

**Лабораторная работа 16 (6 часов)**  
**Языки программирования**

**Разработка синтаксического анализатора**

1. Используйте материалы лекций № 19-21.
2. Используйте результаты лабораторных работ № 13-15.
3. Создайте проект (VS20xx, C++, консольное приложение) с именем **LPLab16**.
4. Разработайте синтаксический анализатор для языка SVV-2015 (описан в лекциях и задании к лабораторной работе 14).
5. В контрольном примере в качестве входных данных используйте таблицу лексем, полученную в лабораторной работе 14.
6. Для представления грамматики языка (в форме Грейбах) SVV-2015 создайте структуры следующей спецификации (рис.1). На рис. 2 представлен пример (фрагмент) представления грамматики для языка SVV-2015 с помощью структуры **Greibach**.

```
#pragma once
#include "Error.h"
typedef short GRBALPHABET; // символы алфавита грамматики терминалы > 0, нетерминалы < 0
namespace GRB
{
    struct Rule //правило в грамматике Грейбах
    {
        GRBALPHABET nn; // нетерминал (левый символ правила) < 0
        int iderror; // идентификатор диагностического сообщения
        short size; // количество цепочек - правых частей правила
        struct Chain // цепочка (правая часть правила)
        {
            short size; // длина цепочки
            GRBALPHABET* nt; // цепочка терминалов (>0) и нетерминалов (<0)
            Chain() {size = 0; nt = 0;};
            Chain(
                short psize, // количество символов в цепочке
                GRBALPHABET s, ... // символы (терминал или нетерминал)
            );
            char* getChain(char* b); // получить правую сторону правила
            static GRBALPHABET T(char t) {return GRBALPHABET(t);}; // терминал
            static GRBALPHABET N(char n) {return -GRBALPHABET(n);}; // не терминал
            static bool isT(GRBALPHABET s) {return s > 0;}; // терминал?
            static bool isN(GRBALPHABET s) {return !isT(s);} // нетерминал?
            static char alphabet_to_char(GRBALPHABET s) {return isT(s)?char(s):char(-s);}; // GRBALPHABET->char
        }
        Chain* chains; // массив цепочек - правых частей правила
        Rule(){nn = 0x00; size = 0;};
        Rule(
            GRBALPHABET pnn, // нетерминал (< 0)
            int iderror, // идентификатор диагностического сообщения (Error)
            short psize, // количество цепочек - правых частей правила
            Chain c, ... // множество цепочек - правых частей правила
        );
    };
}
```

Рис.1. Спецификация структуры для представления грамматики в форме Грейбах.

```

char* getCRule(          // получить правило в виде N->цепочка (для распечатки)
    char* b,            // буфер
    short nchain        // номер цепочки (правой части) в правиле
);

short Rule::getNextChain( // получить следующую за j подходящую цепочку, вернуть ее номер или -1
    GRBALPHABET t,       // первый символ цепочки
    Rule::Chain& pchain,  // возвращаемая цепочка
    short j              // номер цепочки
);

};

struct Greibach // грамматика Грейбах
{
    short size;      // количество правил
    GRBALPHABET startN; // стартовый символ
    GRBALPHABET stbottomT; // дно стека
    Rule* rules;     // множество правил
    Greibach() {short size = 0; startN = 0; stbottomT = 0; rules = 0; };
    Greibach(
        GRBALPHABET pstartN, // стартовый символ
        GRBALPHABET pstbottomT, // дно стека
        short psize, // количество правил
        Rule r, ... // правила
    );
    short getRule( // получить правило, возвращается номер правила или -1
        GRBALPHABET pnn, // левый символ правила
        Rule& prule // возвращаемое правило грамматики
    );
    Rule getRule(short n); // получить правило по номеру
};

Greibach Greibach(); // получить грамматику
};

```

Рис.1. Спецификация структуры для представления грамматики в форме Грейбах (продолжение).

```

#include "GRB.h"
#define GRB_ERROR_SERIES 600
namespace GRB
{
    #define NS(n) Rule::Chain::N(n)
    #define TS(n) Rule::Chain::T(n)
    Greibach greibach( NS('S'), TS('$'), // стартовый символ, дно стека
        6, // количество правил
        Rule(NS('S'), GRB_ERROR_SERIES + 0, // Неверная структура программы
            3, // S->m{NrE}; | t{f(F){NrE};}S | m{NrE};S | t{f(F){NrE};}
            Rule::Chain(8, TS('m'), TS('('), NS('N'), TS('r'), NS('E'), TS(';'), TS(')'), TS(')'), TS(')'),
            Rule::Chain(14, TS('t'), TS('i'), TS('f'), TS('('), NS('F'), TS(')'), TS(')'), NS('N'), TS('r'), NS('E'), TS(';'), TS(')'), TS(')'),
            Rule::Chain(9, TS('m'), TS('('), NS('N'), TS('r'), NS('E'), TS(';'), TS(')'), TS(')'), TS(')'), NS('S'))
        ),
        Rule(NS('N'), GRB_ERROR_SERIES + 1, // Ошибочный оператор
            8, // N->dti;|rE;|i=E;|dtfi(F);|dtiN|rE;N|i=E;N|dtfi(F);N
            Rule::Chain(4, TS('d'), TS('t'), TS('i'), TS(';')),
            Rule::Chain(3, TS('r'), NS('E'), TS(';')),
            Rule::Chain(4, TS('i'), TS('='), NS('E'), TS(';')),
            Rule::Chain(8, TS('d'), TS('t'), TS('f'), TS('i'), TS('('), NS('F'), TS(')'), TS(')'), TS(')'),
            Rule::Chain(5, TS('d'), TS('t'), TS('i'), TS(';'), NS('N')),
            Rule::Chain(4, TS('r'), NS('E'), TS(';'), NS('N')),
            Rule::Chain(5, TS('i'), TS('='), NS('E'), TS(';'), NS('N')),
            Rule::Chain(9, TS('d'), TS('t'), TS('f'), TS('i'), TS('('), NS('F'), TS(')'), TS(')'), TS(')'), NS('N'))
        ),
        Rule(NS('E'), GRB_ERROR_SERIES + 2, // Ошибка в выражении
            8, // E->i|i|(E)|i(W)|iM|iM|(E)M|i(W)M
            Rule::Chain(1, TS('i')),
            Rule::Chain(1, TS('l')),
            Rule::Chain(3, TS('('), NS('E'), TS(')'),
            Rule::Chain(4, TS('i'), TS('('), NS('W'), TS(')'),
            Rule::Chain(2, TS('i'), NS('M')),
            Rule::Chain(2, TS('l'), NS('M')),
            Rule::Chain(4, TS('('), NS('E'), TS(')'), NS('M')),
            Rule::Chain(5, TS('i'), TS('('), NS('W'), TS(')'), NS('M'))
        ),
    );
};

```

Рис.2. Фрагмент программы, демонстрирующий представление грамматики языка SVV-2015 с помощью структуры Greibach.

7. Табл. 1 описывает назначение структур, приведенных на рис.1 и 2.

Таблица 1. Описание структур, для представления грамматики

Структура	Описание
<b>Greibach</b>	<p>Структура: представление грамматики. Все символы (алфавит) грамматики представляются в формате <b>GRBALPHABET</b> (short). Причем терминалы – положительные значения, нетерминалы – отрицательные.</p> <p>Структура включает:</p> <ul style="list-style-type: none"> <li>- множество правил: переменная <b>rules</b>(типа структура <b>Rule</b>);</li> <li>- количество правил: переменная <b>size</b> (short);</li> <li>- стартовый символ грамматики: <b>startN</b> (<b>GRBALPHABET</b>);</li> <li>- служебный символ (дно стека и последняя лексема таблице лексем): <b>stbottomT</b> (<b>GRBALPHABET</b>);</li> <li>- два конструктора;</li> <li>- методы <b>getRule</b>: 1) позволяет получить номер правила или -1 (к точке возврата) и правило (второй параметр типа Rule) по левому символу правила (первый параметр); 2) позволяет получить правило (возвращает к точке вызова параметр типа Rule) по его номеру.</li> </ul>
<b>Rule</b>	<p>Структура: представление одного правила, имеющего вид: A-&gt;xxx yyy ....</p> <p>Структура включает:</p> <ul style="list-style-type: none"> <li>- нетерминал – левый символ правила: <b>nn</b> (<b>GRBALPHABET</b>);</li> <li>- идентификатор ошибки, связанной с правилом: <b>iderror</b> (int) – код ошибки в подсистеме <b>Error</b>;</li> <li>- количество цепочек в правой стороне правила: <b>size</b> (short);</li> <li>- цепочки-правые стороны правила: <b>chains</b> (типа <b>Rule::Chain</b>);</li> <li>- два конструктора;</li> <li>- метод <b>getCRule</b>: позволяет получить правило в виде строки вида N-&gt;цепочка (в символьном ASCII-виде, для отображения);</li> <li>- метод <b>getNextChain</b>: позволяет найти следующую за заданным номером (3й параметр <b>j</b> типа short) цепочку (параметр <b>pchain</b> типа <b>Rule::Chain</b>) и ее номер (к точке возврата типа short)</li> </ul>
<b>Rule::Chain</b>	<p>Структура: представление цепочки – правой стороны правила.</p> <p>Структура включает:</p> <ul style="list-style-type: none"> <li>- размер цепочки: <b>size</b> (short) в символах;</li> <li>- цепочка: <b>nt</b> (<b>GRBALPHABET</b>);</li> <li>- два конструктора;</li> <li>- метод <b>getCChain</b>: позволяет получить строку-цепочку в символьном виде для отображения;</li> <li>- методы <b>T</b> и <b>N</b>: преобразовывают ASCII-символы в <b>GRBALPHABET</b>-символы (терминалы и нетерминалы);</li> <li>- методы <b>isT</b> и <b>isN</b>: проверяют является <b>GRBALPHABET</b>-символ терминалом или нетерминалом;</li> <li>- метод <b>aplphabet_to_char</b>: преобразует заданный (параметр) <b>GRBALPHABET</b>-символ в ASCII-символ.</li> </ul>

8. Настройте таблицу сообщений (подсистема **Error**) так, чтобы диагностические сообщения, связанные с грамматикой языка SVV-2015 содержались в таблице сообщений подсистемы **Error** в диапазоне 600 - 699. На рис. 3 приведен пример описания (в подсистеме Error) таких диагностических сообщений.

```
ERROR errors[ERROR_MAX_ENTRY] = //таблица ошибок
{
    ERROR_ENTRY(0, "Недопустимый код ошибки"), // код ошибки вне диапазона 0 - ERROR_MAX_ENTRY
    ERROR_ENTRY(1, "Системный сбой"),
    ERROR_ENTRY_NODEF(2), ERROR_ENTRY_NODEF(3), ERROR_ENTRY_NODEF(4), ERROR_ENTRY_NODEF(5),
    ERROR_ENTRY_NODEF(6), ERROR_ENTRY_NODEF(7), ERROR_ENTRY_NODEF(8), ERROR_ENTRY_NODEF(9),
    ERROR_ENTRY_NODEF10(10), ERROR_ENTRY_NODEF10(20), ERROR_ENTRY_NODEF10(30), ERROR_ENTRY_NODEF10(40), ERROR_ENTRY_NODEF10(50),
    ERROR_ENTRY_NODEF10(60), ERROR_ENTRY_NODEF10(70), ERROR_ENTRY_NODEF10(80), ERROR_ENTRY_NODEF10(90),
    ERROR_ENTRY(100, "Параметр -in должен быть задан"),
    ERROR_ENTRY_NODEF(101), ERROR_ENTRY_NODEF(102), ERROR_ENTRY_NODEF(103),
    ERROR_ENTRY(104, "Превышена длина входного параметра"),
    ERROR_ENTRY_NODEF(105), ERROR_ENTRY_NODEF(106), ERROR_ENTRY_NODEF(107),
    ERROR_ENTRY_NODEF(108), ERROR_ENTRY_NODEF(109),
    ERROR_ENTRY(110, "Ошибка при открытии файла с исходным кодом (-in)"),
    ERROR_ENTRY(111, "Недопустимый символ в исходном файле (-in)"),
    ERROR_ENTRY(112, "Ошибка при создании файла протокола(-log)"),
    ERROR_ENTRY_NODEF(113), ERROR_ENTRY_NODEF(114), ERROR_ENTRY_NODEF(115),
    ERROR_ENTRY_NODEF(116), ERROR_ENTRY_NODEF(117), ERROR_ENTRY_NODEF(118), ERROR_ENTRY_NODEF(119),
    ERROR_ENTRY_NODEF10(120), ERROR_ENTRY_NODEF10(130), ERROR_ENTRY_NODEF10(140), ERROR_ENTRY_NODEF10(150),
    ERROR_ENTRY_NODEF10(160), ERROR_ENTRY_NODEF10(170), ERROR_ENTRY_NODEF10(180), ERROR_ENTRY_NODEF10(190),
    ERROR_ENTRY_NODEF100(200), ERROR_ENTRY_NODEF100(300), ERROR_ENTRY_NODEF100(400), ERROR_ENTRY_NODEF100(500),
    ERROR_ENTRY(600, "Неверная структура программы"),
    ERROR_ENTRY(601, "Ошибочный оператор"),
    ERROR_ENTRY(602, "Ошибка в выражении"),
    ERROR_ENTRY(603, "Ошибка в параметрах функции"),
    ERROR_ENTRY(604, "Ошибка в параметрах вызываемой функции"),
    ERROR_ENTRY_NODEF(605), ERROR_ENTRY_NODEF(606), ERROR_ENTRY_NODEF(607), ERROR_ENTRY_NODEF(608), ERROR_ENTRY_NODEF(609),
    ERROR_ENTRY_NODEF10(610), ERROR_ENTRY_NODEF10(620), ERROR_ENTRY_NODEF10(630), ERROR_ENTRY_NODEF10(640),
    ERROR_ENTRY_NODEF10(650), ERROR_ENTRY_NODEF10(660), ERROR_ENTRY_NODEF10(670), ERROR_ENTRY_NODEF10(680),
    ERROR_ENTRY_NODEF10(690),
    ERROR_ENTRY_NODEF100(700), ERROR_ENTRY_NODEF100(800), ERROR_ENTRY_NODEF100(900)
};
```

Рис.3. Фрагмент таблицы диагностических сообщений об ошибках, используемых синтаксическим анализатором

9. Разработайте структуры: **Rule**, **Rule::Chain** и **Greibach** для представления грамматики языка SVV-2015. Опишите грамматику с помощью структур примерно так, как это сделано на рис. 2.
10. Для моделирования конечного магазинного автомата создайте структуры по следующей спецификации (рис.4).

```

#define MFST_DIAGN_NUMBER 3
typedef std::stack<short> MFSTSTACK;           // стек автомата
namespace MFST
{
    struct MfstState                          // состояние автомата (для сохранения)
    {
        short lenta_position;                 // позиция на ленте
        short nrulechain;                     // номер текущей цепочки, текущего правила
        MFSTSTACK st;                         // стек автомата
        MfstState();
        MfstState(
            short pposition,                  // позиция на ленте
            MFSTSTACK pst,                    // стек автомата
            short pnrulechain                 // номер текущей цепочки, текущего правила
        );
    };
};

struct Mfst                                  // магазинный автомат
{
    enum RC_STEP {
        NS_OK,                               // код возврата функции step
        NS_NORULE,                            // не найдено правило грамматики (ошибка в грамматике)
        NS_NORULECHAIN,                       // не найдена подходящая цепочка правила (ошибка в исходном коде)
        NS_ERROR,                             // неизвестный нетерминальный символ грамматики
        TS_OK,                                // тек. символ ленты == вершине стека, продвинулась лента, pop стека
        TS_NOK,                               // тек. символ ленты != вершине стека, восстановлено состояние
        LENTA_END,                            // текущая позиция ленты >= lenta_size
        SURPRISE                              // неожиданный код возврата (ошибка в step)
    };

    struct MfstDiagnosis                      // диагностика
    {
        short lenta_position;                 // позиция на ленте
        RC_STEP rc_step;                      // код завершения шага
        short nrule;                          // номер правила
        short nrule_chain;                    // номер цепочки правила
        MfstDiagnosis();
        MfstDiagnosis(
            short plenta_position,             // позиция на ленте
            RC_STEP prc_step,                  // код завершения шага
            short pnrule,                      // номер правила
            short pnrule_chain                 // номер цепочки правила
        );
    } diagnosis[MFST_DIAGN_NUMBER];          // последние самые глубокие сообщения
};

```

Рис.4. Спецификация структуры **Mfst** для моделирования магазинного конченного автомата

```

GRBALPHABET* lenta;           // перекодированная (TS/NS) лента (из LEX)
short lenta_position;          // текущая позиция на ленте
short nrule;                   // номер текущего правила
short nrulechain;              // номер текущей цепочки, текущего правила
short lenta_size;              // размер ленты
GRB::Greibach grebach;         // грамматика Грейбах
LEX::LEX lex;                  // результат работы лексического анализатора
MFSTSTACK st;                  // стек автомата
std::stack<MfstState> storestate; // стек для сохранения состояний
Mfst();
Mfst(
    LEX::LEX plex,              // результат работы лексического анализатора
    GRB::Greibach pgrebach      // грамматика Грейбах
);
char* getCst(char* buf);        // получить содержимое стека
char* getCLenta(char* buf, short pos, short n = 25); // лента: n символов с pos
char* getDiagnosis(short n, char* buf); // получить n-ую строку диагностики или 0x00
bool savestate();               // сохранить состояние автомата
bool reststate();               // восстановить состояние автомата
bool push_chain(                // поместить цепочку правила в стек
    GRB::Rule::Chain chain      // цепочка правила
);
RC_STEP step();                 // выполнить шаг автомата
bool start();                   // запустить автомат
bool savediagnosis(
    RC_STEP pprc_step           // код завершения шага
);
};
};

```

Рис.4. Спецификация структуры **Mfst** для моделирования магазинного конечного автомата (продолжение)

11. Табл. 2 описывает назначение структур, приведенных на рис.4.

Таблица 2. Описание структур, для представления магазинного конечного автомата

Структура	Описание
<b>MfstState</b>	<p>Структура: для сохранения состояния автомата; сохранять состояние автомата необходимо для того, чтобы иметь возможность к этому состоянию вернуться и осуществить альтернативный вариант синтаксического разбора (в силу недетерминированности автомата).</p> <p>Структура включает:</p> <ul style="list-style-type: none"> <li>– текущую позицию на входной ленте автомата: <b>lenta_position</b> (short);</li> <li>– номер текущей цепочки, текущего правила: <b>nrulechain</b> (short);</li> <li>– стек автомата с содержимым на момент сохранения <b>st</b> (<b>MFSTSTACK</b>);</li> <li>– два конструктора.</li> </ul>
<b>Mfst</b>	<p>Структура: представление магазинного конечного автомата.</p> <p>Структура включает:</p> <ul style="list-style-type: none"> <li>– перечисление, содержащее возможные коды возврата метода <b>step</b>: <b>RC_STEP</b>;</li> <li>– массив структур для строк диагностики: <b>diagnosis</b> (<b>MstDiagnosis</b>, описание ниже);</li> </ul>

	<ul style="list-style-type: none"> <li>- входную ленту: <b>lenta (GRBALPHABET*)</b>;</li> <li>- текущая позиция на входной ленте: <b>lenta_position</b> (short);</li> <li>- номер текущего правила грамматики: <b>nrule</b> (short);</li> <li>- номер текущей цепочки текущего правила грамматики: <b>nrulechain</b> (short);</li> <li>- количество символов на ленте: <b>lenta_size</b> (short);</li> <li>- грамматика языка: <b>grebach (GRB::Greibach)</b>;</li> <li>- результат, предварительно выполненного лексического анализа (таблицы лексем и идентификаторов): <b>lex (LEX::LEX)</b>;</li> <li>- стек автомата: <b>st (MFSSTACK)</b>;</li> <li>- стек для хранения состояний (структур <b>MfstState</b>) автомата: <b>storestate</b> (std::stack&lt; <b>MfstState</b>&gt;);</li> <li>- два конструктора;</li> <li>- функция <b>getCSt</b>: принимает один параметр – буфер; заполняет буфер содержимым стека (в формате ASCII-строки) для отображения, в конце 0x00; возвращает к точке вызова указатель на буфер;</li> <li>- функция <b>getCLenta</b>: заполняет буфер (первый параметр) содержимым ленты с заданной позиции (второй параметр) заданное количество символов (третий параметр) в формате ASCII-строки для отображения, в конце строки 0x00; возвращает к точке вызова указатель на строки буфер;</li> <li>- функция <b>getDiagnosis</b>: по заданному номеру (первый параметр) строки диагностики записывает строку в буфер (второй параметр) в формате ASCII-строки для отображения и возвращает указатель на буфер;</li> <li>- функция <b>savestate</b>: сохраняет текущее состояние автомата в <b>storestate</b>, всегда возвращает <b>true</b>;</li> <li>- функция <b>reststate</b>: восстанавливает последнее сохраненное состояние автомата из <b>storestate</b>, возвращает <b>true</b>, если восстановление выполнено (есть данные для восстановления);</li> <li>- функция <b>push_chain</b>: помещает реверс цепочки (единственный параметр) в стек автомата, всегда возвращает <b>true</b>;</li> <li>- функция <b>step</b>: выполняет такт работы автомата, формирует диагностические сообщения, осуществляет отладочный вывод на консоль;</li> <li>- функция <b>start</b>: запускает работу автомата, в цикле выполняет функцию <b>step</b>, осуществляет вывод диагностических сообщений;</li> <li>- функция <b>savediagnosis</b>: сохраняет в массиве <b>diagnosis</b> строку диагностики; в массиве <b>diagnosis</b> сохраняются диагностические сообщения в порядке убывания позиции ленты (вызвавшей диагностику) и только в равным длине массива (макрос <b>MFST_DIAGN_NUMBER</b>).</li> </ul>
<b>MfstDiagnosis</b>	<p>Структура (внутренняя для <b>Mfst</b>): представление строки диагностики.</p> <p>Структура включает:</p> <ul style="list-style-type: none"> <li>- позиция входной ленты: <b>lenta_position</b> (short);</li> <li>- код возврата, сформированный функцией <b>step</b>;</li> </ul>

	<ul style="list-style-type: none"> <li>- номер действующего на момент диагностики правила грамматики: <b>nrule</b> (short);</li> <li>- номер текущей цепочки действующего на момент диагностики правила грамматики: <b>nrule_chain</b>(short).</li> <li>- два конструктора.</li> </ul>
--	--

12. Вызов лексического анализатора выполните в следующем виде (рис.5).

```
#include "stdafx.h"
#include <iostream>
#include <locale>
#include "MFST.h" // магазинный автомат
int _tmain(int argc, _TCHAR* argv[])
{
    setlocale(LC_ALL, "rus");

    int s = 0;
    LEX::LEX lex;           // лексического анализа
    lex.lextable.table[ s] = LT::Entry('t',1);    // LT::Entry( лексема , номер исходной строки )
    lex.lextable.table[++s] = LT::Entry('i',1);
    lex.lextable.table[++s] = LT::Entry('f',1);
    // и т.д. заполнение таблицы лексем
    // .....
    lex.lextable.table[++s] = LT::Entry(';',11);
    lex.lextable.table[++s] = LT::Entry('$',12);
    lex.lextable.size = ++s;
    MFST_TRACE_START           // отладка
    MFST::Mfst mfst(lex, GRB::getGreibach()); // автомат
    mfst.start();              // старт синтаксического анализа

    system("pause");
    return 0;
}
```

Рис.5. Подготовка таблицы лексем и вызов синтаксического анализатора

13. В результате работы синтаксического анализа на консоль должна выводиться трассировка каждого шага содержащая: номер шага, действующее правило (левые и правые части), состояние ленты и стека. Кроме того, в трассировке должны быть отражены операции сохранения и восстановления состояния автомата. В случае обнаружения ошибки должны быть отражены соответствующие диагностические сообщения (см. рис. 6 и 7).



Лин	Правило	Входная лента	Стек
0	S->tif<F><NrE;>;S	tif<ti,ti><dti;i=io<i>i>i>;	S\$
0	SAVESTATE:	1	
1		tif<ti,ti><dti;i=io<i>i>i>;	tif<F><NrE;>;S\$
2		if<ti,ti><dti;i=io<i>i>i>;r	if<F><NrE;>;S\$
3		f<ti,ti><dti;i=io<i>i>i>;ri	f<F><NrE;>;S\$
4		<ti,ti><dti;i=io<i>i>i>;ri;	<F><NrE;>;S\$
5	F->ti	ti,ti><dti;i=io<i>i>i>;ri;>	F><NrE;>;S\$
5	SAVESTATE:	2	F><NrE;>;S\$
6		ti,ti><dti;i=io<i>i>i>;ri;>	ti><NrE;>;S\$
7		i,ti><dti;i=io<i>i>i>;ri;>;	i><NrE;>;S\$
7		ti><dti;i=io<i>i>i>;ri;>;m	><NrE;>;S\$
8	TS_NOK/NS_NORULECHAIN		
8	RESSTATE	ti,ti><dti;i=io<i>i>i>;ri;>	F><NrE;>;S\$
9	F->ti,F	ti,ti><dti;i=io<i>i>i>;ri;>	F><NrE;>;S\$
9	SAVESTATE:	2	
10		ti,ti><dti;i=io<i>i>i>;ri;>	ti,F><NrE;>;S\$
11		i,ti><dti;i=io<i>i>i>;ri;>;	i,F><NrE;>;S\$
12		ti><dti;i=io<i>i>i>;ri;>;m	F><NrE;>;S\$
13	F->ti	ti><dti;i=io<i>i>i>;ri;>;m<	F><NrE;>;S\$
13	SAVESTATE:	3	F><NrE;>;S\$
14		ti><dti;i=io<i>i>i>;ri;>;m<	ti><NrE;>;S\$
15		i><dti;i=io<i>i>i>;ri;>;m<d	i><NrE;>;S\$
16		><dti;i=io<i>i>i>;ri;>;m<dt	><NrE;>;S\$
17		<dti;i=io<i>i>i>;ri;>;m<dti	<NrE;>;S\$
18	N->dti;	dti;i=io<i>i>i>;ri;>;m<dti;	NrE;>;S\$
18	SAVESTATE:	4	NrE;>;S\$
19		dti;i=io<i>i>i>;ri;>;m<dti;	dti;rE;>;S\$
20		ti;i=io<i>i>i>;ri;>;m<dti;r	ti;rE;>;S\$
21		i;i=io<i>i>i>;ri;>;m<dti;ri	i;rE;>;S\$
22		i=io<i>i>i>;ri;>;m<dti;ri;	rE;>;S\$
23	TS_NOK/NS_NORULECHAIN		
23	RESSTATE	dti;i=io<i>i>i>;ri;>;m<dti;	NrE;>;S\$
24	N->dtfi<F>;	dti;i=io<i>i>i>;ri;>;m<dti;	NrE;>;S\$
24	SAVESTATE:	4	
70		dti;ri;>;\$	NrE;>;\$
71	N->dti;	dti;ri;>;\$	NrE;>;\$
71	SAVESTATE:	14	
71		dti;ri;>;\$	dti;rE;>;\$
72		ti;ri;>;\$	ti;rE;>;\$
73		i;ri;>;\$	i;rE;>;\$
74		;ri;>;\$	;rE;>;\$
75		ri;>;\$	rE;>;\$
76		i;>;\$	E;>;\$
77	E->i	i;>;\$	E;>;\$
77	SAVESTATE:	15	
77		i;>;\$	i;>;\$
78		;;>;\$	;;>;\$
79		>;\$	>;\$
80		;\$	;\$
81		\$	\$
82			
83	LENTA_END		
84	----->LENTA_END		
0 : всего строк 42, синтаксический анализ выполнен без ошибок			
Для продолжения нажмите любую клавишу . . .			

Рис.6. Пример отладочного вывода трассировки синтаксического разбора и диагностики

Шаг	Правило	Входная лента	Стек
0	S->tif(F)<NrE;>;S	tif(ti,ti)<dtii=iv(iv);r	S\$
0	SAVESTATE:	1	
0		tif(ti,ti)<dtii=iv(iv);r	tif(F)<NrE;>;S\$
1		if(ti,ti)<dtii=iv(iv);ri	if(F)<NrE;>;S\$
2		f(ti,ti)<dtii=iv(iv);ri;	f(F)<NrE;>;S\$
3		<ti,ti)<dtii=iv(iv);ri;>	<F)<NrE;>;S\$
4		ti,ti)<dtii=iv(iv);ri;>;	F)<NrE;>;S\$
5	F->ti	ti,ti)<dtii=iv(iv);ri;>;	F)<NrE;>;S\$
5	SAVESTATE:	2	
5		ti,ti)<dtii=iv(iv);ri;>;	ti)<NrE;>;S\$
6		i,ti)<dtii=iv(iv);ri;>;m	i)<NrE;>;S\$
7		,ti)<dtii=iv(iv);ri;>;m<	><NrE;>;S\$
8	TS_NOK/NS_NORULECHAIN		
8	RESSTATE		
8		ti,ti)<dtii=iv(iv);ri;>;	F)<NrE;>;S\$
9	F->ti,F	ti,ti)<dtii=iv(iv);ri;>;	F)<NrE;>;S\$
9	SAVESTATE:	2	
9		ti,ti)<dtii=iv(iv);ri;>;	ti,F)<NrE;>;S\$
10		i,ti)<dtii=iv(iv);ri;>;m	i,F)<NrE;>;S\$
11		,ti)<dtii=iv(iv);ri;>;m<	,F)<NrE;>;S\$
12		ti)<dtii=iv(iv);ri;>;m<d	F)<NrE;>;S\$
13	F->ti	ti)<dtii=iv(iv);ri;>;m<d	F)<NrE;>;S\$
13	SAVESTATE:	3	
13		ti)<dtii=iv(iv);ri;>;m<d	ti)<NrE;>;S\$
14		i)<dtii=iv(iv);ri;>;m<dt	i)<NrE;>;S\$
15		<dtii=iv(iv);ri;>;m<dti	><NrE;>;S\$
16		<dtii=iv(iv);ri;>;m<dti;	<NrE;>;S\$
17		dtii=iv(iv);ri;>;m<dti;r	NrE;>;S\$
18	N->dti;	dtii=iv(iv);ri;>;m<dti;r	NrE;>;S\$
18	SAVESTATE:	4	
18		dtii=iv(iv);ri;>;m<dti;r	dti;rE;>;S\$
19		tii=iv(iv);ri;>;m<dti;ri	ti;rE;>;S\$
20		ii=iv(iv);ri;>;m<dti;ri;	i;rE;>;S\$
21		i=iv(iv);ri;>;m<dti;ri;>	;rE;>;S\$
22	TS_NOK/NS_NORULECHAIN		
22	RESSTATE		
22		dtii=iv(iv);ri;>;m<dti;r	NrE;>;S\$
23	N->dtfi(F);	dtii=iv(iv);ri;>;m<dti;r	NrE;>;S\$
23	SAVESTATE:	4	
23		dtii=iv(iv);ri;>;m<dti;r	dtfi(F);rE;>;S\$
24		tii=iv(iv);ri;>;m<dti;ri	tfi(F);rE;>;S\$
25		ii=iv(iv);ri;>;m<dti;ri;	fi(F);rE;>;S\$
40	TS_NOK/NS_NORULECHAIN		
40	RESSTATE		
40		ti)<dtii=iv(iv);ri;>;m<d	F)<NrE;>;S\$
41	TNS_NORULECHAIN/NS_NORULE		
41	RESSTATE		
41		ti,ti)<dtii=iv(iv);ri;>;	F)<NrE;>;S\$
42	TNS_NORULECHAIN/NS_NORULE		
42	RESSTATE		
42		tif(ti,ti)<dtii=iv(iv);r	S\$
43	TNS_NORULECHAIN/NS_NORULE		
44	----->NS_NORULE		
-----			
501:	строка 3, Ошибочный оператор		
503:	строка 1, Ошибка в параметрах функции		
503:	строка 1, Ошибка в параметрах функции		
Для продолжения нажмите любую клавишу . . . _			

Рис.7. Пример отладочного вывода трассировки синтаксического разбора и диагностики