

Лабораторная работа 18 (6 часов)

Конструирование программного обеспечения

Разработка синтаксического анализатора

1. Используйте материалы лекций № 16 -18.
2. Создайте проект (VS20xx, C++, консольное приложение) с именем **SELab18**.
3. Разработайте синтаксический анализатор для языка SVV-2015 (описан в лекциях и задании к лабораторной работе 18).
4. В контрольном примере в качестве входных данных используйте таблицу лексем, полученную в лабораторной работе 17.
5. 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* getCCchain(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
        }* 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 getGreibach(); // 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}; | tfi(F){NrE};S | m{NrE};S | tfi(F){NrE};
            Rule::Chain(8, TS('m'), TS('{'), NS('N'), TS('r'), NS('E'),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('}')),
            Rule::Chain(9, TS('m'), TS('{'), NS('N'), TS('r'), NS('E'),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(';')),
            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(';'), NS('N'))
        ),
        Rule(NS('E'), GRB_ERROR_SERIES + 2, // Ошибка в выражении
            8, // E->i|l|(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**.

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

Таблица 1. Описание структур, для представления грамматики

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

7. 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

8. 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.
9. 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 и цепочка, цепочка записана в стек
        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, рор 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)

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

Таблица 2. Описание структур, для представления магазинного конечного автомата

Структура	Описание
MfstState	<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"> – текущую позицию на входной ленте автомата: lenta_position (short); – номер текущей цепочки, текущего правила: nrulechain (short); – стек автомата с содержимым на момент сохранения st (MFSTSTACK); – два конструктора.
Mfst	<p>Структура: представление магазинного конечного автомата.</p> <p>Структура включает:</p> <ul style="list-style-type: none"> – перечисление, содержащее возможные коды возврата метода step: RC_STEP; – массив структур для строк диагностики: diagnosis (MstDiagnosis, описание ниже);

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

11. 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

12. 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).

Шаг	Правило	Входная лента	Стек
0	S->tif<F><NrE;>;S	tif<ti,ti><dti;i=iv<iv>;	S\$
0	SAVESTATE:	1	
1		tif<ti,ti><dti;i=iv<iv>;	tif<F><NrE;>;S\$
1		if<ti,ti><dti;i=iv<iv>;r	if<F><NrE;>;S\$
2		f<ti,ti><dti;i=iv<iv>;ri	f<F><NrE;>;S\$
3		<ti,ti><dti;i=iv<iv>;ri;	<F><NrE;>;S\$
4		ti,ti><dti;i=iv<iv>;ri;	F><NrE;>;S\$
5	F->ti	ti,ti><dti;i=iv<iv>;ri;	F><NrE;>;S\$
5	SAVESTATE:	2	
5		ti,ti><dti;i=iv<iv>;ri;	ti><NrE;>;S\$
6		i,ti><dti;i=iv<iv>;ri;	i><NrE;>;S\$
7		,ti><dti;i=iv<iv>;ri;;m	><NrE;>;S\$
8	TS_NOK/NS_NORULECHAIN		
8	RESSTATE	ti,ti><dti;i=iv<iv>;ri;	F><NrE;>;S\$
9	F->ti,F	ti,ti><dti;i=iv<iv>;ri;	F><NrE;>;S\$
9	SAVESTATE:	2	
9		ti,ti><dti;i=iv<iv>;ri;	ti,F><NrE;>;S\$
10		i,ti><dti;i=iv<iv>;ri;;	i,F><NrE;>;S\$
11		,ti><dti;i=iv<iv>;ri;;m	,F><NrE;>;S\$
12		ti><dti;i=iv<iv>;ri;;m<	F><NrE;>;S\$
13	F->ti	ti><dti;i=iv<iv>;ri;;m<	F><NrE;>;S\$
13	SAVESTATE:	3	
13		ti><dti;i=iv<iv>;ri;;m<	ti><NrE;>;S\$
14		i><dti;i=iv<iv>;ri;;m<d	i><NrE;>;S\$
15		><dti;i=iv<iv>;ri;;m<dt	><NrE;>;S\$
16		<dti;i=iv<iv>;ri;;m<dti	<NrE;>;S\$
17		dti;i=iv<iv>;ri;;m<dti;	NrE;>;S\$
18	N->dti;	dti;i=iv<iv>;ri;;m<dti;	NrE;>;S\$
18	SAVESTATE:	4	
18		dti;i=iv<iv>;ri;;m<dti;	dti;rE;>;S\$
19		ti;i=iv<iv>;ri;;m<dti;r	ti;rE;>;S\$
20		i;i=iv<iv>;ri;;m<dti;ri	i;rE;>;S\$
21		;i=iv<iv>;ri;;m<dti;ri;	;rE;>;S\$
22		i=iv<iv>;ri;;m<dti;ri;>	rE;>;S\$
23	TS_NOK/NS_NORULECHAIN		
23	RESSTATE	dti;i=iv<iv>;ri;;m<dti;	NrE;>;S\$
24	N->dtfi<F>;	dti;i=iv<iv>;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