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 C++ 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
```

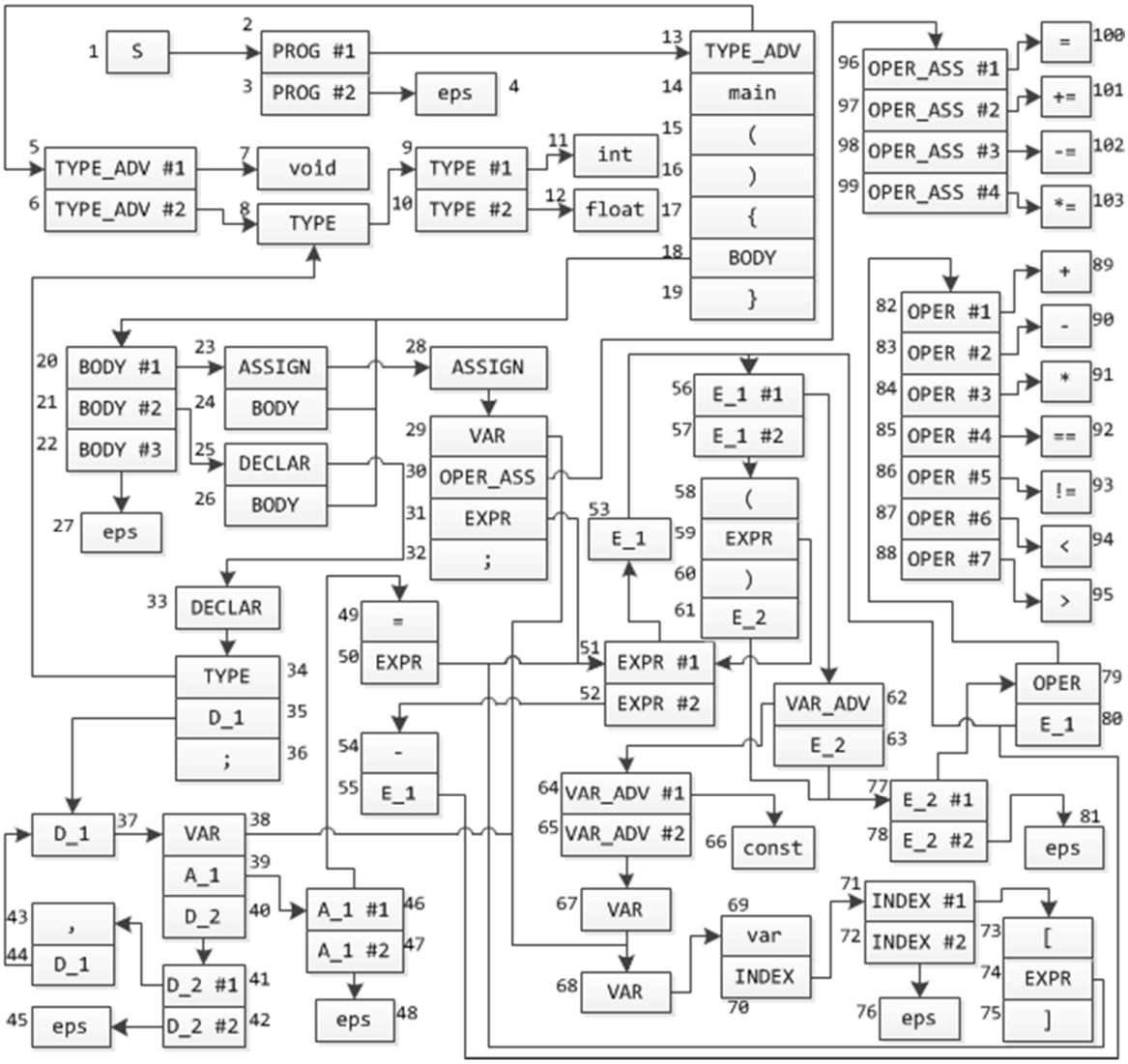
```
# Тело программы
BODY -> DECLAR BODY                 # Объявление переменной
BODY -> ASSIGN BODY                 # Присваивание
BODY -> eps

# 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;
};
```