Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий
институт
Кафедра «Информатика»
кафедра

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №5

«Синтаксический анализ контекстно-свободных языков»
тема

Вариант 1

Преподаватель _____ А.С. Кузнецов
подпись, дата инициалы, фамилия

Студент КИ18-17/2б _____ А.С. Ядров
номер группы подпись, дата инициалы, фамилия

Красноярск 2021

1 Цель работы

Исследование контекстно-свободных грамматик и алгоритмов синтаксического анализа контекстно-свободных языков.

2 Задание

Для выполнения данной практической работы необходимо выполнить следующие задания:

– ознакомиться с теоретическими сведениями об LL(1)-грамматиках и SLR(1)-грамматиках;

– получить у преподавателя собственный вариант задания с описанием трех контекстно-свободных языков, синтаксис которых должен быть описан создаваемыми LL(1) и SLR(1)-грамматиками, а цепочки, принадлежащие языку, распознаваемы соответствующими алгоритмами;

– используя изученные механизмы, разработать для первого заданного языка в системе JFLAP согласно постановке задачи соответствующую КСГ. В случае невозможности создания КСГ это должно доказываться формально;

– предложить программную реализацию синтаксического анализатора методом рекурсивного спуска для второго заданного языка. Необходимо провести формальное доказательство принадлежности либо непринадлежности к классу LL(1)-грамматики, лежащей в основе разработанного синтаксического анализатора;

– используя изученные механизмы, разработать для третьего заданного языка в системе JFLAP соответствующую КСГ. Невозможность - доказывается формально;

– написать отчет и представить его к защите, которая может осуществляться как в аудитории, так и дистанционно.

Для задания были взяты следующие задания:

Часть 1. Вариант 1. Язык оператора присваивания, в правой части которого задано арифметическое выражение. Элементами выражений являются

целочисленные константы в двоичной системе счисления, имена переменных из одного символа (от a до f), знаки операций и скобки для изменения порядка вычисления подвыражений. Операции (в сторону уменьшения приоритета): унарный минус, мультипликативные, аддитивные, присваивание.

Часть 2. Вариант 1. Язык арифметических выражений, элементами которых являются целочисленные константы в двоичной, восьмеричной или десятичной системах счисления, имена переменных из 1-2 символов, знаки операций и скобки для изменения порядка вычисления подвыражений. Операции (в сторону уменьшения приоритета): унарный минус, мультипликативные, аддитивные, присваивание.

Часть 3. Вариант 1. Элементами арифметического выражения являются целочисленные константы в 2- и 10-чной системах счисления, имена переменных из одного символа (от a до f), знаки операций и скобки для изменения порядка вычисления подвыражений. Операции (в сторону уменьшения приоритета): унарный минус, мультипликативные, возведение в степень, аддитивные, присваивание.

3 Ход работы

Часть 1. В ходе выполнения лабораторной работы была получена LL(1)-грамматика следующего вида (рисунок 1).

LHS		
S	⟶	aR
S	⟶	bR
S	⟶	cR
S	⟶	dR
S	⟶	eR
S	⟶	fR
R	⟶	=E
E	⟶	TJ
J	⟶	+TJ
J	⟶	-TJ
J	⟶	λ
T	⟶	FU
U	⟶	*FU
U	⟶	/FU
U	⟶	λ
F	⟶	{E}
F	⟶	1Z
Z	⟶	1Z
Z	⟶	0Z
Z	⟶	λ
E	⟶	-TJ
F	⟶	0
F	⟶	a
F	⟶	b
F	⟶	c
F	⟶	d
F	⟶	e
F	⟶	f

Рисунок 1 – LL(1)

После построения LL(1)-грамматики для заданного варианта был произведён LL(1) parse, результаты которого можно видеть ниже (рисунок 2).

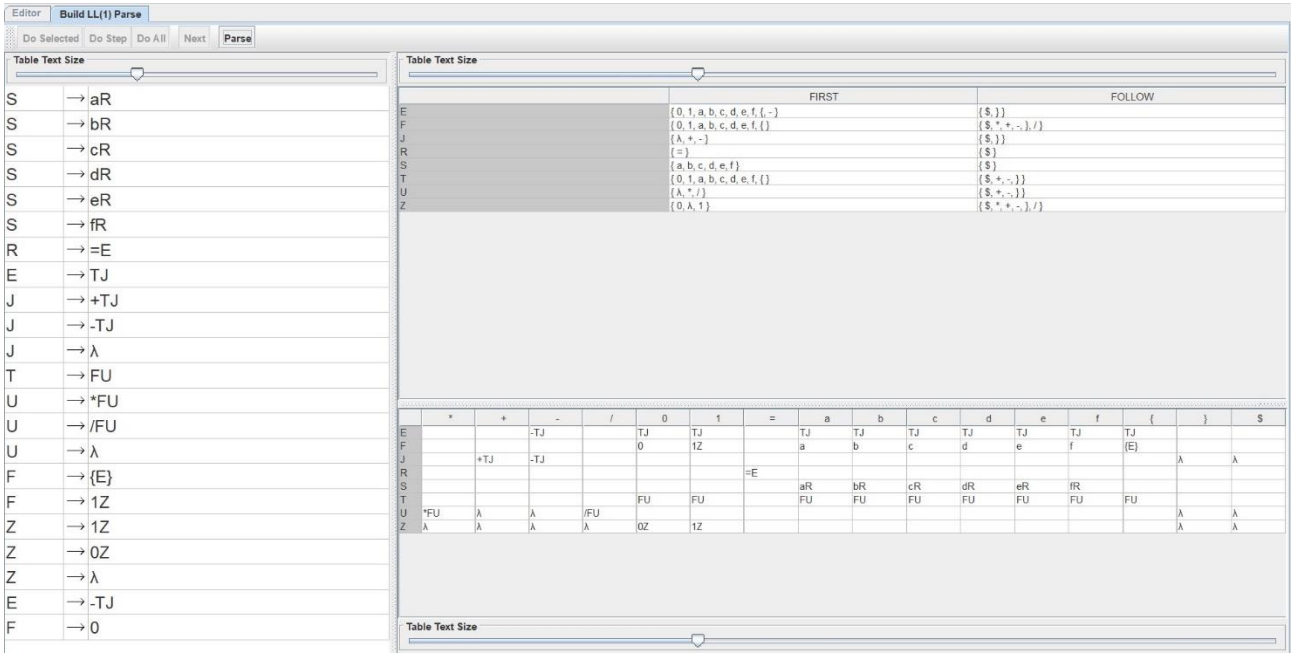


Рисунок 2 – Множество первых порождаемых символов и символов последователей и таблица синтаксического анализа

Следующим шагом было необходимо перехватить экраны распознавания тестовых цепочек автомата, созданного на основе полученной LL(1)-грамматики (не менее шести примеров) (рисунки 3-8).

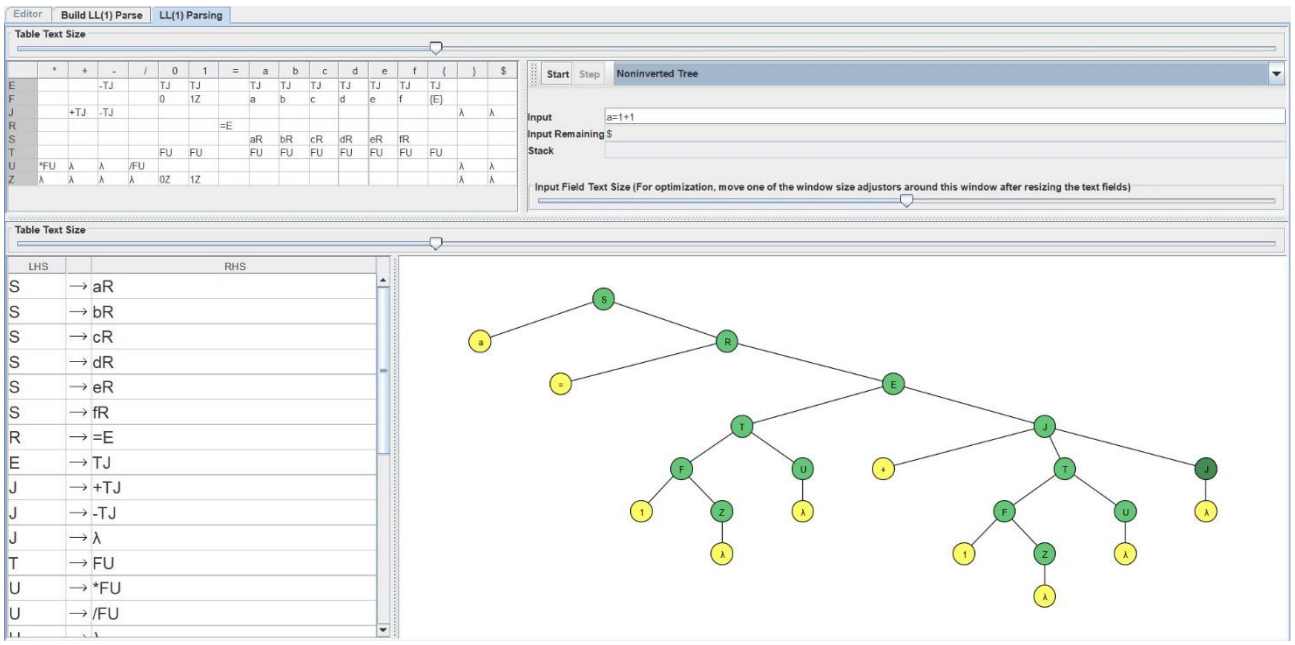


Рисунок 3 – Цепочка «a=1+1»

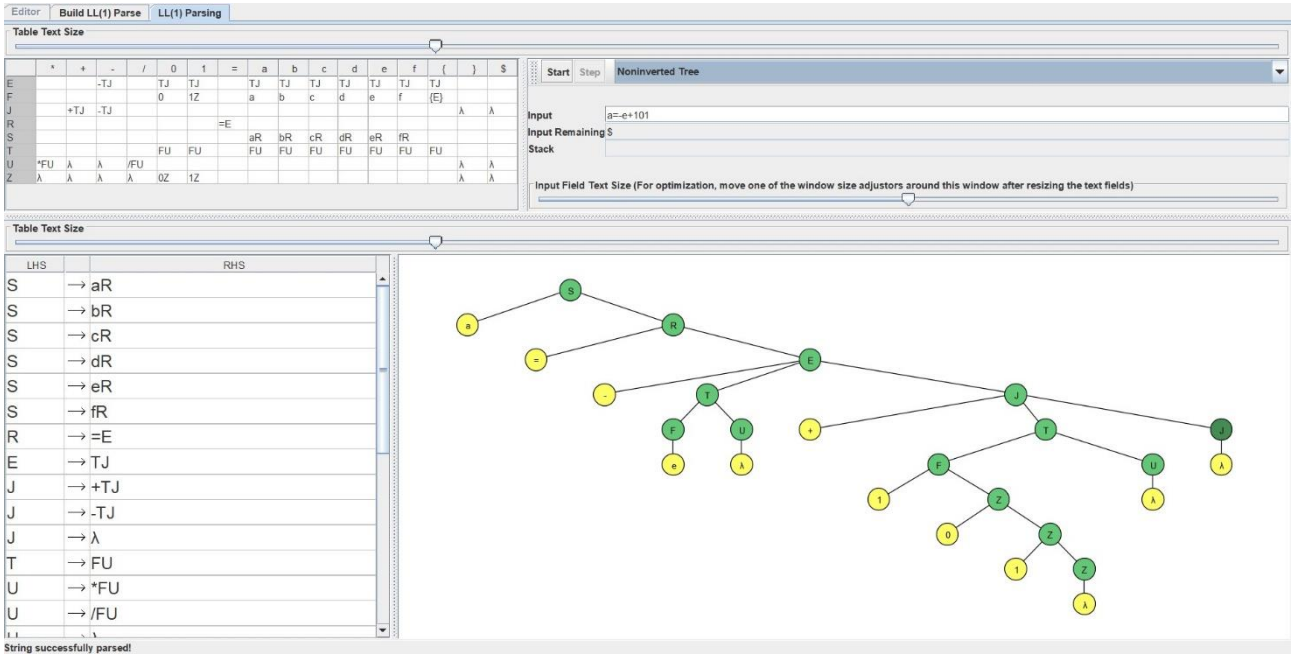


Рисунок 4 – Цепочка «a=-e+101»

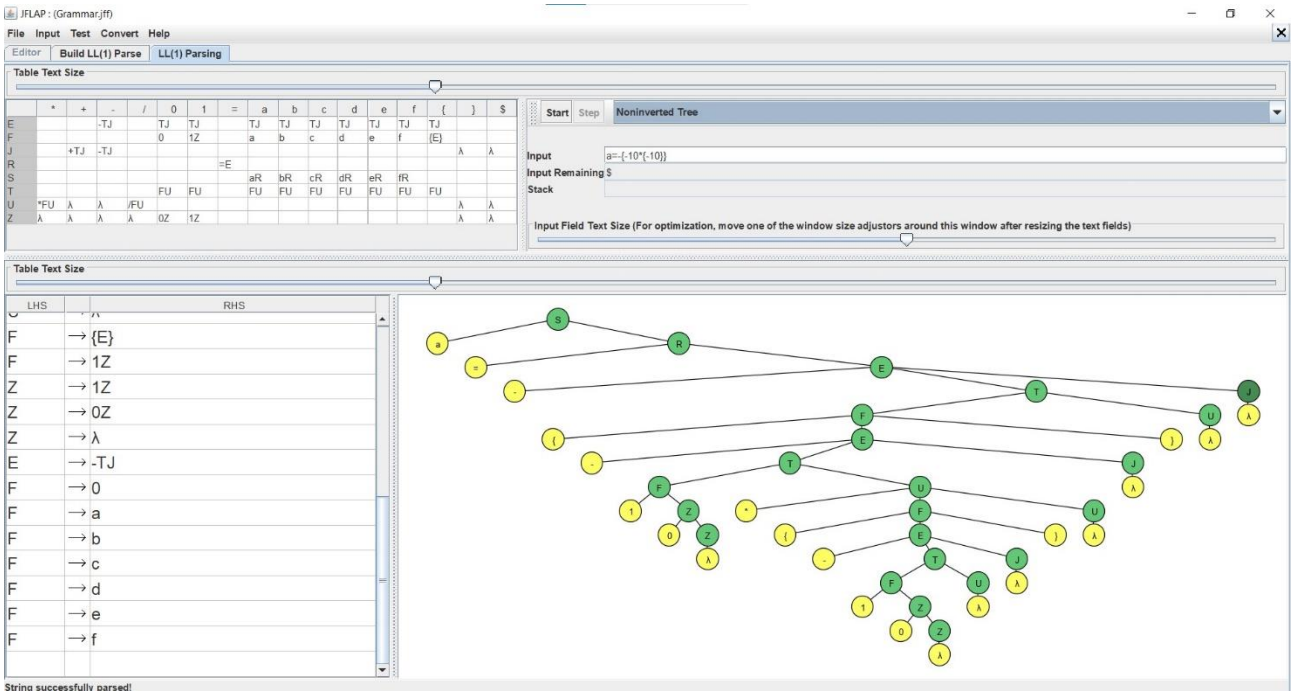


Рисунок 5 – Цепочка «a=-{-10*{-10}}»

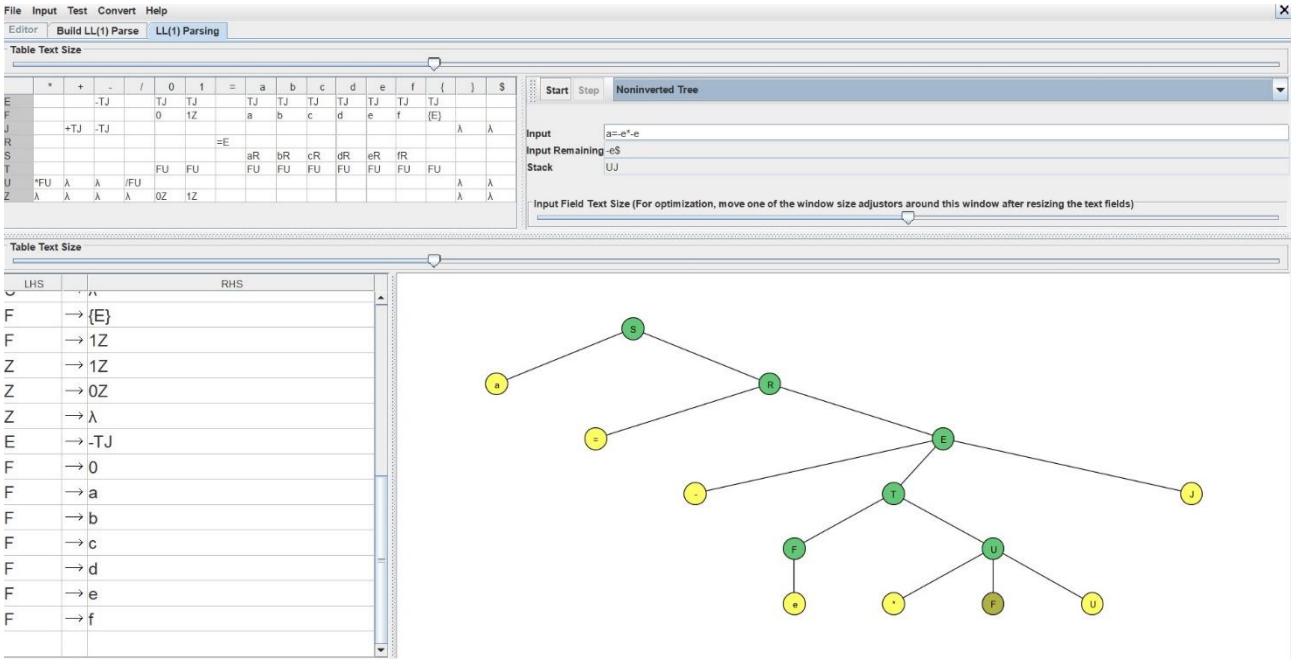


Рисунок 6 – Цепочка a=-e*-e

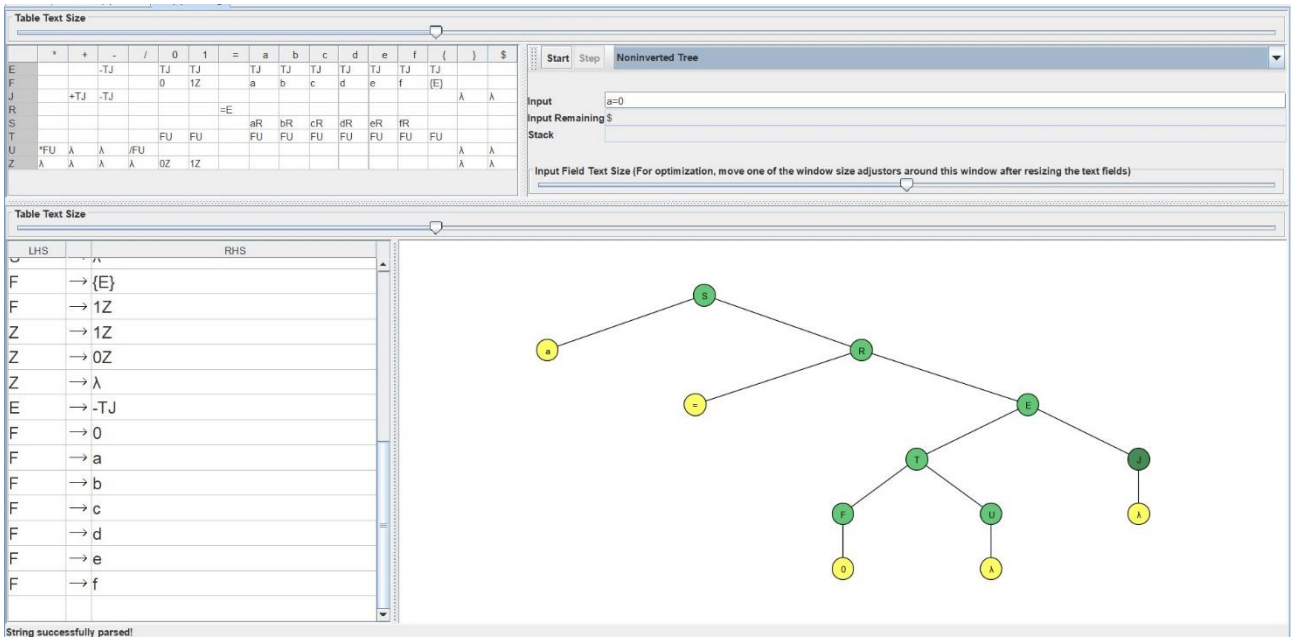


Рисунок 7 – Цепочка «a=0»

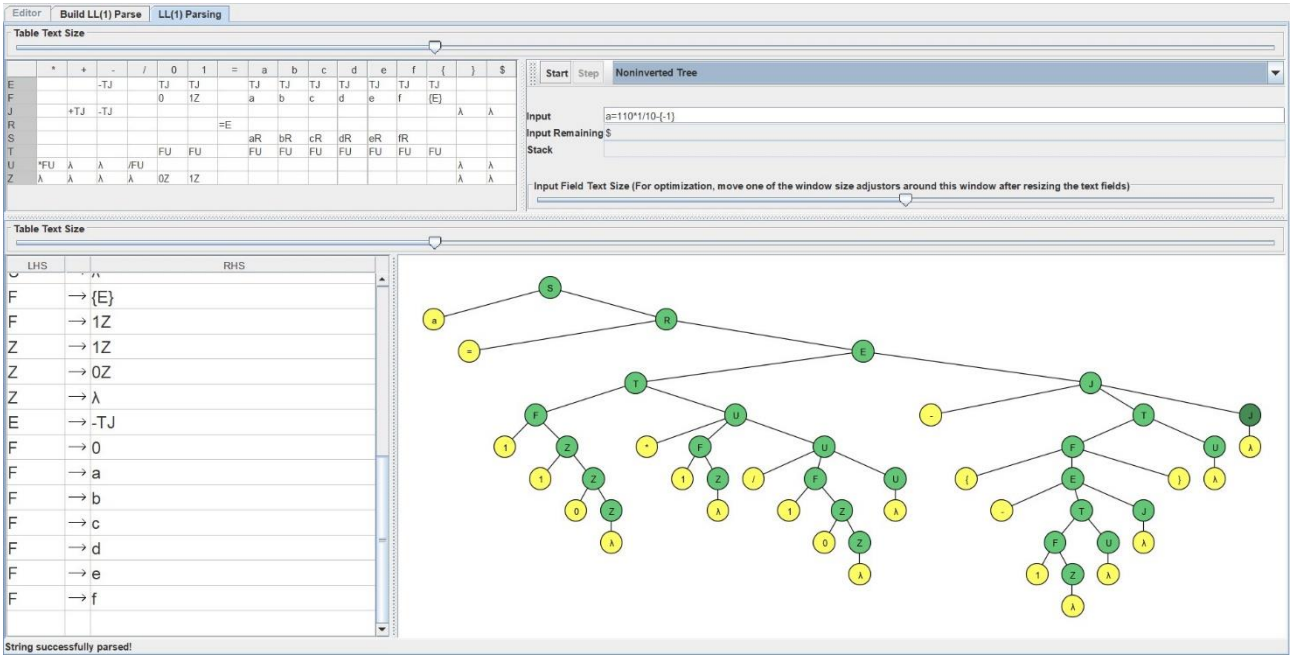


Рисунок 8 – Цепочка «a=110*1/10-{-1}»

Часть 2. В ходе выполнения лабораторной работы был реализован алгоритм рекурсивного спуска для данного варианта. Результаты были удовлетворительны (рисунки 9-10).

```
C:\Users\User\OneDrive\Рабочий стол\читы для дота\University\Теория и разработка языков программирование 6 семестр\ТРЯП5
>a.exe a=1
Accept.

C:\Users\User\OneDrive\Рабочий стол\читы для дота\University\Теория и разработка языков программирование 6 семестр\ТРЯП5
>a.exe a = 1
Reject.

C:\Users\User\OneDrive\Рабочий стол\читы для дота\University\Теория и разработка языков программирование 6 семестр\ТРЯП5
>a.exe "a = 1"
Accept.

C:\Users\User\OneDrive\Рабочий стол\читы для дота\University\Теория и разработка языков программирование 6 семестр\ТРЯП5
>a.exe "a = 1  -(-20)"
Accept.

C:\Users\User\OneDrive\Рабочий стол\читы для дота\University\Теория и разработка языков программирование 6 семестр\ТРЯП5
>a.exe "ac = 1  -(-20)"
Accept.

C:\Users\User\OneDrive\Рабочий стол\читы для дота\University\Теория и разработка языков программирование 6 семестр\ТРЯП5
>a.exe "acd = 1  -(-20)"
Reject.

C:\Users\User\OneDrive\Рабочий стол\читы для дота\University\Теория и разработка языков программирование 6 семестр\ТРЯП5
>a.exe "acd = 0  -(-20)"
Reject.

C:\Users\User\OneDrive\Рабочий стол\читы для дота\University\Теория и разработка языков программирование 6 семестр\ТРЯП5
>a.exe "ac = 0  -(-20)"
Accept.

C:\Users\User\OneDrive\Рабочий стол\читы для дота\University\Теория и разработка языков программирование 6 семестр\ТРЯП5
>a.exe "ac = -47 + 21 *(-ec)"
Accept.

C:\Users\User\OneDrive\Рабочий стол\читы для дота\University\Теория и разработка языков программирование 6 семестр\ТРЯП5
>
```

Рисунок 9 – Тестирование конечного продукта ч.1

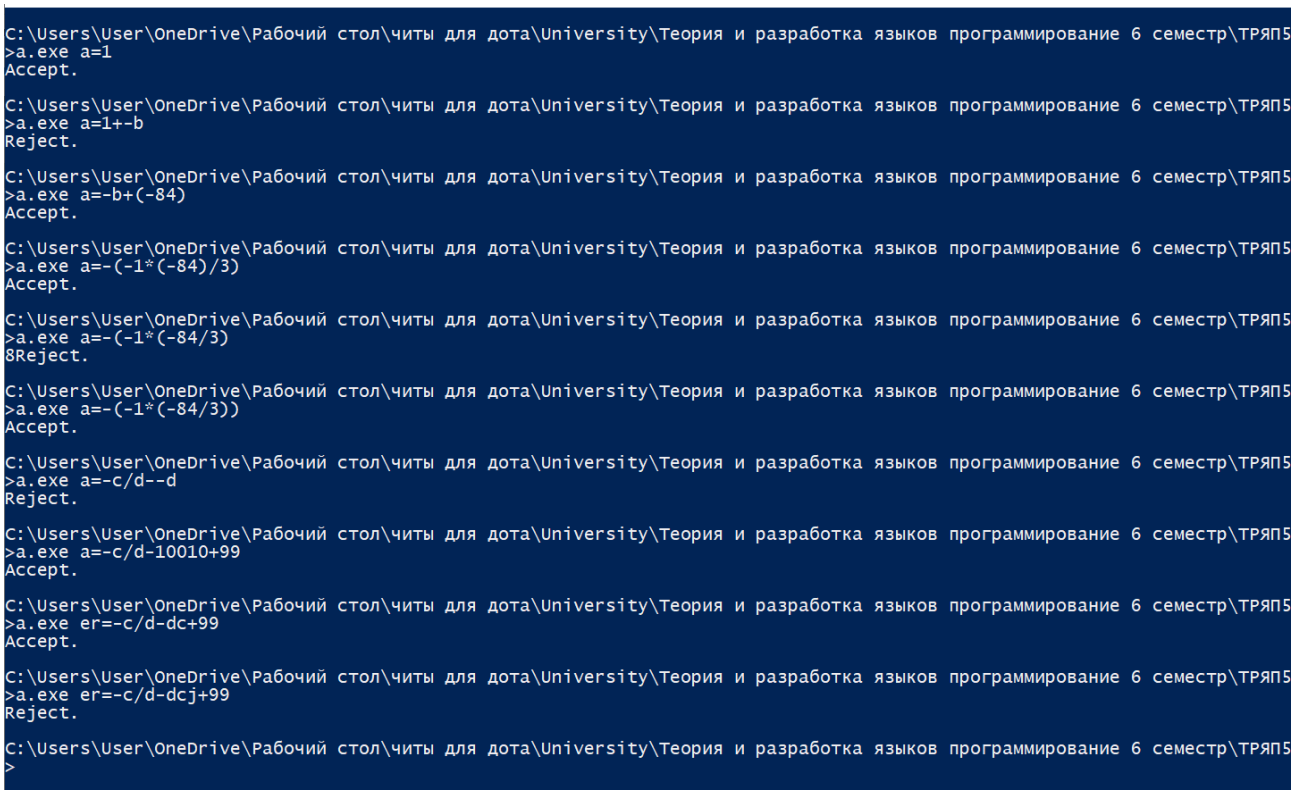


Рисунок 10 – Тестирование конечного продукта ч.2

Данная грамматика, которая была реализована является LL(1) грамматикой, о чем можно сделать выбор, исходя из таблицы 1.

Таблица 1 – Множество FIRST-FOLLOW символов

	FIRST	FOLLOW
S	{ variance }	{ $ }
R	{ = }	{ $ }
A	{ - }	{ $, ;,) }
B	{ variance, number, decimal_number, octal_number, (}	{ $, *, /, %, +, -,) }
J	{ + , - , λ }	{ $, ;,) }
P	{ *, /, %, +, -,) }	{ $, ; }

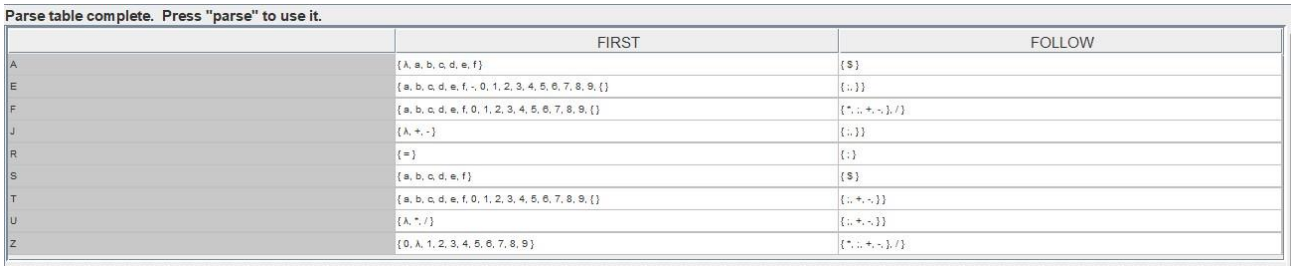
Часть 3. В ходе выполнения лабораторной работы была получена SLR(1)-грамматика следующего вида (рисунок 11).

LHS		
S	⟶	aR;A
S	⟶	bR;A
S	⟶	cR;A
S	⟶	dR;A
S	⟶	eR;A
S	⟶	fR;A
R	⟶	=E
E	⟶	TJ
J	⟶	+TJ
J	⟶	-TJ
J	⟶	λ
T	⟶	FU
U	⟶	*FU
U	⟶	/FU
U	⟶	λ
F	⟶	{E}
F	⟶	1Z
Z	⟶	1Z
Z	⟶	0Z
Z	⟶	λ
E	⟶	-TJ
F	⟶	0
F	⟶	a
F	⟶	b
F	⟶	c
F	⟶	d
F	⟶	e

LHS		
F	⟶	d
F	⟶	e
F	⟶	f
F	⟶	2Z
F	⟶	3Z
F	⟶	4Z
F	⟶	5Z
F	⟶	6Z
F	⟶	7Z
F	⟶	8Z
F	⟶	9Z
Z	⟶	2Z
Z	⟶	3Z
Z	⟶	4Z
Z	⟶	5Z
Z	⟶	6Z
Z	⟶	7Z
Z	⟶	8Z
Z	⟶	9Z
A	⟶	aR;A
A	⟶	bR;A
A	⟶	cR;A
A	⟶	dR;A
A	⟶	eR;A
A	⟶	fR;A
A	⟶	λ

Рисунок 11 – SLR(1) Grammar

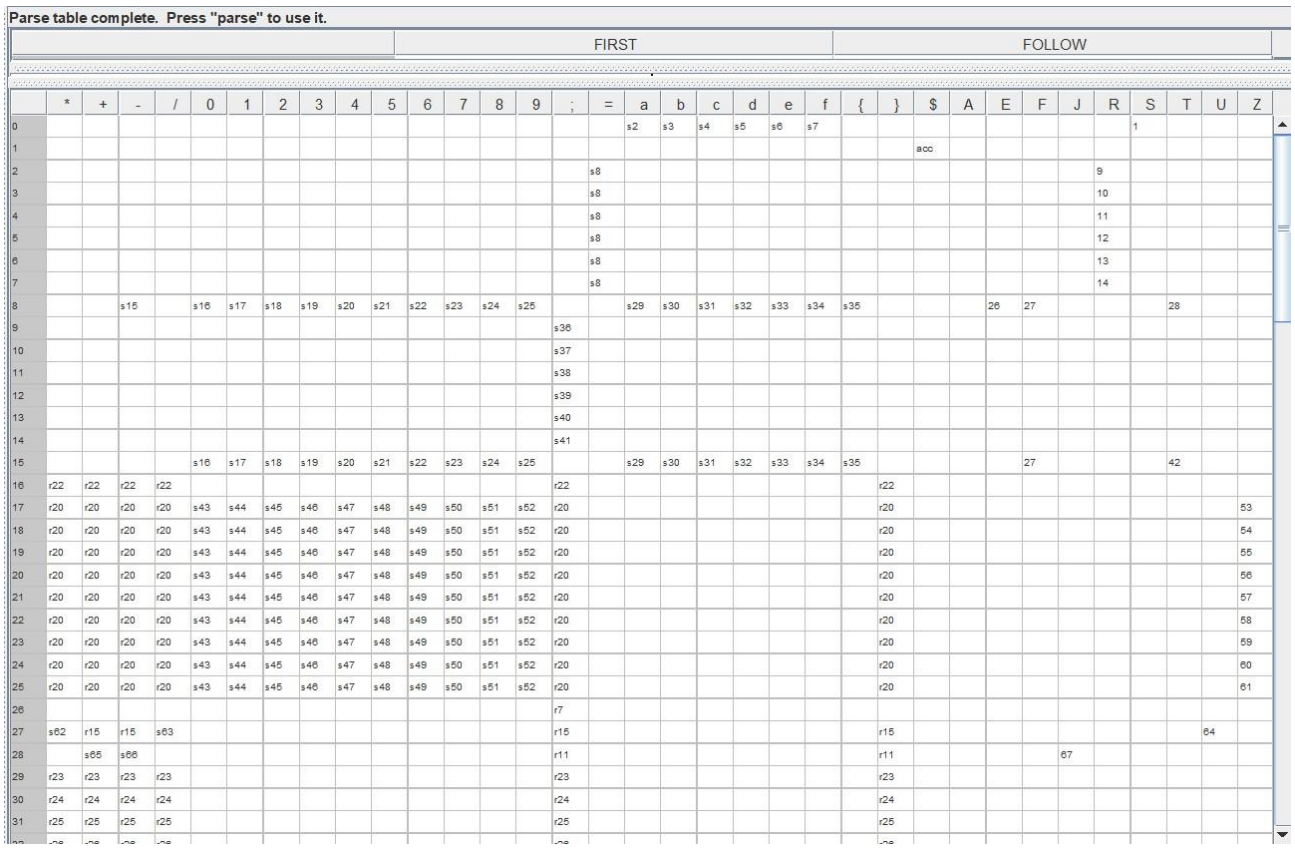
Следующим шагом было необходимо произвести SLR(1)-Parse (рисунки 12-16).



Parse table complete. Press "parse" to use it.

	FIRST	FOLLOW
A	{ A, a, b, c, d, e, f }	{ $ }
E	{ a, b, c, d, e, f, -, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, (}	{ ;,) }
F	{ a, b, c, d, e, f, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, (}	{ *, ;, +, -,), / }
J	{ A, +, - }	{ ;,) }
R	{ = }	{ ; }
S	{ a, b, c, d, e, f }	{ $ }
T	{ a, b, c, d, e, f, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, (}	{ ;, +, -,) }
U	{ A, *, / }	{ ;, +, -,) }
Z	{ 0, A, 1, 2, 3, 4, 5, 6, 7, 8, 9 }	{ *, ;, +, -,), / }

Рисунок 12 – Множество первых порождаемых символов и символов последователей



Parse table complete. Press "parse" to use it.

	FIRST																				FOLLOW														
	*	+	-	/	0	1	2	3	4	5	6	7	8	9	;	=	a	b	c	d	e	f	{	}	$	A	E	F	J	R	S	T	U	Z
0																	s2	s3	s4	s5	s6	s7								1				
1																									acc									
2																s8													9					
3																s8													10					
4																s8													11					
5																s8													12					
6																s8													13					
7																s8													14					
8			s15		s16	s17	s18	s19	s20	s21	s22	s23	s24	s25			s29	s30	s31	s32	s33	s34	s35				26	27				28			
9															s36																				
10															s37																				
11															s38																				
12															s39																				
13															s40																				
14															s41																				
15					s16	s17	s18	s19	s20	s21	s22	s23	s24	s25			s29	s30	s31	s32	s33	s34	s35					27				42			
16	r22	r22	r22	r22											r22									r22										
17	r20	r20	r20	r20	s43	s44	s45	s46	s47	s48	s49	s50	s51	s52	r20									r20										53
18	r20	r20	r20	r20	s43	s44	s45	s46	s47	s48	s49	s50	s51	s52	r20									r20										54
19	r20	r20	r20	r20	s43	s44	s45	s46	s47	s48	s49	s50	s51	s52	r20									r20										55
20	r20	r20	r20	r20	s43	s44	s45	s46	s47	s48	s49	s50	s51	s52	r20									r20										56
21	r20	r20	r20	r20	s43	s44	s45	s46	s47	s48	s49	s50	s51	s52	r20									r20										57
22	r20	r20	r20	r20	s43	s44	s45	s46	s47	s48	s49	s50	s51	s52	r20									r20										58
23	r20	r20	r20	r20	s43	s44	s45	s46	s47	s48	s49	s50	s51	s52	r20									r20										59
24	r20	r20	r20	r20	s43	s44	s45	s46	s47	s48	s49	s50	s51	s52	r20									r20										60
25	r20	r20	r20	r20	s43	s44	s45	s46	s47	s48	s49	s50	s51	s52	r20									r20										61
26															r7																				
27	s62	r15	r15	s63											r15									r15									64		
28		s65	s66												r11									r11					67						
29	r23	r23	r23	r23											r23									r23										
30	r24	r24	r24	r24											r24									r24										
31	r25	r25	r25	r25											r25									r25										

Рисунок 13 – Таблица синтаксического анализа ч.1

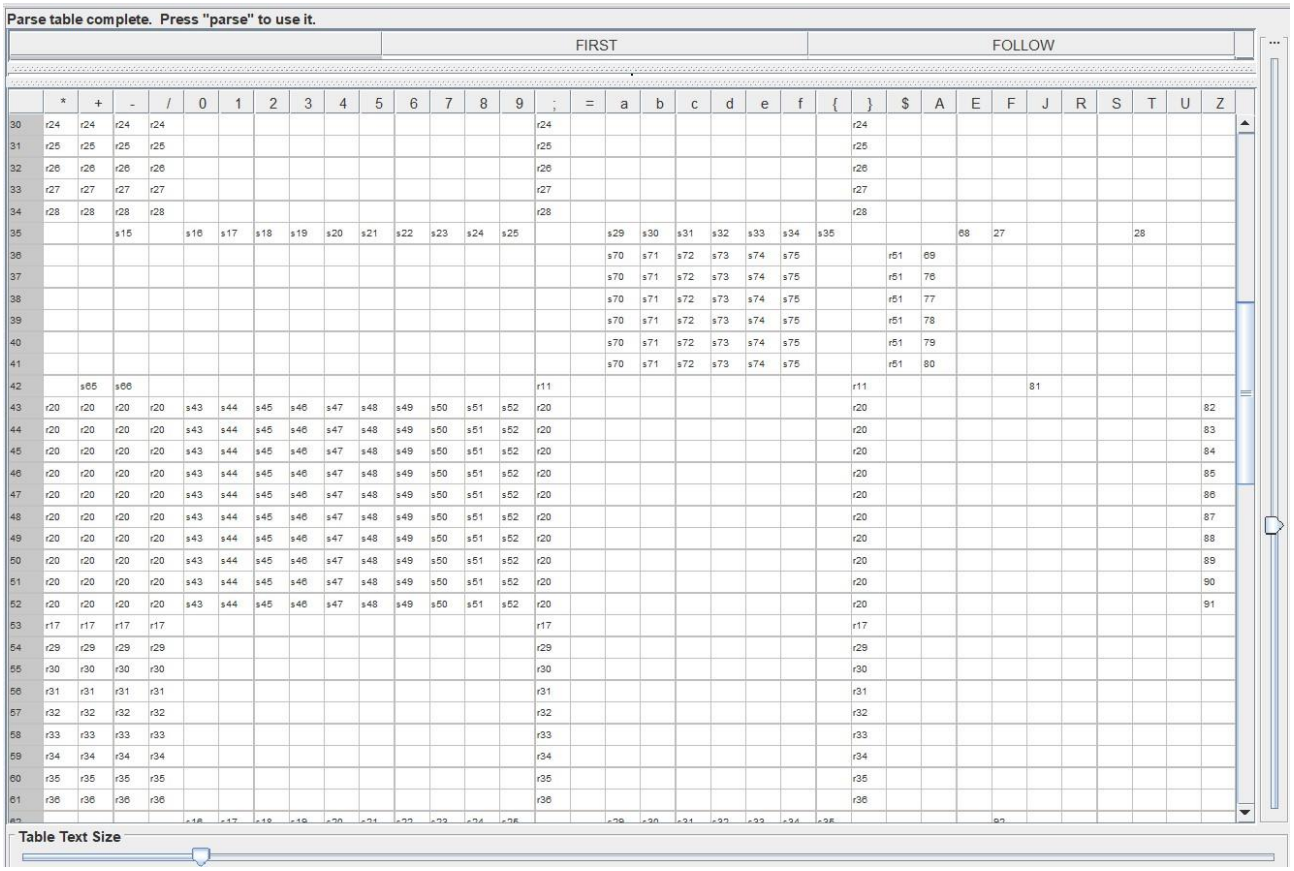


Рисунок 14 – Таблица синтаксического анализа ч.2

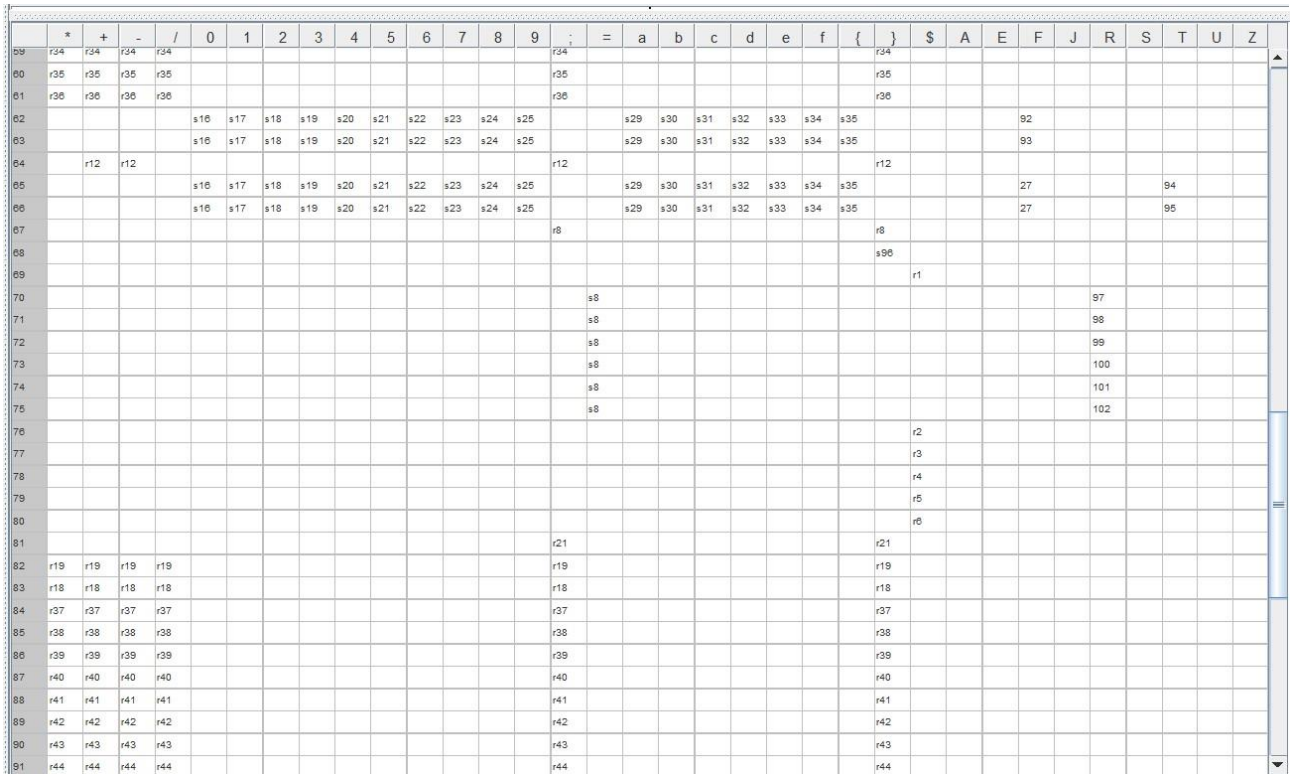


Рисунок 15 – Таблица синтаксического анализа ч.3

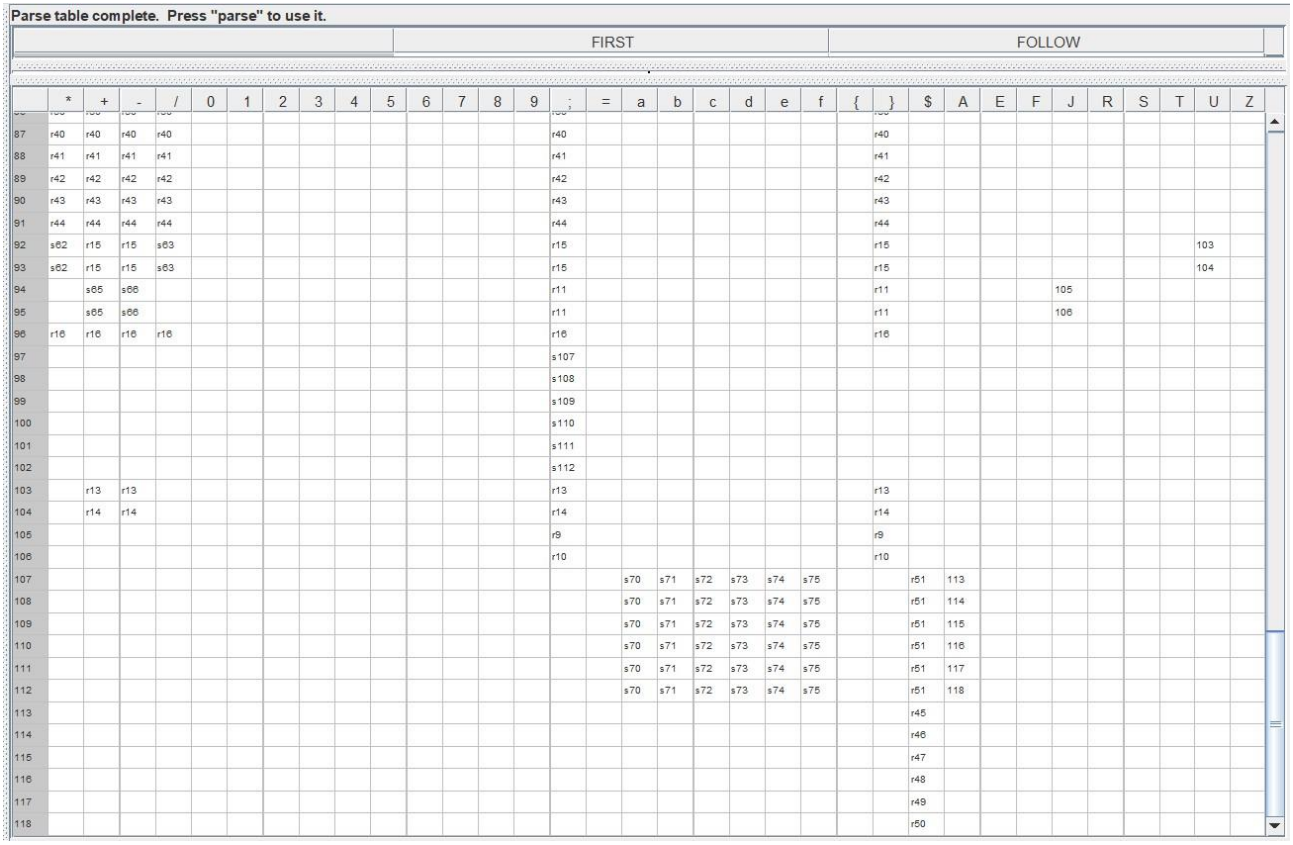


Рисунок 16 – Таблица синтаксического анализа ч.4

Следующим шагом было необходимо перехватить экраны распознавания тестовых цепочек автомата, созданного на основе полученной SLR(1)-грамматики (не менее шести примеров) (рисунок 17-22).

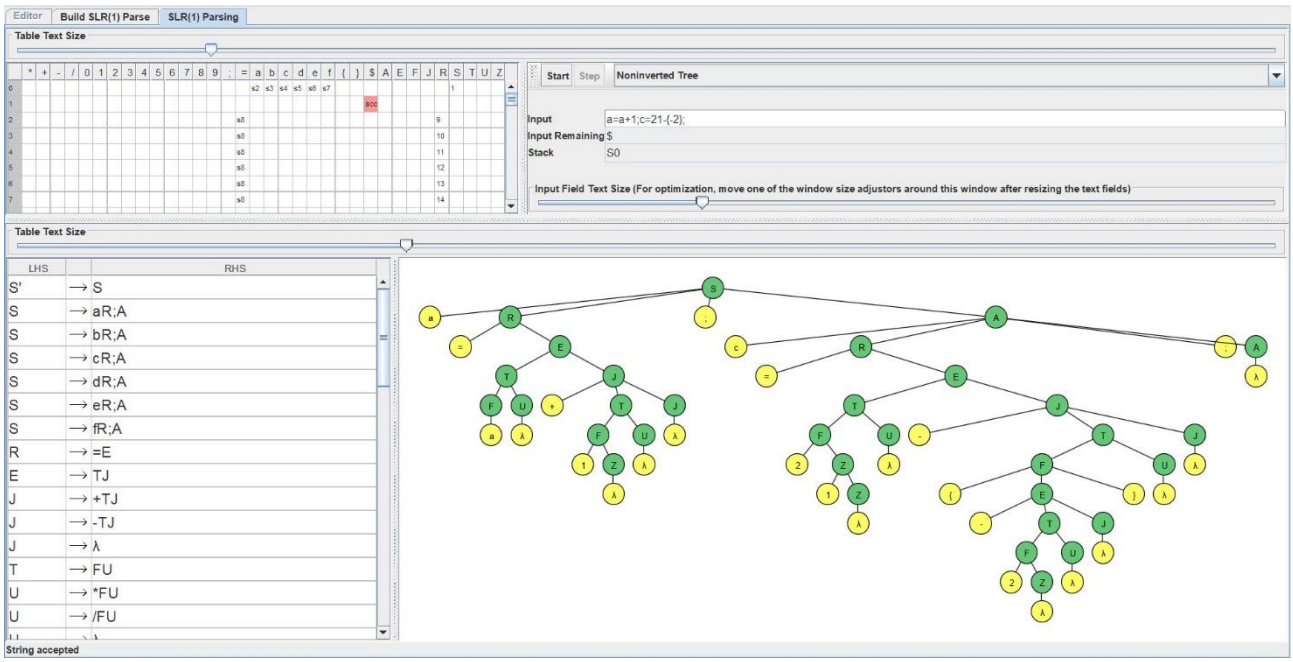


Рисунок 17 – Цепочка «a=a+1;c=21-{-2};»

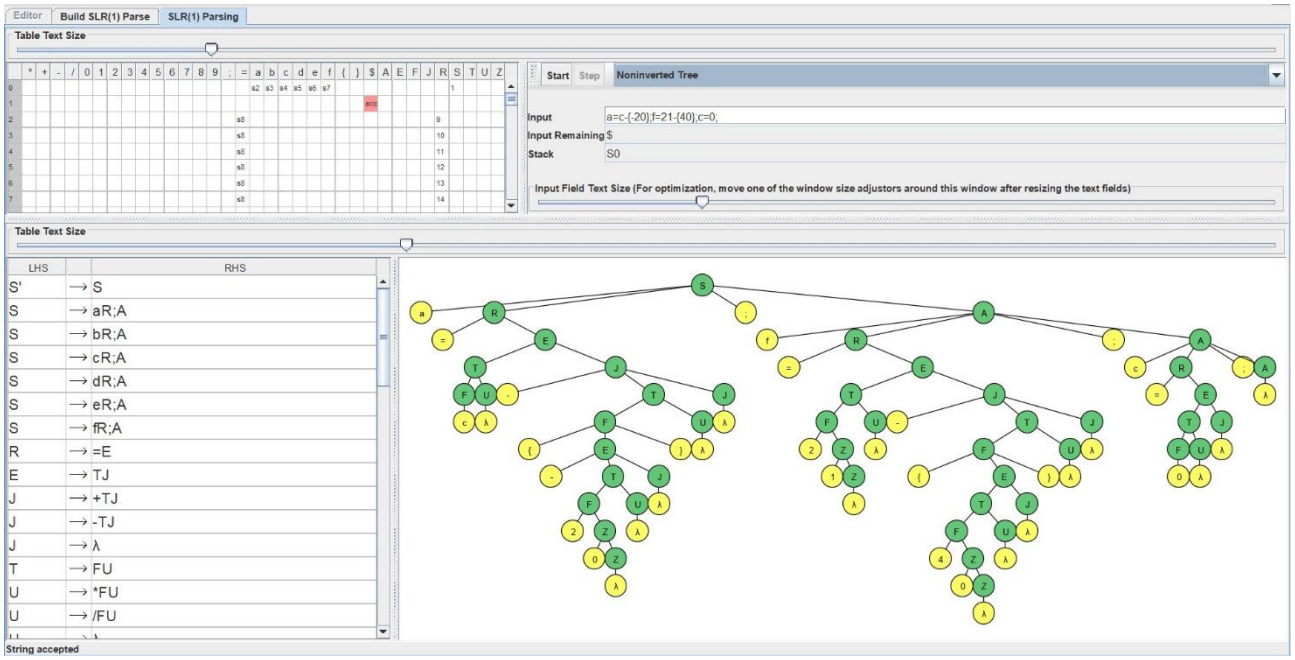


Рисунок 18 – Цепочка «a=c-{-20};f=21-{40};c=0;»

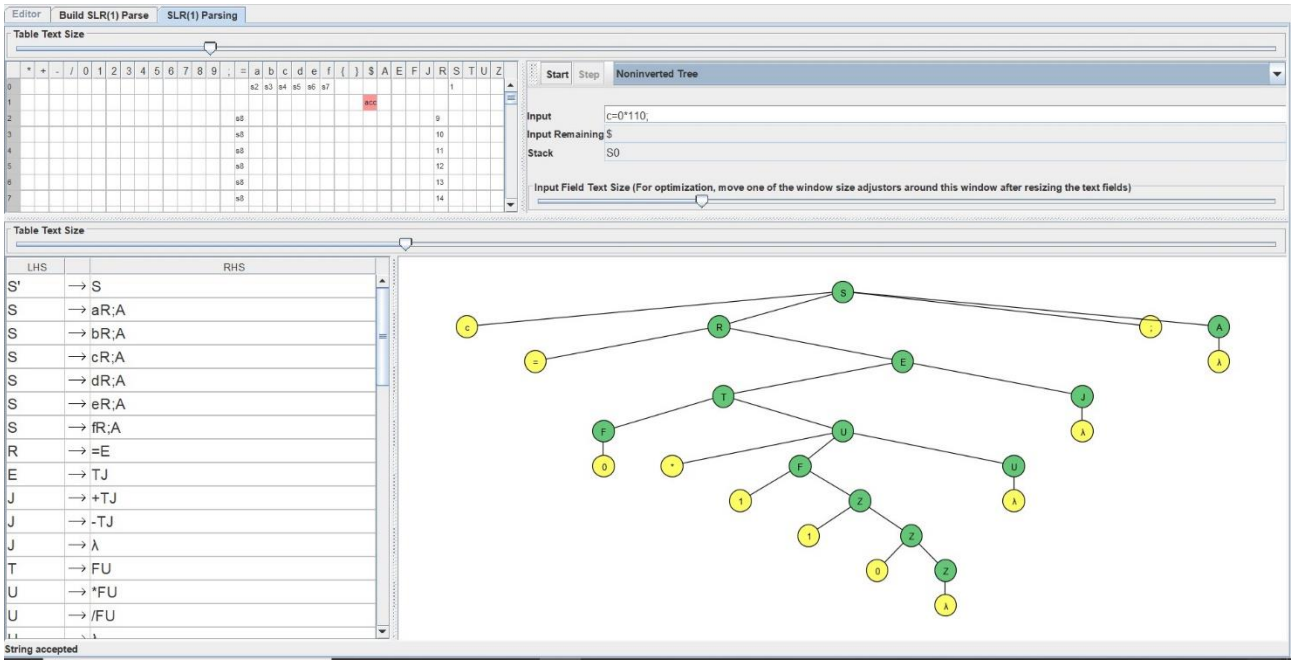


Рисунок 19 – Цепочка «c=0*110;»

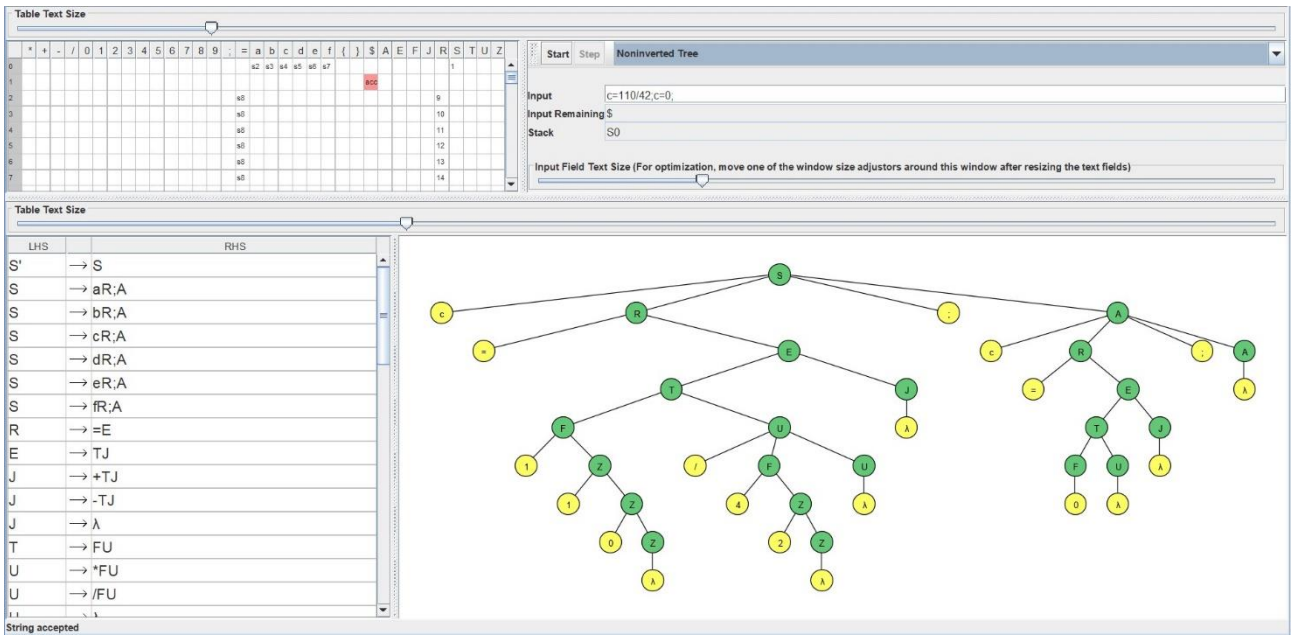


Рисунок 20 – Цепочка «c=110/42;c=0»

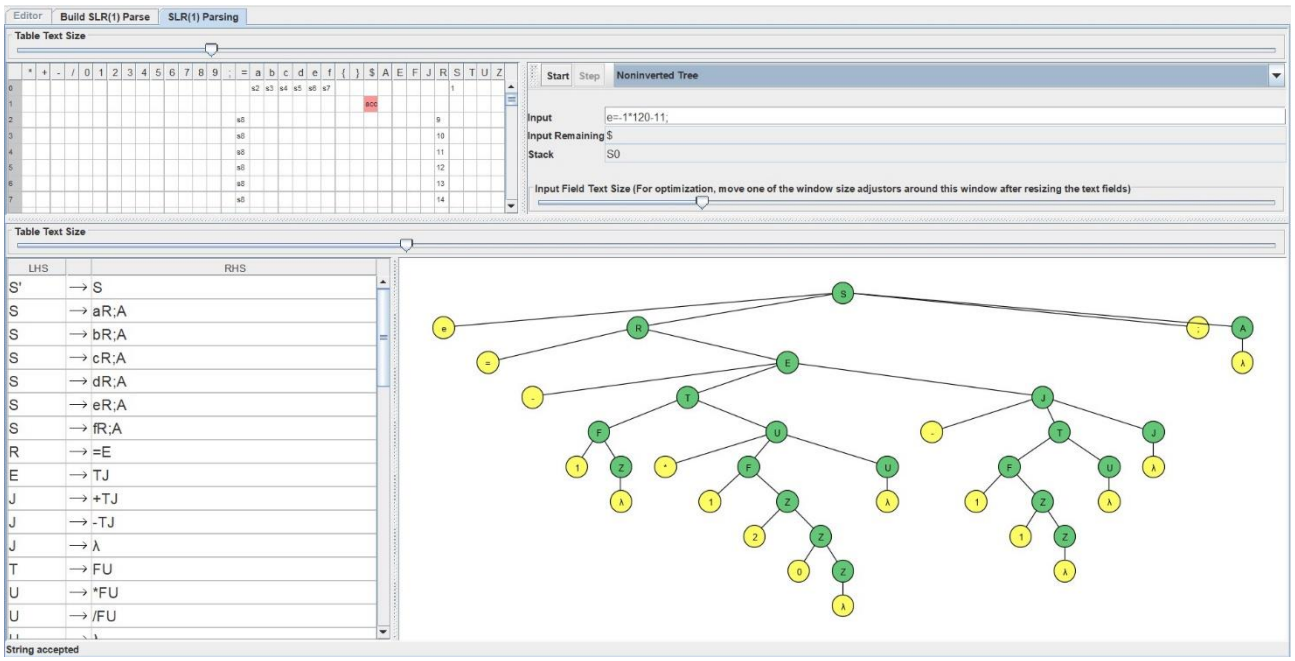


Рисунок 21 – Цепочка «e=-1*120-11;»

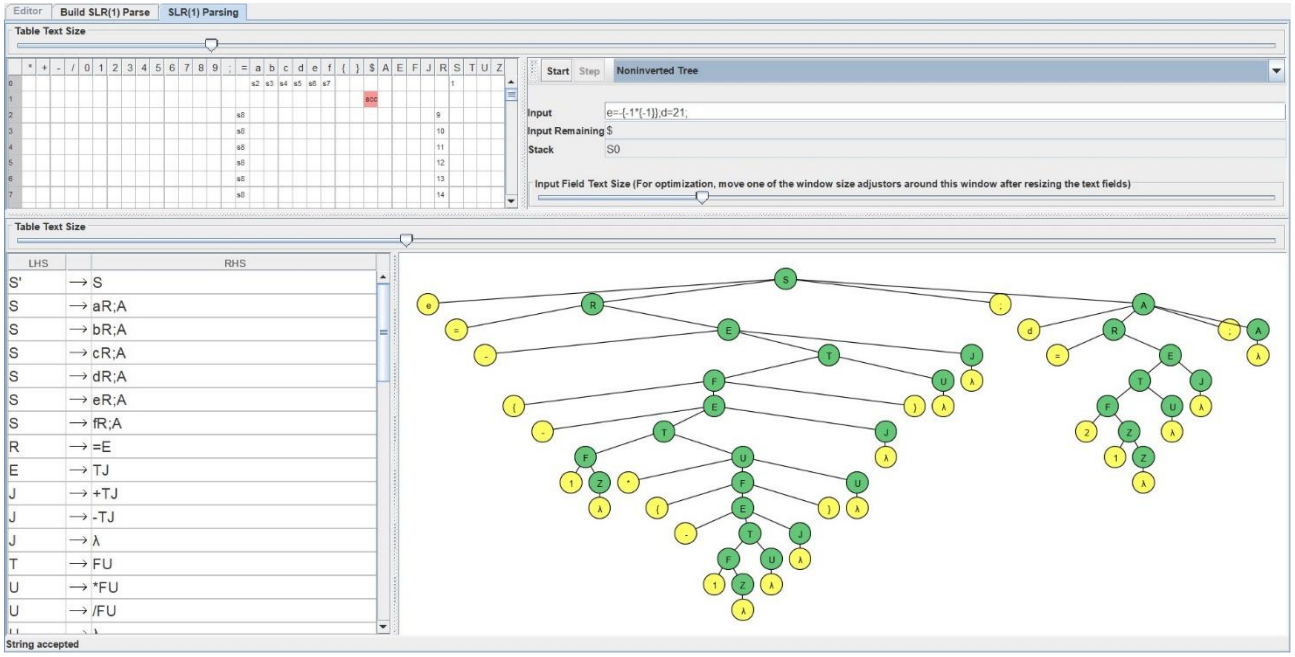


Рисунок 22 – Цепочка «e=-{-1*{-1}};d=21;»

4 Вывод

В ходе выполнения лабораторной работы было проведено исследование контекстно-свободных грамматик и алгоритмов синтаксического анализа контекстно-свободных языков.

ПРИЛОЖЕНИЕ

Листинг 1 – RecursiveDescent.cpp

```
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
#include <string.h>

/* Terminals */
#define variance       0
#define equal          1
#define minus          2
#define number         3
#define zero           4
#define sign           5
#define left_bracket   6
#define right_bracket  7
#define decimal_number 8
#define octal_number   9
#define delimiter     10


/* EOP */
#define EOP           11

#define UNDEF         12

/* Lexeme class */
int lexeme = 0;

/* Non terminals */
void S(), R(), A(), B(), J(), P(), G(), K();

/* Checking binary of string which contains number */
bool IsBinary(char* s);

/* Checking octal of string which contains number */
bool IsOctal(char* s);

/* Reject input line */
void error();
```

```
    /* Look ahead lexeme */
    int get_token();

    char g_inputBuffer[1024] = "";
    char* g_prog = NULL;

    int main(int argc, char* argv[])
    {
        if (1 == argc)
        {
            printf("Enter  your  input  line  (delimiter  betweens  expressions  =
';'): ");
            gets(g_inputBuffer);
        }
        else
        {
            strcpy(g_inputBuffer, argv[1]);
        }

        g_prog = (char*)g_inputBuffer;

        lexeme = get_token();
        S();
        lexeme = get_token();
        if (lexeme == EOP)
            printf("Accept.\n");
        else
            printf("Reject.\n");
    }

    void S()
    {
        if (lexeme != variance)
            error();

        lexeme = get_token();
        R();

        if (lexeme != delimiter)
           error();

        lexeme = get_token();
```

```
    if (lexeme != EOP)
        S();
}

void R()
{
    if (lexeme != equal)
    {
        error();
    }
    lexeme = get_token();
    if (lexeme == minus)
        A();
    else
        B();
}

void A()
{
    if (lexeme != minus)
    {
        error();
    }
    lexeme = get_token();
    B();
}

void B()
{
    if ((lexeme == number) || (lexeme == variance) ||
            (lexeme == decimal_number) || (lexeme == octal_number))
    {
        lexeme = get_token();
        if (lexeme != EOP && (lexeme == sign || lexeme == minus))
        {
            J();
        }
    }
    else if (lexeme == left_bracket)
    {
        lexeme = get_token();
        if (lexeme == minus)
            A();
```

```
        else
            B();
        P();
    }
    else
    {
        error();
    }
}

void J()
{
    if (lexeme != sign && lexeme != minus)
    {
        error();
    }
    lexeme = get_token();
    B();
}

void P()
{
    if (lexeme == sign || lexeme == minus)
    {
        lexeme = get_token();
        B();
        P();
    }
    else if (lexeme == right_bracket)
    {
        lexeme = get_token();
        if (lexeme != EOP && (lexeme == sign || lexeme == minus))
            J();
    }
    else
    {
        printf("%d", lexeme);
        error();
    }
}

int get_token()
{
```

```
char token[132] = "";
char* tok = token;

/* Skip spaces */
while (isspace(*g_prog))
    ++g_prog;

 /*End line marker*/
if (*g_prog == NULL)
     return EOP;

printf(g_prog);
printf("\n");
/* Keywords */
if (isalpha(*g_prog)) // Чтение переменной из 1-2 символов
{
    ++g_prog;
    if (isalpha(*g_prog))
    {
        ++g_prog;
    }
    return variance;
}
else if(*g_prog == '=')  // Чтение знака равенства
{
    ++g_prog;
    return equal;
}
else if(*g_prog == '-')  // Чтение унарного минуса
{
    ++g_prog;
    return minus;
}
else if (isdigit(*g_prog))  // Чтение любого числа
{

    if (*g_prog != '0')
    {
        while (isdigit(*g_prog))
        {
            *tok = *g_prog;
            ++tok;
            ++g_prog;
```

```
                }
            }
            else
            {
                *tok = *g_prog;
                ++tok;
                ++g_prog;
            }
            *tok = '\0';
            if (0 == strcmp(token, "0"))
                return number;
            if (IsBinary(tok))
                return decimal_number;
            if (IsOctal(tok))
                return octal_number;
            else
                return number;
        }
        else if ((*g_prog == '+') || (*g_prog == '*') || (*g_prog == '%') ||
(*g_prog == '/'))
        {
            ++g_prog;
            return sign;
        }
        else if (*g_prog == '(')
        {
            ++g_prog;
            return left_bracket;
        }
        else if (*g_prog == ')')
        {
            ++g_prog;
            return right_bracket;
        }
        else if (*g_prog == ';')
        {
            ++g_prog;
            return delimiter;
        }
        return UNDEF;
    }

    void error()
```

```
{
    printf("Reject.\n");
    exit(EXIT_FAILURE);
}

bool IsBinary( char* s )
{
   bool bIsBinary = true;
   int i = 0;
   while(s[i] != '\0')
   {
       if (s[i] != '0' && s[i] != '1')
           bIsBinary = false;
       i++;
   }
   return bIsBinary;
}

bool IsOctal( char* s )
{
   bool IsOctal = true;
   int i = 0;
   while(s[i] != '\0')
   {
       if (s[i] != '0' && s[i] != '1' &&
               s[i] != '2' && s[i] != '3' &&
               s[i] != '4' && s[i] != '5' &&
               s[i] != '6' && s[i] != '7')
           IsOctal = false;
       i++;
   }
   return IsOctal;
}
```