*Федеральное государственное бюджетное образовательное учреждение* *высшего профессионального образования*

|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | ***«Московский государственный технический университет  имени Н.Э. Баумана»***  ***(МГТУ им. Н.Э. Баумана)*** |

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Компьютерные системы и сети

**Отчет**

**по домашней работе №2**

**Вариант №26**

**Дисциплина:** МЗЯ и ОК

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

Студент гр. ИУ6-41\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Ветров А. С.

(Подпись, дата) (Фамилия И.О.)

Преподаватель  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** Иванова Г. С.

(Подпись, дата) (Фамилия И.О.)

Москва, 2018

**Задание:**

C++. Разработать грамматику и распознаватель описаний условной операции: **<выражение1>?<выражение2>:<выражение3>;.** Учесть, что Выражение 1 может включать операции отношения. В выражениях могут быть использованы только простые переменные и константы целого типа. Например: **a>45?h\*(a+6):k;**

**Анализ и описание грамматики в форме Бекуса-Наура и в виде синтаксических диаграмм:**

1. Описание в форме Бекуса-Наура:

1) <Строка> ::= <Лог выр>?<Выр>:<Выр>;

2) <Лог выр> ::= <Выр><Срав>

3) <Срав> ::= e | > <Выр> | < <Выр> | <= <Выр> | >= <Выр> | == <Выр> | != <Выр>

4) <Выр> ::= <Терм> <Слож>

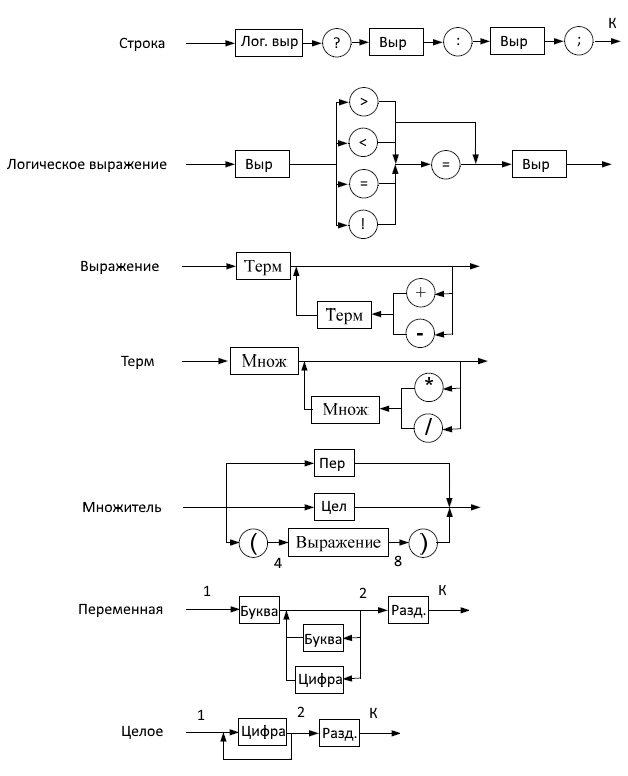
5) <Слож> ::= e | + <Терм> <Слож> | - <Терм><Слож>

6) <Терм> ::= <Множ><Умн>

7) <Умн> ::= e | \* <Множ><Умн> | / <Множ><Умн>

8)<Множ> ::= <Целое> | <Пер> | (<Выр>)

2. Описание в форме синтаксических диаграмм



1. Анализ грамматики

В данной грамматике все правила имеют вид:

,

поэтому она является контекстно-свободной (Тип 2).

В данной грамматике отсутствуют правила с левосторонней рекурсией, во всех правилах для нетерминалов выбор определяется первым символом, за исключением правила №3, где необходимо проанализовать первые два. На основании этого можно определить класс грамматики как LL(2). Реализовывать нисходящий синтаксический анализатор будем методом рекурсивного спуска.

**Исходный код:**

#include "stdafx.h"

using namespace std;

set<char> abc;

set<char> dig;

set<char> sep;

string src;

bool isVar();

bool isInt();

bool isMult();

bool isTerm();

bool isArExp();

bool isLogExp();

bool isIfExp();

int \_tmain(int argc, \_TCHAR\* argv[]){

for(char i = 97; i < 123; i++) { abc.insert(i); }

for(char i = 65; i < 91; i++) { abc.insert(i); }

for(char i = 48; i < 58; i++) { dig.insert(i); }

sep.insert('+'); sep.insert('-'); sep.insert('\*');

sep.insert('/'); sep.insert(')'); sep.insert('(');

sep.insert('?'); sep.insert(':'); sep.insert(';');

sep.insert('>'); sep.insert('<'); sep.insert('=');

sep.insert('!');

char s[255];

puts("Input conditional statement or \'end\' to exit:");

gets(s);

while ( memcmp(s,"end",3) != 0 ){

src = s;

isIfExp();

puts("Input conditional statement or \'end\' to exit:");

gets(s);

}

}

bool isVar(){

string s;

if ( abc.find(src[0]) != abc.end() ){

s += src[0];

src.erase(0,1);

while ( !src.empty() &&

( abc.find(src[0]) != abc.end() ||

dig.find(src[0]) != dig.end() ) ){

s += src[0];

src.erase(0,1);

}

if ( src.empty() || sep.find(src[0]) != sep.end() ){

printf("Variable: %s \n", s.c\_str());

return true;

} else {

printf("Error! Expected digit or letter, got: %c \n", src[0]);

return false;

}

} else { return false; }

}

bool isInt(){

string s;

if ( dig.find(src[0]) != dig.end() ){

s += src[0];

src.erase(0,1);

while ( !src.empty() && dig.find(src[0]) != dig.end() ){

s += src[0];

src.erase(0,1);

}

if ( src.empty() || sep.find(src[0]) != sep.end() ){

printf("Integer: %s \n", s.c\_str());

return true;

} else {

printf("Error! Expected digit, got: %c \n", src[0]);

return false;

}

} else { return false; }

}

bool isMult(){

bool r;

if ( src[0] == '(' ){

printf("Left bracket: \'(\' \n");

src.erase(0,1);

r = isArExp();

if ( r && ( src[0] == ')') ){

printf("Right bracket: \')\' \n");

src.erase(0,1);

} else {

if ( r ){

r = false;

printf("Error! Expected \')\', got: %c \n", src[0]);

}

}

} else {

r = isVar();

if ( !r ){

r = isInt();

if ( !r ){ printf("Error! Wrong symbol: %c \n", src[0]); }

}

}

return r;

}

bool isTerm(){

bool r;

r = isMult();

if ( r ){

while ( ( src[0] == '\*' || src[0] == '/' ) && r ){

if ( src[0] == '\*' ){

printf("Multiplication: \'\*\' \n"); } else { printf("Division: \'/\' \n"); }

src.erase(0,1);

r = isMult();

}

}

return r;

}

bool isArExp(){

bool r;

r = isTerm();

if ( r ){

while ( ( src[0] == '+' || src[0] == '-' ) && r ){

if ( src[0] == '+' ){

printf("Addition: \'+\' \n"); } else { printf("Subtraction: \'-\' \n"); }

src.erase(0,1);

r = isTerm();

}

}

return r;

}

bool isLogExp(){

bool r;

r = isArExp();

if ( r ){

if ( src[0] == '>' || src[0] == '<' ){

if ( src[1] == '=' ){

if ( src[0] == '>' ){

printf("Greater or equal than: \'>=\' \n"); } else { printf("Less or equal than: \'<=\' \n"); }

src.erase(0, 2);

r = isArExp();

} else {

if ( src[0] == '>' ){

printf("Greater than: \'>\' \n"); } else { printf("Less then: \'<\' \n"); }

src.erase(0,1);

r = isArExp();

}

} else {

if ( ( src[0] == '=' && src[1] == '=' ) ||

( src[0] == '!' && src[1] == '=' ) ){

if ( src[0] == '!' ){

printf("Not equal: \'!=\' \n"); } else { printf("Equal: \'==\' \n"); }

src.erase(0, 2);

r = isArExp();

} else {

r = false;

printf("Error! Expected logical operator, got: %c \n", src[0]);

}

}

}

return r;

}

bool isIfExp(){

bool r;

r = isLogExp();

if ( r ){

if ( src[0] == '?' ){

printf("End of condition: '\?'\ \n");

src.erase(0,1);

r = isArExp();

if ( r ){

if ( src[0] == ':' ){

printf("Else symbol: '\:'\ \n");

src.erase(0,1);

r = isArExp();

if ( r ){

if ( src[0] == ';' ){

printf("End of statement symbol: '\;'\ \n");

src.erase(0,1);

if ( src.empty() ){

printf("Statement is correct! \n");

} else {

r = false;

printf("Error! Expected end of statement, got %c \n", src[0]);

}

} else {

r = false;

printf("Error! Expected \';\', got %c \n", src[0]);

}

}

} else {

r = false;

printf("Error! Expected \':\', got %c \n", src[0]);

}

}

} else {

r = false;

printf("Error! Expected \'?\', got %c \n", src[0]);

}

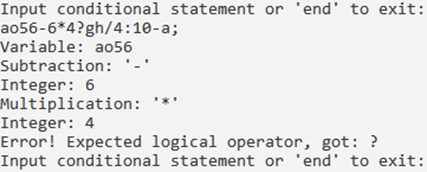
}

return r;

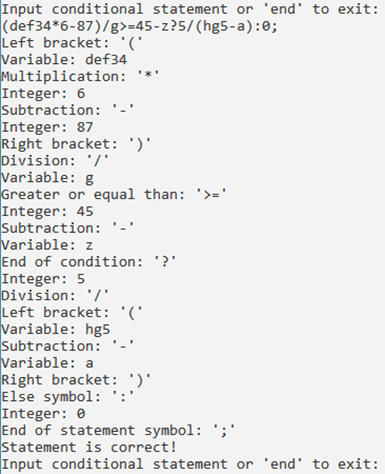
}

**Тестирование:**

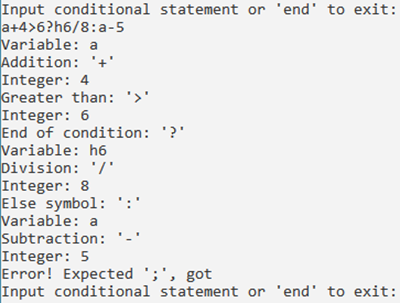
1)



2)



3)



**Выводы:** Изучен способ программирования простейших грамматических и лексических анализаторов. На языке С++ реализован нисходящий рекурсивный распознаватель LL(2)-грамматики.