt";
				Storage s(inp);
				Assert::IsFalse(s.parse());
			}
			catch (const std::exception& err) {
				Assert::IsTrue(false);
			}
		}
	};
}
```

Содержимое всех TXT-файлов приведено в приложении.

№	Название функции	Ожидаемый вывод	Вывод	Результат теста
1	`Test0`	Ошибки есть	Ошибки есть	Пройден

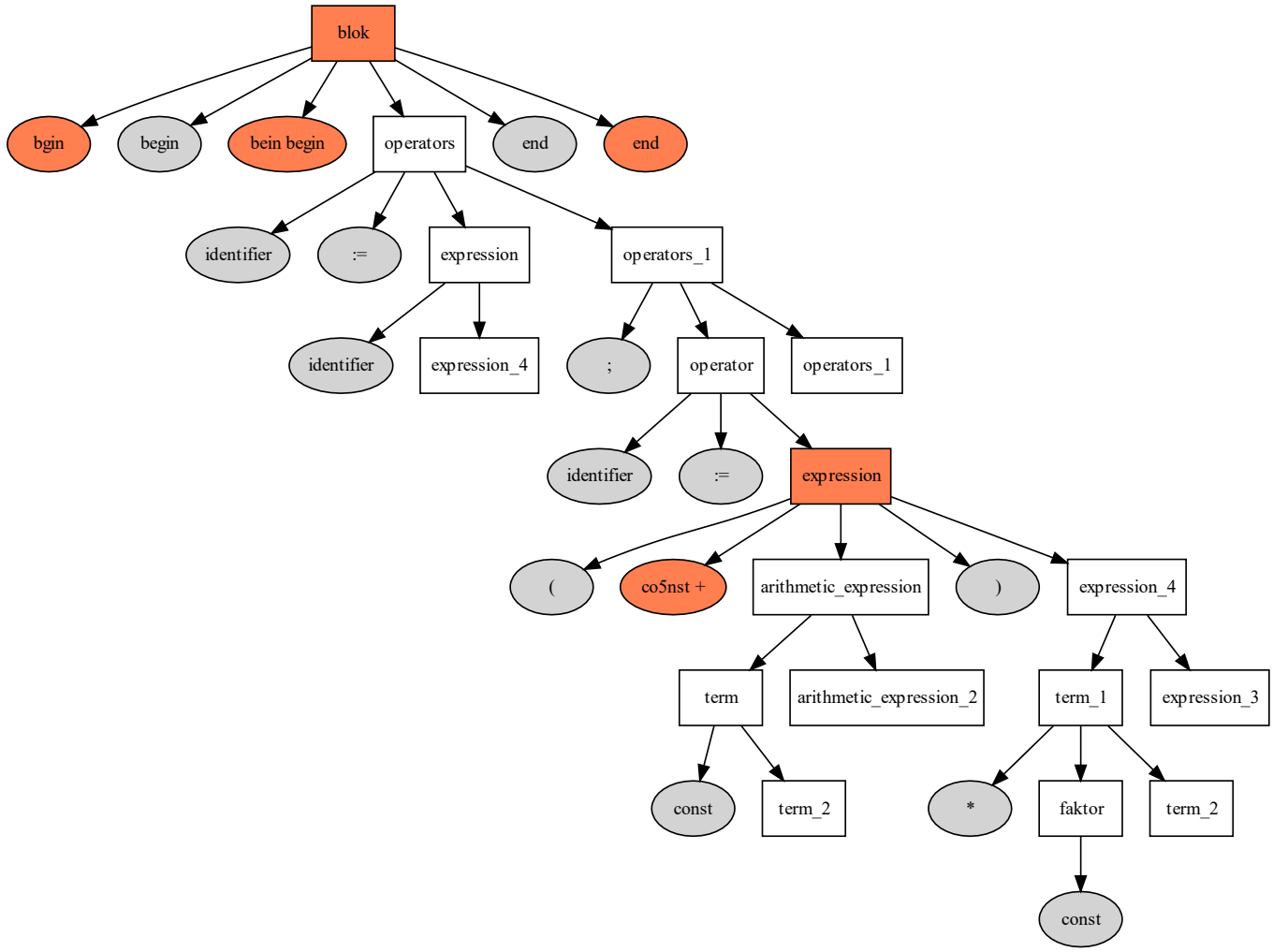
2	`Test1`	Ошибок нет	Ошибок нет	Пройден
3	`Test2`	Ошибки есть	Ошибки есть	Пройден
4	`Test3`	Ошибки есть	Ошибки есть	Пройден
5	`Test4`	Ошибки есть	Ошибки есть	Пройден
6	`Test5`	Ошибки есть	Ошибки есть	Пройден

Результаты выполнения программы

Программа принимает TXT-файл. Результатом является строка «Ошибок нет» или «Текст содержит ошибки» и файл в формате SVG с визуализацией дерева обзора. Все нетерминалы в дереве обозначены прямоугольниками, терминалы – овалами. Если терминал содержит ошибку, то он и его родитель закрашиваются в красный цвет. Если ожидаемый терминал отсутствует в тексте, то на дереве он не отображается, а его родитель закрашивается в красный цвет. Если нетерминал на дереве не имеет потомков и при этом не является красным, то согласно правилам грамматики он порождает пустой символ.

Ниже приведён пример выполнения программы для файла `inp0.txt`, содержимое которого приведено в приложении.

```
Текст содержит ошибки
```



Выводы

В результате выполнения лабораторной работы была написана программа нисходящего синтаксического анализа с использованием метода рекурсивного спуска. Также проведена визуализация дерева разбора.

Приложение

inp0.txt

```
bgin begin bein begin
identifier
:=   identifier  ;
identifier := ( co5nst + const ) * const
end end
```

inp1.txt

```
begin
     identifier := identifier  ;
     identifier := ( const + const ) * const <> identifier + const /
identifier ;
     identifier := identifier * const = identifier
end
```

inp2.txt

```
begin
     identifier := identifier  ;
     identifier := ( const + const ) * const <> identifier + const /
identifier = const => ( const );
     identifier := identifier * const = identifier
end
```

inp3.txt

```
begin
     identifier := identifier  ;
     identifier := ( const + const ) * con4st < > identifier + const
/ identifier = const => ( const ) ;
     identifier := identifier * const = identifier
end
```

inp4.txt

```
begin
end
begin
     identifier := identifier  ;
     identifier := ( const + const ) * const <> identifier + const ;
end
```

inp5.txt

```
begin
     identifier :=
```