|  |  |
| --- | --- |
| Gerb-BMSTU_01 | ***«*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\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**ОТЧЕТ**

**по домашнему заданию № 2**

**Дисциплина: Машинно-зависимые языки и основы компиляции**

**Название лабораторной работы: Лексические и синтаксические анализаторы**

Студент гр. ИУ6-41Б  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ С. А. Самодурова**

(Подпись, дата) (И.О. Фамилия)

Преподаватель  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ С. С. Данилюк**

(Подпись, дата) (И.О. Фамилия)

Москва, 2023

**Цель:** закрепить знания теоретических основ и основных методов, приемов разработки лексических и синтаксических анализаторов регулярных и контекстно-свободных формальных языков.

**Текст задания:**

Разработать грамматику и распознаватель имени процедуры или функции языка программирования Pascal. Считать, что только параметры только стандартных (скалярных) типов. Например:

Procedure as3(Var a\_34,bjt:real; car:integer; const ddd:byte);

Function hjkh(Var k:integer):Char;

**Грамматика в форме Бекуса-Наура:**

<procedure declaration> ::= procedure <идентификатор> ([<список параметров>]);

<function declaration> ::= function <идентификатор> ([<список параметров>]) : <тип>;

<список параметров> ::= <параметр> {; <параметр>}

<параметр> ::= <var параметр> | <value параметр>

<var параметр> ::= var <идентификатор> : <тип>

<value параметр> ::= <идентификатор> : <тип>

<тип> ::= integer | real | boolean | char | string

<идентификатор> ::= <буква> {<буква> | <цифра>}

<буква> ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | \_

<цифра> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Синтаксическая диаграмма представлена на рисунке 1.

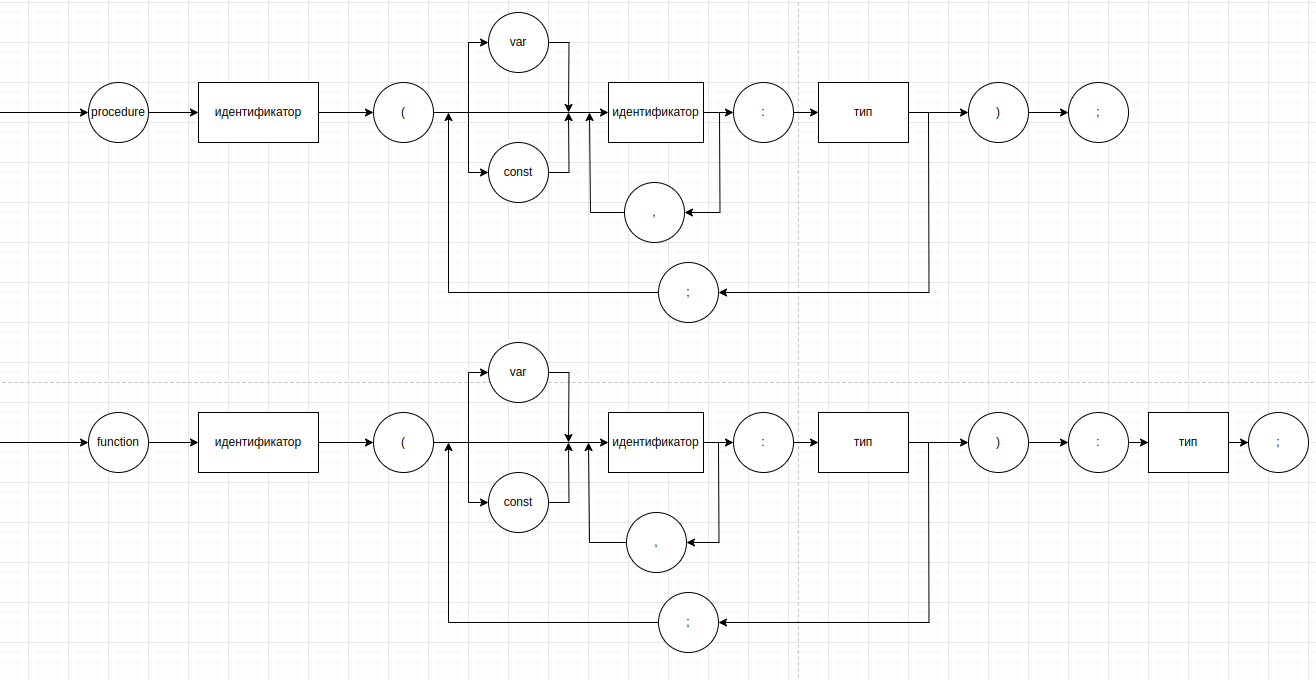


Рисунок 1 — синтаксическая диаграмма

**Код программы:**

#include <iostream>

using namespace std;

string Types[6] {"integer", "real", "boolean", "char", "byte", "string"};

string Rez\_words[2] {"var", "const"};

string delete\_space(string s){

int i = 0;

while (s[i]==' '){

s = s.substr(1);

}

return s;

}

int check\_type(string s){

if (s.length()<1){

cout << "Type cant be blank! " << s << endl;

return 0;

}

for (int i=0;i<6;i++){

if (s==Types[i]){

cout <<"Reserved word: "<< Types[i] << endl;

return 1;

}

}

cout << "Incorrect type: " << s << endl;

return 0;

}

int check\_word(string s){

if (s.length()<1){

cout << "Word cant be blank! " << s << endl;

return 0;

}

for (int i=0;i<2;i++){

if (s==Rez\_words[i]){

cout <<"Reserved word: "<< Rez\_words[i] << endl;

return 1;

}

}

cout << "Incorrect word: " << s << endl;

return 0;

}

int check\_symbols(string s){

if (!(char(s[0]) > 64 and char(s[0]) < 91 or char(s[0]) > 96 and char(s[0]) < 123 or char(s[0]) == 95)){

cout<<"Wrong symbol: "<<s[0]<<endl;

return 0;

}

for (int i =1; i < s.length(); i++){

if (!((char(s[i]) > 64 and char(s[i]) < 91) or (char(s[i]) == 95) or (char(s[i]) > 96 and char(s[i]) < 123) or (char(s[i]) > 47 and char(s[i]) < 58))){

cout<<"Wrong symbol: "<<s[i]<<endl;

return 0;

}

}

return 1;

}

int check\_name\_func(string s){

s = delete\_space(s);

if(check\_symbols(s)==0){

return 0;

}

if (s.length()==0){

cout << "Name of func argument cant be blank" << endl;

return 0;

}

return 1;

}

int check\_name(string stroka){

// здесь можно добавить проверку на имя внутренней функции

// if (s[0]!='\*'){

// return 0;

// }

// cout<<s<<endl;

string name;

if (stroka.find(' ')!=string::npos){

string word = stroka.substr(0, stroka.find(' '));

if (check\_word(word)==0){

return 0;

}

stroka = stroka.erase(0, stroka.find(' '));

}

stroka = delete\_space(stroka);

while (stroka.length()!=0){

// cout<<stroka<<"do"<<endl;

if (stroka.find(',') != string::npos){

name = stroka.substr(0, stroka.find(','));

//

// cout<<", "<<name<<endl

if(check\_symbols(name)==0){

return 0;

}

cout<<"Identifier: "<<name<<endl;

stroka = stroka.erase(0, stroka.find(',')+1);

}

else {

name = stroka.substr(0, stroka.find(':'));

if(check\_symbols(name)==0){

return 0;

}

cout<<"Identifier: "<<name<<endl;

stroka = stroka.erase(0, stroka.find(':')+1);

}

// cout<<stroka<<endl;

if (name.length()==0){

cout << "Name of parametr argument cant be blank" << endl;

return 0;

}

}

return 1;

}

int check\_arguments(string s){

s = s + "!";

// cout<<s<<endl;

while (s.length()>3){

string arg;

string type;

if (s.find(':')!=string::npos){

arg = s.substr(0, s.find(':')+1);

if(check\_name(arg)==0){

return 0;

}

s=s.erase(0, s.find(':')+1);

if (s.find(';')!=string::npos){

type = s.substr(0, s.find(';'));

if(check\_type(type)==0){

return 0;

}

s=s.erase(0, s.find(' '));

}

else {

type = s.substr(0, s.find('!'));

if(check\_type(type)==0){

return 0;

}

s=s.erase(0, s.find('!'));

}

}

s = delete\_space(s);

}

return 1;

}

int check\_brackets(string str){ ////ok

int br1 = 0, br2 = 0;

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

if (str[i] == '(')

br1++;

if (str[i] == ')')

br2++;

}

if (br1 == br2)

return 1;

cout << "Incorrect brackets" << endl;

return 0;

}

int check\_after\_proc(string s){

s = s.substr(s.rfind(')')+1, s.length());

// cout << s;

int k=0;

for (int i =0; i< s.length(); i++){

if (s[i]!=' ' && s[i]!=';'){

cout << "Incorrect symbols after string: " << s[i] << endl;

return 0;

}

if (s[i]==';')

k++;

}

if (k!=1){

cout << "Incorrect count of ; after string " << endl;

return 0;

}

return 1;

}

int check\_after\_func(string s){

if (s.find(':')==string::npos){

cout << "Type of function cant be blank" << endl;

return 0;

}

s=s.substr(1,s.find(';')-1);

// cout<<s<<endl;

if (check\_type(s)==0){

return 0;

}

return 1;

}

int check\_proc(string s){

string vd = s.substr(0, 9);

cout << "Procedure with name: ";

vd = s.substr(10, s.length()-6);

string func\_name = vd.substr(0, vd.find('('));

cout <<func\_name<< " and params: " <<endl;

check\_name\_func(func\_name);

if (check\_brackets(s)==0){

return 0;

}

vd=vd.erase(0,vd.find('('));

// cout<<vd<<endl;

string str = vd.substr(vd.find('(')+1, vd.find(')')-1);

if (check\_arguments(str)==0){

return 0;

}

if (check\_after\_proc(s)==0){

return 0;

}

return 1;

}

int check\_func(string s){

cout << "Function with name: ";

string vd = s.substr(9, s.length()-6);

string func\_name = vd.substr(0, vd.find('('));

cout <<func\_name<< " and params: " <<endl;

check\_name\_func(func\_name);

if (check\_brackets(s)==0){

return 0;

}

vd=vd.erase(0,vd.find('('));

cout<<vd<<endl;

string str = vd.substr(vd.find('(')+1, vd.find(')')-1);

if (check\_arguments(str)==0){

return 0;

}

cout << "Returns: ";

if (check\_after\_func(vd.substr(vd.find(')')+1, vd.find(';')))){

return 0;

}

if (check\_after\_proc(s)==0){

return 0;

}

cout << str << endl;

return 1;

}

int check(string s){

s = delete\_space(s);

if (s.substr(0, 9) == "Procedure") {

check\_proc(s);

}

else if (s.substr(0, 8) == "Function") {

check\_func(s);

}

else {

cout << "Inccorect symbols";

return 0;

}

return 1;

}

int main()

{

string inp = "Procedure as3(var a\_34,bjt:real; car:integer; const ddd:byte);";

cout << inp << endl;

check(inp);

return 0;

}

**Результат работы программы:**

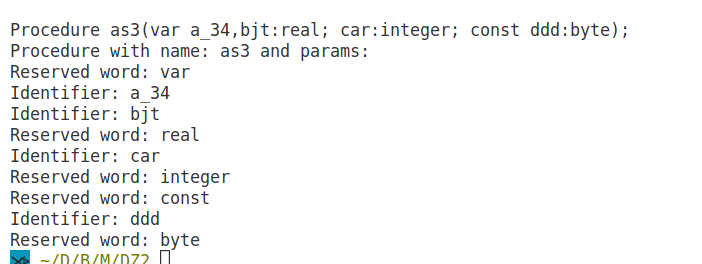
****

Рисунок 2 — результат выполнения

Таблица 1 — проведенные тесты

|  |  |  |  |
| --- | --- | --- | --- |
| № | Входные данные | Ожидаемый результат | Полученный результат |
| 1 | Procedure as3(var a\_34,bjt:real; car:integer; const ddd:byte); | Procedure as3(var a\_34,bjt:real; car:integer; const ddd:byte);  Procedure with name: as3 and params:  Reserved word: var  Identifier: a\_34  Identifier: bjt  Reserved word: real  Identifier: car  Reserved word: integer  Reserved word: const  Identifier: ddd  Reserved word: byte | Procedure as3(var a\_34,bjt:real; car:integer; const ddd:byte);  Procedure with name: as3 and params:  Reserved word: var  Identifier: a\_34  Identifier: bjt  Reserved word: real  Identifier: car  Reserved word: integer  Reserved word: const  Identifier: ddd  Reserved word: byte |
| 2 | Procedure as3(g:byte); | Procedure as3(g:byte);  Procedure with name: as3 and params:  Identifier: g  Reserved word: byte | Procedure as3(g:byte);  Procedure with name: as3 and params:  Identifier: g  Reserved word: byte |
| 3 | Function hjkh(var k:integer):char; | Function hjkh(var k:integer):char;  Function with name: hjkh and params:  (var k:integer):char;  Reserved word: var  Identifier: k  Reserved word: integer  Returns: Reserved word: char | Function hjkh(var k:integer):char;  Function with name: hjkh and params:  (var k:integer):char;  Reserved word: var  Identifier: k  Reserved word: integer  Returns: Reserved word: char |

**Контрольные вопросы**

1. Дайте определение формального языка и формальной грамматики.

Формальная грамматика – это математическая система, определяющая язык посредством порождающих правил – правил продукции. Она определяется как четверка: G = (VT, VN, P, S), где VT – алфавит языка или множество терминальных (незаменяемых) символов; VN – множество нетерминальных (заменяемых) символов – вспомогательный алфавит, символы которого обозначают допустимые понятия языка, VT ∩ VN = ∅; V = VT ∪ VN – словарь грамматики; P – множество порождающих правил – каждое правило состоит из пары строк (α, β), где α∈ V+ – левая часть правила, β V\*– правая часть правила: α ∈ β, где строка α должна содержать хотя бы один нетерминал; S ∈ VN –начальный символ – аксиома грамматики.

Формальная грамматика, определяет правила допустимых конструкций

языка.

2. Как определяется тип грамматики по Хомскому?

Тип 0 – грамматики фразовой структуры или грамматики «без ограничений»: α -> β, где α ∈ V+, β ∈ V\*– в таких грамматиках допустимо наличие любых правил вывода, что свойственно грамматикам естественных языков.

Тип 1 – контекстно-зависимые (неукорачивающие) грамматики: α -> β, где α ∈ V+, β ∈ V\*, | α | <= | β | – в этих грамматиках для правил вида α X β → α x β возможность подстановки строки х вместо символа X определяется присутствием подстрок α и β, т. е. контекста, что также свойственно грамматикам естественных языков.

Тип 2 – контекстно-свободные грамматики: A -> β, где A ∈ VN, β ∈ V\* поскольку в левой части правила стоит нетерминал, подстановки не зависят от контекста.

Тип 3 – регулярные грамматики: A -> α, A -> αB, где A, В ∈ VN, α ∈ VT.

3. Поясните физический смысл и обозначения формы Бэкуса –Наура.

Форма Бэкуса-Наура (БНФ) связывает терминальные и нетерминальные символы, используя две операции: «::=» – «можно заменить на»; «|» – «или». Главный плюс этого - группировка правил, определяющих каждый нетерминал.

4. Что такое лексический анализ? Какие методы выполнения лексического анализа вы знаете?

Лексический анализ – это первый этап процесса компиляции текста, написанного на формальном языке программирования. Сканированием преобразуется строка в набор символов. Каждый из которых представляет из себя токен.

5. Что такое синтаксический анализ? Какие методы синтаксического анализа вы знаете? К каким грамматикам применяются перечисленные вами методы?

Синтаксический анализ – процесс распознавания конструкций языка в строке токенов. Метод рекурсивного спуска для грамматик LR(k). Стековый метод для грамматик LL(k).

6. Что является результатом лексического анализа?

Результатом лексического анализа является строка токенов, чаще всего записанных в виде таблицы.

7. Что является результатом синтаксического анализа?

Результатом, помимо распознавания заданной конструкции, является информация об ошибках в выражениях, операторах и описаниях программы.

8. В чем заключается метод рекурсивного спуска?

Метод рекурсивного спуска заключается в построении синтаксических диаграмм всех разбираемых конструкций, потом по этим диаграммам разработать функции проверки конструкций, а затем составить основную программу, начинающую вызов функций с функции, реализующей аксиому языка.

9. Что такое таблица предшествования и для чего она строится?

Она строится во время использования грамматики с предшествованием LLk). Когда строка разбивается на набор символов, являющиеся токенами, строится таблицы предшествования. Она нужна для того, чтобы было видно какой результат следует ожидать от той, или иной операции (когда начинается и заканчивается “основа”).

10. Как с использованием таблицы предшествования осуществляют синтаксический анализ?

Таблица использует специальные обозначении:

– ? – ошибка;

– < – начало основы;

– > – конец основы;

– () – скобки – принадлежат одной основе;

– ► – начало выражения;

– ◄ – конец выражения.

Когда идет проход по строке токенов, происходит сопоставление токенов со значениями в таблице, из чего определяется начало или конец основы или сигнал об ошибке.

**Вывод:** в данной лабораторной работе я закрепила знания о теоретических основах и основных методах, приемах разработки лексических и синтаксических анализаторов регулярных и контекстно-свободных формальных языков.