Министерство образования и науки Российской Федерации

Новосибирский государственный технический университет

Кафедра прикладной математики

Языки программирования и методы трансляции

Лабораторная работа №3

Факультет прикладной математики и информатики

Группа ПМ-01

Студенты Александров М.Е.

Жигалов П.С.

Преподаватели Еланцева И.Л.

Полетаева И.А.

Вариант 7

Новосибирск

2013

1. Цель работы

Изучить табличные методы синтаксического анализа. Получить представление о методах диагностики и исправления синтаксических ошибок. Научиться проектировать синтаксический анализатор на основе табличных методов.

2. Задание

Подмножество языка С++ включает:

* данные типа **int, float, массивы** из элементов указанных типов;
* инструкции описания переменных;
* операторы присваивания в любой последовательности;
* операции **+, – , \*, = =, != , <, >** .

В соответствии с выбранным вариантом заданий к лабораторным работам реализовать синтаксический анализатор с использованием одного из табличных методов (LL-, LR-метод, метод предшествования).

Этапы проектирования синтаксического анализатора:

1. Сконструировать КС-грамматику в соответствии с вариантом задания.

2. В случае несоответствия построенной грамматики требованиям выбранного табличного метода разбора следует провести эквивалентные преобразования грамматики либо выбрать другой метод разбора.

3. Построить таблицу разбора и запрограммировать драйвер, реализующий работу с этой таблицей.

Исходные данные – файл токенов, таблицы лексем.

Результатом работы синтаксического анализатора является:

• синтаксическое дерево или постфиксная запись;

• файл сообщений об ошибках. В лабораторной работе необходимо реализовать возможности табличного метода по диагностике и исправлению синтаксических ошибок в исходной программе.

3. Структура входных и выходных данных

Входные данные представляют собой имена файлов: файла токенов, файла ошибок и файла для вывода постфиксной записи, а также полученные в результате работы №2 таблицы. Результатом работы программы являются два файла – файл с постфиксной записью и файл ошибок.

4. Грамматика языка

# Типы

TYPE -> int

TYPE -> float

TYPE\_ADV -> TYPE

TYPE\_ADV -> void

# Операции

OPER -> +

OPER -> -

OPER -> \*

OPER -> ==

OPER -> !=

OPER -> <

OPER -> >

# Операции присваивания

OPER\_ASS -> =

OPER\_ASS -> +=

OPER\_ASS -> -=

OPER\_ASS -> \*=

# Начальный символ

S -> PROG

# Программа

PROG -> TYPE\_ADV main ( ) { BODY }

PROG -> eps

# Тело программы

BODY -> DECLAR BODY # Объявление переменной

BODY -> ASSIGN BODY # Присваивание

BODY -> eps

# Для работы с переменными, константами и массивами

VAR -> var INDEX # Имя переменной с индексом

VAR\_ADV -> VAR # Или переменная

VAR\_ADV -> const # Или константа

INDEX -> eps # Индекса либо нет (переменная)

INDEX -> [ EXPR ] # Либо есть (массив)

# Присваивание

ASSIGN -> VAR OPER\_ASS EXPR ; # Переменная, операция, выражение, ";"

EXPR -> - E\_1 # Может быть с унарным минусом

EXPR -> E\_1 # Или же без него

E\_1 -> VAR\_ADV E\_2 # Переменная / константа и дальнейшие операции

E\_1 -> ( EXPR ) E\_2 # Может быть выражение в скобках

E\_2 -> eps # Может ничего не быть

E\_2 -> OPER E\_1 # А может быть знак операции и продолжение выражения

# Объявление переменной

DECLAR -> TYPE D\_1 ; # Начинается с типа переменной / массива

D\_1 -> VAR A\_1 D\_2 # Потом обязательно имя / массива

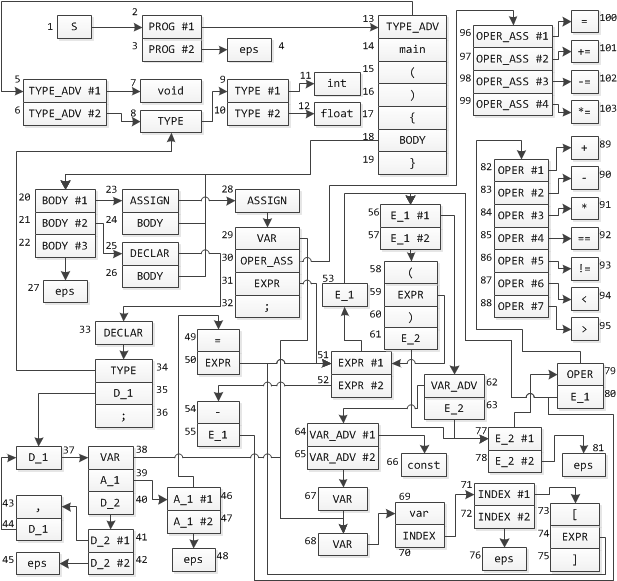
A\_1 -> = EXPR # Может быть сразу с присваиванием

A\_1 -> eps # А может и без

D\_2 -> , D\_1 # Могут быть еще объявления

D\_2 -> eps # А могут и не быть

5. Схема разбора



6. Таблица разбора

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ***N*** | ***Terminals*** | ***Jump*** | ***Accept*** | ***Stack*** | ***Return*** | ***Error*** |
| 1 | void int float | 2 | 0 | 0 | 0 | 1 |
| 2 | void int float | 13 | 0 | 0 | 0 | 0 |
| 3 | eps | 4 | 0 | 0 | 0 | 1 |
| 4 | eps | -1 | 0 | 0 | 1 | 1 |
| 5 | void | 7 | 0 | 0 | 0 | 0 |
| 6 | int float | 8 | 0 | 0 | 0 | 1 |
| 7 | void | -1 | 1 | 0 | 1 | 1 |
| 8 | int float | 9 | 0 | 0 | 0 | 1 |
| 9 | int | 11 | 0 | 0 | 0 | 0 |
| 10 | float | 12 | 0 | 0 | 0 | 1 |
| 11 | int | -1 | 1 | 0 | 1 | 1 |
| 12 | float | -1 | 1 | 0 | 1 | 1 |
| 13 | void int float | 5 | 0 | 1 | 0 | 1 |
| 14 | main | 15 | 1 | 0 | 0 | 1 |
| 15 | ( | 16 | 1 | 0 | 0 | 1 |
| 16 | ) | 17 | 1 | 0 | 0 | 1 |
| 17 | { | 18 | 1 | 0 | 0 | 1 |
| 18 | var int float } | 20 | 0 | 1 | 0 | 1 |
| 19 | } | -1 | 1 | 0 | 1 | 1 |
| 20 | var | 23 | 0 | 0 | 0 | 0 |
| 21 | int float | 25 | 0 | 0 | 0 | 0 |
| 22 | } | 27 | 0 | 0 | 0 | 1 |
| 23 | var | 28 | 0 | 1 | 0 | 1 |
| 24 | var int float } | 20 | 0 | 0 | 0 | 1 |
| 25 | int float | 33 | 0 | 1 | 0 | 1 |
| 26 | var int float } | 20 | 0 | 0 | 0 | 1 |
| 27 | } | -1 | 0 | 0 | 1 | 1 |
| 28 | var | 29 | 0 | 0 | 0 | 1 |
| 29 | var | 68 | 0 | 1 | 0 | 1 |
| 30 | = += -= \*= | 96 | 0 | 1 | 0 | 1 |
| 31 | - ( const var | 51 | 0 | 1 | 0 | 1 |
| 32 | ; | -1 | 1 | 0 | 1 | 1 |
| 33 | int float | 34 | 0 | 0 | 0 | 1 |
| 34 | int float | 8 | 0 | 1 | 0 | 1 |
| 35 | var | 37 | 0 | 1 | 0 | 1 |
| 36 | ; | -1 | 1 | 0 | 1 | 1 |
| 37 | var | 38 | 0 | 0 | 0 | 1 |
| 38 | var | 68 | 0 | 1 | 0 | 1 |
| 39 | = , ; | 46 | 0 | 1 | 0 | 1 |
| 40 | , ; | 41 | 0 | 0 | 0 | 1 |
| 41 | , | 43 | 0 | 0 | 0 | 0 |
| 42 | ; | 45 | 0 | 0 | 0 | 1 |
| 43 | , | 44 | 1 | 0 | 0 | 1 |
| 44 | var | 37 | 0 | 0 | 0 | 1 |
| 45 | ; | -1 | 0 | 0 | 1 | 1 |
| 46 | = | 49 | 0 | 0 | 0 | 0 |
| 47 | , ; | 48 | 0 | 0 | 0 | 1 |
| 48 | , ; | -1 | 0 | 0 | 1 | 1 |
| 49 | = | 50 | 1 | 0 | 0 | 1 |
| 50 | - ( const var | 51 | 0 | 0 | 0 | 1 |
| 51 | ( const var | 53 | 0 | 0 | 0 | 0 |
| 52 | - | 54 | 0 | 0 | 0 | 1 |
| 53 | ( const var | 56 | 0 | 0 | 0 | 1 |
| 54 | - | 55 | 1 | 0 | 0 | 1 |
| 55 | ( const var | 56 | 0 | 0 | 0 | 1 |
| 56 | const var | 62 | 0 | 0 | 0 | 0 |
| 57 | ( | 58 | 0 | 0 | 0 | 1 |
| 58 | ( | 59 | 1 | 0 | 0 | 1 |
| 59 | - ( const var | 51 | 0 | 1 | 0 | 1 |
| 60 | ) | 61 | 1 | 0 | 0 | 1 |
| 61 | + - \* == != < > ; , ] ) | 77 | 0 | 0 | 0 | 1 |
| 62 | const var | 64 | 0 | 1 | 0 | 1 |
| 63 | + - \* == != < > ; , ] ) | 77 | 0 | 0 | 0 | 1 |
| 64 | const | 66 | 0 | 0 | 0 | 0 |
| 65 | var | 67 | 0 | 0 | 0 | 1 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ***N*** | ***Terminals*** | ***Jump*** | ***Accept*** | ***Stack*** | ***Return*** | ***Error*** |
| 66 | const | -1 | 1 | 0 | 1 | 1 |
| 67 | var | 68 | 0 | 0 | 0 | 1 |
| 68 | var | 69 | 0 | 0 | 0 | 1 |
| 69 | var | 70 | 1 | 0 | 0 | 1 |
| 70 | [ + - \* == != < > l , ; ) ] = | 71 | 0 | 0 | 0 | 1 |
| 71 | [ | 73 | 0 | 0 | 0 | 0 |
| 72 | + - \* == != < > ; , ] ) = | 76 | 0 | 0 | 0 | 1 |
| 73 | [ | 74 | 1 | 0 | 0 | 1 |
| 74 | - ( const var | 51 | 0 | 1 | 0 | 1 |
| 75 | ] | -1 | 1 | 0 | 1 | 1 |
| 76 | + - \* == != < > ; , ] ) = | -1 | 0 | 0 | 1 | 1 |
| 77 | + - \* == != < > | 79 | 0 | 0 | 0 | 0 |
| 78 | ; , ] ) | 81 | 0 | 0 | 0 | 1 |
| 79 | + - \* == != < > | 82 | 0 | 1 | 0 | 1 |
| 80 | ( const var | 56 | 0 | 0 | 0 | 1 |
| 81 | ; , ] ) | -1 | 0 | 0 | 1 | 1 |
| 82 | + | 89 | 0 | 0 | 0 | 0 |
| 83 | - | 90 | 0 | 0 | 0 | 0 |
| 84 | \* | 91 | 0 | 0 | 0 | 0 |
| 85 | == | 92 | 0 | 0 | 0 | 0 |
| 86 | != | 93 | 0 | 0 | 0 | 0 |
| 87 | < | 94 | 0 | 0 | 0 | 0 |
| 88 | > | 95 | 0 | 0 | 0 | 1 |
| 89 | + | -1 | 1 | 0 | 1 | 1 |
| 90 | - | -1 | 1 | 0 | 1 | 1 |
| 91 | \* | -1 | 1 | 0 | 1 | 1 |
| 92 | == | -1 | 1 | 0 | 1 | 1 |
| 93 | != | -1 | 1 | 0 | 1 | 1 |
| 94 | < | -1 | 1 | 0 | 1 | 1 |
| 95 | > | -1 | 1 | 0 | 1 | 1 |
| 96 | = | 100 | 0 | 0 | 0 | 0 |
| 97 | += | 101 | 0 | 0 | 0 | 0 |
| 98 | -= | 102 | 0 | 0 | 0 | 0 |
| 99 | \*= | 103 | 0 | 0 | 0 | 1 |
| 100 | = | -1 | 1 | 0 | 1 | 1 |
| 101 | += | -1 | 1 | 0 | 1 | 1 |
| 102 | -= | -1 | 1 | 0 | 1 | 1 |
| 103 | \*= | -1 | 1 | 0 | 1 | 1 |

7. Тесты

7.1. Корректный код

Код:

void main()

{

int a = 0, b, c[2+3], d;

c[0] = 1;

a = 1 + 2 + 7 != 3 \* (4 + 6) + 1;

b = 1 + 2;

b[1] = 2;

}

Таблицы:

ID`s:

0: [ d int dim=1 init={0} ]

97: [ a int dim=1 init={0} ]

98: [ b int dim=1 init={0} ]

99: [ c int dim=1 init={0} ]

CONST`s:

48: [ 0 notype dim=1 init={0} ]

49: [ 1 notype dim=1 init={0} ]

50: [ 2 notype dim=1 init={0} ]

51: [ 3 notype dim=1 init={0} ]

52: [ 4 notype dim=1 init={0} ]

54: [ 6 notype dim=1 init={0} ]

55: [ 7 notype dim=1 init={0} ]

Постфиксная запись

a 0 = ; c 2 3 + [\*] ; c 0 [] 1 = ; a 1 2 + 7 + 3 4 6 + \* 1 + != = ; b 1 2 + = ; b 1 [] 2 = ;

7.2. Необъявленный идентификатор

Код:

void main()

{

f = 0;

int f;

}

Файл ошибок:

Syntax Error: Undefined identifier "f"

7.3. Неразрешенный терминал

Код:

void main()

{

int ; a = 0;

}

Файл ошибок:

Syntax Error: Unexpected terminal ";"

Must be: "var"

7.4. Попытка присвоить значение константе

Код:

void main()

{

0 = 0;

}

Файл ошибок:

Syntax Error: Unexpected terminal "0"

Must be: "var" "int" "float" "}"

7.5. Попытка инициализировать массив во время объявления

Код:

void main()

{

float c[7] = 10;

}

Файл ошибок:

Syntax Error: Can`t assign to array "c"

7.6. Неверный баланс скобок

Код:

void main()

{

int a, b, c, d;

a = b + (c - d));

}

Файл ошибок:

Syntax Error: Unexpected terminal ")"

Must be: ";"

8. Код программы

translator.h

#ifndef TRANSLATOR\_H\_INCLUDED

#define TRANSLATOR\_H\_INCLUDED

#include <iostream>

#include <sstream>

#include <fstream>

#include <string>

#include <stack>

#include <vector>

#include "table\_const.h"

#include "table\_var.h"

#include "lexeme.h"

#include "token.h"

using namespace std;

class translator

{

private:

// Постоянные таблицы

table\_const<char> letters; // 0

table\_const<char> numbers; // 1

table\_const<string> operations; // 2

table\_const<string> keywords; // 3

table\_const<char> separators; // 4

// Переменные таблицы

table\_var identifiers; // 5

table\_var constants; // 6

// Файловые потоки

ifstream in\_source;

ofstream out\_token;

ofstream out\_error;

// Анализ строки

bool analyze\_lexical\_string(string str);

// Удаление комментариев

bool analyze\_lexical\_decomment(string& str, bool is\_changed);

// Счетчики для подробных сообщений об ошибке

int analyze\_lexical\_strnum, analyze\_lexical\_strinc;

// Удаление пробелов

static inline void ltrim(string& out\_)

{

int notwhite = out\_.find\_first\_not\_of(" \t\n");

out\_.erase(0, notwhite);

}

static inline void rtrim(string& out\_)

{

int notwhite = out\_.find\_last\_not\_of(" \t\n");

out\_.erase(notwhite + 1);

}

static inline void trim(string& out\_)

{

ltrim(out\_);

rtrim(out\_);

}

/\*\* Синтаксический анализ \*/

// Определяем какая строка содержится в токене

string get\_token\_text(token get\_t);

// Структура элемент таблицы разбора

struct table\_parse\_elem

{

vector<string> terminal; // Терминалы

int jump; // Переход

bool accept; // Принимать или нет

bool stack\_; // Класть в стек или нет

bool return\_; // Возвращать или нет

bool error; // Может ли быть ошибка

};

// Таблица разбора

vector<table\_parse\_elem> table\_parse;

// Структура элемент постфиксной записи

struct postfix\_elem

{

string id;

short int type;

postfix\_elem()

{

id = "", type = 0;

}

postfix\_elem(string id\_, int type\_)

{

id = id\_, type = type\_;

}

postfix\_elem(string id\_)

{

id = id\_, type = 1;

}

friend ostream& operator << (ostream& ostream\_, const postfix\_elem& pe\_)

{

ostream\_ << pe\_.id;

return ostream\_;

}

};

// Сравнение приоритетов операций

bool priority\_le(string what, string with\_what);

// Постфиксная запись

vector<postfix\_elem> postfix\_record;

// Построение постфиксной записи

bool make\_postfix(vector<token> t);

public:

// Конструктор со вводом постоянных таблиц

translator();

// Отладочный вывод таблиц

void debug\_print(ostream& stream);

// Лексический анализ

bool analyze\_lexical(string file\_source, string file\_tokens, string file\_error);

// Синтаксический анализ

bool analyze\_syntactical(string file\_tokens, string file\_error);

// Печать постфиксной записи в файл и на экран

void postfix\_print(string file\_tree);

};

#endif // TRANSLATOR\_H\_INCLUDED

translator.cpp (то, что было изменено или добавлено)

#include "translator.h"

/\*\* ================ Общие функции собственно транслятора ================ \*/

// Конструктор со вводом постоянных таблиц и таблицы разбора

translator::translator()

{

letters.read\_file("files/table\_letters.txt");

numbers.read\_file("files/table\_numbers.txt");

operations.read\_file("files/table\_operations.txt");

keywords.read\_file("files/table\_keywords.txt");

separators.read\_file("files/table\_separators.txt");

ifstream in\_table\_parse;

in\_table\_parse.open("files/table\_parse.txt", ios::in);

string str;

getline(in\_table\_parse, str, '\n');

struct table\_parse\_elem te;

te.jump = 1;

te.accept = false;

te.stack\_ = true;

te.return\_ = false;

te.error = true;

table\_parse.push\_back(te);

while(!in\_table\_parse.eof())

{

struct table\_parse\_elem te;

string str;

in\_table\_parse >> str;

if(in\_table\_parse.eof())

break;

stringstream a;

str = "";

while(str.length() == 0 || str.find("\t") != string::npos)

getline(in\_table\_parse, str, '\t');

a.str(str);

while(a.good())

{

a >> str;

te.terminal.push\_back(str);

}

in\_table\_parse >> te.jump >> te.accept >> te.stack\_ >> te.return\_ >> te.error;

table\_parse.push\_back(te);

}

table\_parse[0].terminal.resize(table\_parse[1].terminal.size());

for(int i = 0; i < (int)table\_parse[1].terminal.size(); i++)

table\_parse[0].terminal[i] = table\_parse[1].terminal[i];

in\_table\_parse.close();

}

/\*\* ================ Функции синтаксического анализатора ================ \*/

// Получение строки, на которую указывает токен

string translator::get\_token\_text(token t)

{

string str = "";

char sym = '\0';

lexeme l("");

switch(t.table)

{

case 2:

operations.get\_val(t.place, str);

return str;

case 3:

keywords.get\_val(t.place, str);

return str;

case 4:

separators.get\_val(t.place, sym);

str.append(&sym, 1);

return str;

case 5:

identifiers.get\_lexeme(t.place, t.chain, l);

return l.name;

case 6:

constants.get\_lexeme(t.place, t.chain, l);

return l.name;

}

return str;

}

// Синтаксический анализатор

bool translator::analyze\_syntactical(string tokens\_file, string errors\_file)

{

ifstream in\_token(tokens\_file.c\_str(), ios::in);

out\_error.open(errors\_file.c\_str(), ios::out);

token curr\_token, next\_token;

stack<int> parse\_stack;

bool error\_flag = false;

int curr\_row = 0;

bool have\_type = false; // Находимся ли мы в строке с объявлением типа

int type\_type; // Если находимся, то какой тип объявляем

bool need\_postfix = false; // Нужно ли выполнять построение постфиксной записи для данной строки

vector<token> code\_expr\_infix; // Если да, то сюда помещаем токены в инфиксном (обычном) порядке

bool need\_array\_resize = false; // Объявляем ли мы сейчас размер массива

vector<token> array\_resize\_expr\_infix; // Если да, то сюда помещаем токены в инфиксном (обычном) порядке

bool eof\_flag = in\_token.eof(); // Флаг конца файла (чтобы считать последний токен)

in\_token >> curr\_token >> next\_token;

while(!eof\_flag && !error\_flag)

{

string token\_str = get\_token\_text(curr\_token);

trim(token\_str);

if(curr\_token.table == 5) token\_str = "var";

if(curr\_token.table == 6) token\_str = "const";

// Ищем терминалы из списка

bool find\_terminal = false;

cout << "Curr Row = " << curr\_row << endl;

cout << "Token: " << curr\_token;

cout << "Token String: " << token\_str << endl;

for(int i = 0; i < (int)table\_parse[curr\_row].terminal.size() && !find\_terminal; i++)

{

cout << "Scan " << table\_parse[curr\_row].terminal[i] << " : ";

if(table\_parse[curr\_row].terminal[i] == token\_str)

find\_terminal = true;

cout << find\_terminal << endl;

}

// Если нашли

if(find\_terminal)

{

if(table\_parse[curr\_row].stack\_)

parse\_stack.push(curr\_row + 1);

if(table\_parse[curr\_row].accept)

{

if((token\_str == "var" || token\_str == "const") &&

(get\_token\_text(next\_token) == "=" ||

(get\_token\_text(next\_token) == "[" && !have\_type)))

need\_postfix = true;

if((token\_str == "var" || token\_str == "const") && have\_type && get\_token\_text(next\_token) == "[")

need\_array\_resize = true;

// Обработка необъявленного типа

if(!have\_type && token\_str == "var")

{

lexeme lex\_var;

identifiers.get\_lexeme(curr\_token.place, curr\_token.chain, lex\_var);

if(lex\_var.type == 0)

{

error\_flag = true;

out\_error << "Syntax Error: Undefined identifier \"" << lex\_var.name << "\"" << endl;

cerr << "Syntax Error: Undefined identifier \"" << lex\_var.name << "\"" << endl;

}

}

// Обработка унарного минуса

bool flag\_unary\_minus = false;

if(curr\_row == 54 && need\_postfix)

{

int one\_hash, one\_chain;

constants.add("-1");

constants.get\_location("-1", one\_hash, one\_chain);

code\_expr\_infix.push\_back(token(6, one\_hash, one\_chain));

int mult\_pos;

operations.get\_num("\*", mult\_pos);

code\_expr\_infix.push\_back(token(2, mult\_pos, -1));

flag\_unary\_minus = true;

}

if(need\_postfix && !flag\_unary\_minus)

code\_expr\_infix.push\_back(curr\_token);

// Обработка унарного минуса

flag\_unary\_minus = false;

if(curr\_row == 54 && need\_array\_resize)

{

int one\_hash, one\_chain;

constants.add("-1");

constants.get\_location("-1", one\_hash, one\_chain);

array\_resize\_expr\_infix.push\_back(token(6, one\_hash, one\_chain));

int mult\_pos;

operations.get\_num("\*", mult\_pos);

array\_resize\_expr\_infix.push\_back(token(2, mult\_pos, -1));

flag\_unary\_minus = true;

}

if(need\_array\_resize && !flag\_unary\_minus)

{

array\_resize\_expr\_infix.push\_back(curr\_token);

if(token\_str == "=" || token\_str == "+=" || token\_str == "-=" || token\_str == "\*=")

{

error\_flag = true;

out\_error << "Syntax Error: Can`t assign to array \"" << get\_token\_text(array\_resize\_expr\_infix[0]) << "\"" << endl;

cerr << "Syntax Error: Can`t assign to array \"" << get\_token\_text(array\_resize\_expr\_infix[0]) << "\"" << endl;

}

}

// Если закончили разбор присваивания или части объявления

if(token\_str == ";" || token\_str == ",")

{

// Добавим все, что разобрали, в постфиксную запись

if(!make\_postfix(code\_expr\_infix))

error\_flag = true;

if(need\_array\_resize && !error\_flag)

{

if(!make\_postfix(array\_resize\_expr\_infix))

error\_flag = true;

}

// Сбрасываем все флаги

code\_expr\_infix.clear();

array\_resize\_expr\_infix.clear();

need\_postfix = false;

need\_array\_resize = false;

}

// Если закончили разбор объявления, сбросим флаг объявления

if(token\_str == ";")

have\_type = false;

// Если попался тип, запоминаем его

if(token\_str == "int" || token\_str == "float")

{

have\_type = true;

if(token\_str == "int")

type\_type = 1;

if(token\_str == "float")

type\_type = 2;

}

// Заносим тип в таблицу идентификаторов

if(token\_str == "var" && have\_type && curr\_row == 69)

identifiers.set\_type(get\_token\_text(curr\_token), type\_type);

eof\_flag = in\_token.eof();

curr\_token = next\_token;

if(!eof\_flag)

in\_token >> next\_token;

}

if(table\_parse[curr\_row].return\_)

{

if(!parse\_stack.empty())

{

curr\_row = parse\_stack.top();

parse\_stack.pop();

}

else // Если внезапно стек пуст

{

error\_flag = true;

cerr << "Syntax Error: Parse stack is empty!" << endl;

cerr << "Return requested by row " << curr\_row << " at token " << curr\_token

<< " (value = \"" << get\_token\_text(curr\_token) << "\")" << endl;

out\_error << "Syntax Error: Parse stack is empty!" << endl;

out\_error << "Return requested by row " << curr\_row << " at token " << curr\_token

<< " (value = \"" << get\_token\_text(curr\_token) << "\")" << endl;

}

}

else

curr\_row = table\_parse[curr\_row].jump;

}

else

{

// Если ошибка безальтернативная

if(table\_parse[curr\_row].error)

{

error\_flag = true;

out\_error << "Syntax Error: Unexpected terminal \"" << get\_token\_text(curr\_token) << "\"" << endl;

out\_error << "Must be: ";

for(int i = 0; i < (int)table\_parse[curr\_row].terminal.size(); i++)

out\_error << "\"" << table\_parse[curr\_row].terminal[i] << "\" ";

out\_error << endl;

cerr << "Syntax Error: Unexpected terminal \"" << get\_token\_text(curr\_token) << "\"" << endl;

cerr << "Must be: ";

for(int i = 0; i < (int)table\_parse[curr\_row].terminal.size(); i++)

cerr << "\"" << table\_parse[curr\_row].terminal[i] << "\" ";

cerr << endl;

}

else

{

curr\_row++;

}

}

};

// Если внезапно стек не пуст

if(!error\_flag && !parse\_stack.empty())

{

error\_flag = true;

cerr << "Syntax Error: Parse stack isn`t empty!" << endl;

cerr << "Size = " << parse\_stack.size() << endl;

cerr << "Contains: ";

out\_error << "Syntax Error: Parse stack isn`t empty!" << endl;

out\_error << "Size = " << parse\_stack.size() << endl;

out\_error << "Contains: ";

while(!parse\_stack.empty())

{

cerr << "\"" << parse\_stack.top() << "\" " << endl;

out\_error << "\"" << parse\_stack.top() << "\" " << endl;

parse\_stack.pop();

}

cerr << endl;

out\_error << endl;

}

in\_token.close();

out\_error.close();

return !error\_flag;

}

// Построение постфиксной записи

bool translator::make\_postfix(vector<token> t)

{

stack<string> stack\_temp;

bool error\_flag = false;

int index = 0;

while(index < (int)t.size() && !error\_flag)

{

int i;

for(i = index; i < (int)t.size() && !error\_flag && get\_token\_text(t[i]) != ";" && get\_token\_text(t[i]) != ","; i++)

{

string token\_text = get\_token\_text(t[i]);

if(t[i].table == 5 || t[i].table == 6)

{

postfix\_record.push\_back(postfix\_elem(token\_text));

}

else if(token\_text == "(" || token\_text == "[")

{

stack\_temp.push(token\_text);

}

else if(token\_text == ")")

{

while(!stack\_temp.empty() && stack\_temp.top() != "(")

{

string tmpstr = stack\_temp.top();

postfix\_record.push\_back(postfix\_elem(tmpstr));

stack\_temp.pop();

}

if(stack\_temp.empty())

{

cerr << "Syntax Error: Unexpected \")\" !" << endl;

out\_error << "Syntax Error: Unexpected \")\" !" << endl;

error\_flag = true;

}

else

{

stack\_temp.pop();

}

}

else if(token\_text == "]")

{

while(!stack\_temp.empty() && stack\_temp.top() != "[")

{

string tmpstr = stack\_temp.top();

postfix\_record.push\_back(postfix\_elem(tmpstr));

stack\_temp.pop();

}

if(stack\_temp.empty())

{

cerr << "Syntax Error: Unexpected \"]\" !" << endl;

out\_error << "Syntax Error: Unexpected \"]\" !" << endl;

error\_flag = true;

}

else

{

postfix\_record.push\_back(postfix\_elem("[]", 3));

stack\_temp.pop();

}

}

else if(t[i].table == 2)

{

while(!stack\_temp.empty() && priority\_le(token\_text, stack\_temp.top()))

{

string tmpstr = stack\_temp.top();

postfix\_record.push\_back(postfix\_elem(tmpstr));

stack\_temp.pop();

}

stack\_temp.push(token\_text);

}

}

if(error\_flag)

{

postfix\_record.clear();

return false;

}

else

{

while(!stack\_temp.empty() &&

stack\_temp.top() != "(" && stack\_temp.top() != ")" &&

stack\_temp.top() != "[" && stack\_temp.top() != "]")

{

string tmpstr = stack\_temp.top();

postfix\_record.push\_back(postfix\_elem(tmpstr, 1));

stack\_temp.pop();

}

if(!stack\_temp.empty())

{

cerr << "Syntax Error: Brackets balance error!" << endl;

out\_error << "Syntax Error: Brackets balance error!" << endl;

error\_flag = true;

}

}

if(error\_flag)

{

postfix\_record.clear();

return false;

}

if(postfix\_record[postfix\_record.size() - 1].id == "[]")

{

postfix\_record[postfix\_record.size() - 1] = postfix\_elem("[\*]", 2);

}

index = i + 1;

postfix\_record.push\_back(postfix\_elem(";", 4));

}

return true;

}

// Печать постфиксной записи в файл и на экран

void translator::postfix\_print(string file\_tree)

{

ofstream out(file\_tree.c\_str());

cout << "Postfix notation:" << endl;

for(int i = 0; i < (int)postfix\_record.size(); i++)

{

cout << postfix\_record[i] << " ";

out << postfix\_record[i] << " ";

}

cout << endl;

out.close();

}

// Сравнение приоритетов операций

bool translator::priority\_le(string what, string with\_what)

{

int pw = 0, pww = 0;

if(what == "=" || what == "+=" || what == "-=" || what == "\*=") pw = 10;

else if(what == "!=" || what == ">" || what == "<" || what == "==") pw = 20;

else if(what == "+" || what == "-") pw = 30;

else pw = 40;

if(with\_what == "=" || with\_what == "+=" || with\_what == "-=" || with\_what == "\*=") pww = 10;

else if(with\_what == "!=" || with\_what == ">" || with\_what == "<" || with\_what == "==") pww = 20;

else if(with\_what == "+" || with\_what == "-") pww = 30;

else if(with\_what == "\*") pww = 40;

if(pw <= pww) return true;

return false;

};