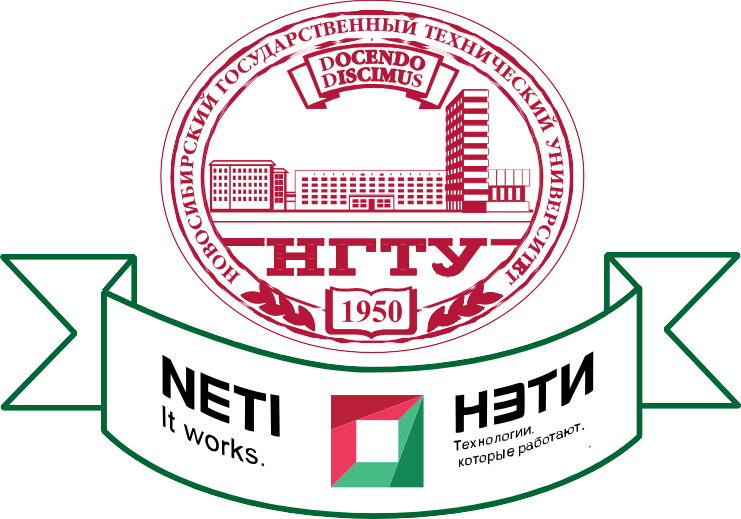
|  |  |
| --- | --- |
| Министерство науки и высшего образования  Российской Федерации | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования | |
| «НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ» | |
| Кафедра теоретической и прикладной информатики | |
| Лабораторная работа № 3 | |
| по дисциплине «Языки программирования и методы трансляции» | |
| **РАЗРАБОТКА И РЕАЛИЗАЦИЯ БЛОКА СИНТАКСИЧЕСКОГО АНАЛИЗА** | |
| Факультет: | ПМИ |
| Группа: | ПМ-83 |
| Бригада: | 5 |
| Студенты: | Кравец Екатерина |
|  | Шумский Владислав  Ефимцев Фёдор |
| Преподаватель: | Еланцева И. Л. |
| Новосибирск | |
| 2021 |  |



# Цель работы

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

1. **Задача**

## 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.

Подмножество языка С++ включает:

* данные типа **int**;
* инструкции описания переменных;
* операторы присваивания в любой последовательности, **switch** любой вложенности и в любой последовательности;
* операции **+, – , \*, ==, !=, <, >.**

1. **Структура входных и выходных данных**

**Входные:**

Таблицы лексем

Файл токенов tokenFile.txt:

Каждый токен состоит из 3 чисел: номер таблицы и 2 индекса для каждой лексемы.

0 – таблица ключевых слов

1 – таблица операций

2 – таблица разделителей

3 – таблица констант

4 – таблица идентификаторов

**Выходные:**

При успешной обработке входной программы формируется постфиксная запись в файле postfix.txt, ошибки записываются в файл errorFile.txt

**Структура постфиксной записи:**

switch(A1){

case e1: C1; break;

case e2: C2; break;

default: D; break;

}

E1 L0 CJ C1 L1 UJ L0: E2 L2 CJ C2< место для вложенных switch> L3 UJ L2: <место для следующих case текущего switch> D L3: L1:

L0, L1, L2, L3 – метки условного оператора с логическими выражениями E1, E2 где Е1: A1 == e1, E2: A1 == e2;

CJ – условный переход по значению «ложь» (УПЛ);

C1, C2, D – операторы;

UJ – безусловный переход (БП);

1. **Грамматика входного языка**

// Начальный символ

S -> PROG

// Программа

PROG -> int main ( ) { BODY return 0; }

PROG -> eps

// Тело программы

BODY -> DECLAR ; BODY // Объявление переменной

BODY -> ASSIGN ; BODY // Присваивание

BODY -> SWITCH BODY // Оператор switch

BODY -> eps

// Объявление переменной

DECLAR -> int NAME1

NAME1 -> var VALUE NAME2 // После типа обязательно имя

VALUE -> = EXPR // Инициализация

VALUE -> eps // Без инициализации

NAME2 -> , NAME1 // Перечисление нескольких переменных

NAME2 -> eps // Без перечисления

// Присваивание

ASSIGN -> var = EXPR // Переменная, операция, выражение

// Оператор switch

SWITCH -> switch (EXPR) { CASE default: BODY break;} // Оператор switch

CASE -> case const: BODY break; CASE2

CASE2 -> eps // Конец операторов case в switch

CASE2 -> CASE / /Либо следующий case

// Для работы с переменными, константами

VAR\_ADV -> var // Переменная

VAR\_ADV -> const // Константа

// Выражение

EXPR -> VAR\_ADV EXPR2 // Переменная / константа и дальнейшие операции

EXPR -> ( EXPR ) EXPR2 // Выражение в скобках

EXPR2 -> eps // Конец выражения

EXPR2 -> OPER EXPR // Либо знак операции и продолжение выражения

// Операции

OPER -> +

OPER -> -

OPER -> \*

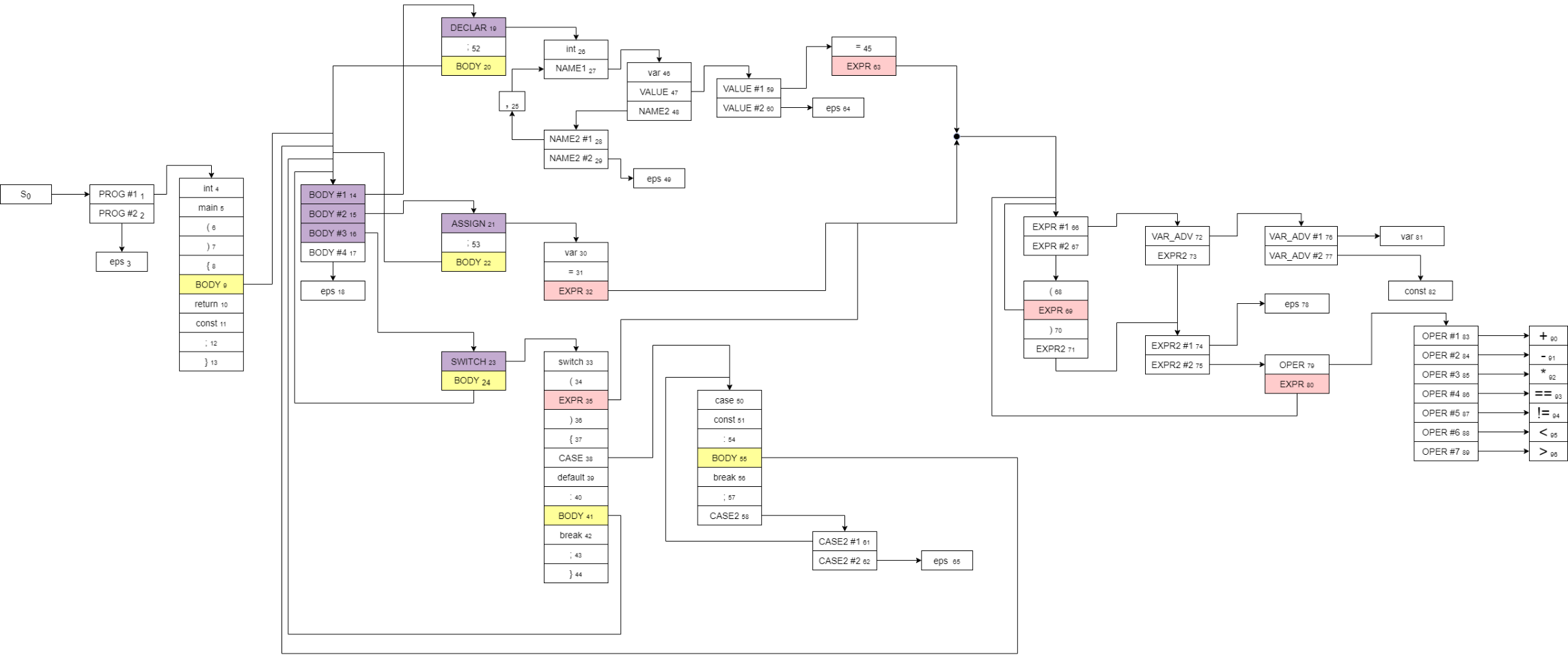
OPER -> ==

OPER -> !=

OPER -> <

OPER -> >

1. **Схема разбора**



1. **Таблица разбора**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№** | **terminals** | **jump** | **accept** | **stack** | **return** | **error** |
| **0** | int eps | 1 | 0 | 0 | 0 | 1 |
| **1** | int | 4 | 0 | 0 | 0 | 0 |
| **2** | eps | 3 | 0 | 0 | 0 | 1 |
| **3** | eps | -1 | 0 | 0 | 1 | 1 |
| **4** | int | 5 | 1 | 0 | 0 | 1 |
| **5** | main | 6 | 1 | 0 | 0 | 1 |
| **6** | ( | 7 | 1 | 0 | 0 | 1 |
| **7** | ) | 8 | 1 | 0 | 0 | 1 |
| **8** | { | 9 | 1 | 0 | 0 | 1 |
| **9** | int var switch return | 14 | 0 | 1 | 0 | 1 |
| **10** | return | 11 | 1 | 0 | 0 | 1 |
| **11** | const | 12 | 1 | 0 | 0 | 1 |
| **12** | ; | 13 | 1 | 0 | 0 | 1 |
| **13** | } | -1 | 1 | 0 | 1 | 1 |
| **14** | int | 19 | 0 | 0 | 0 | 0 |
| **15** | var | 21 | 0 | 0 | 0 | 0 |
| **16** | switch | 23 | 0 | 0 | 0 | 0 |
| **17** | return break | 18 | 0 | 0 | 0 | 1 |
| **18** | return break | -1 | 0 | 0 | 1 | 1 |
| **19** | int | 26 | 0 | 1 | 0 | 1 |
| **20** | int var switch return break | 14 | 0 | 0 | 0 | 1 |
| **21** | var | 30 | 0 | 1 | 0 | 1 |
| **22** | int var switch return break | 14 | 0 | 0 | 0 | 1 |
| **23** | switch | 33 | 0 | 1 | 0 | 1 |
| **24** | int var switch return break | 14 | 0 | 0 | 0 | 1 |
| **25** | , | 27 | 1 | 0 | 0 | 1 |
| **26** | int | 27 | 1 | 0 | 0 | 1 |
| **27** | var | 46 | 0 | 0 | 0 | 1 |
| **28** | , | 25 | 0 | 0 | 0 | 0 |
| **29** | ; | 49 | 0 | 0 | 0 | 1 |
| **30** | var | 31 | 1 | 0 | 0 | 1 |
| **31** | = | 32 | 1 | 0 | 0 | 1 |
| **32** | var const ( | 66 | 0 | 0 | 0 | 1 |
| **33** | switch | 34 | 1 | 0 | 0 | 1 |
| **34** | ( | 35 | 1 | 0 | 0 | 1 |
| **35** | var const ( | 66 | 0 | 1 | 0 | 1 |
| **36** | ) | 37 | 1 | 0 | 0 | 1 |
| **37** | { | 38 | 1 | 0 | 0 | 1 |
| **38** | case | 50 | 0 | 1 | 0 | 1 |
| **39** | default | 40 | 1 | 0 | 0 | 1 |
| **40** | : | 41 | 1 | 0 | 0 | 1 |
| **41** | int var switch break | 14 | 0 | 1 | 0 | 1 |
| **42** | break | 43 | 1 | 0 | 0 | 1 |
| **43** | ; | 44 | 1 | 0 | 0 | 1 |
| **44** | } | -1 | 1 | 0 | 1 | 1 |
| **45** | = | 63 | 1 | 0 | 0 | 1 |
| **46** | var | 47 | 1 | 0 | 0 | 1 |
| **47** | ; , = | 59 | 0 | 1 | 0 | 1 |
| **48** | , ; | 28 | 0 | 0 | 0 | 1 |
| **49** | ; | -1 | 0 | 0 | 1 | 1 |
| **50** | case | 51 | 1 | 0 | 0 | 1 |
| **51** | const | 54 | 1 | 0 | 0 | 1 |
| **52** | ; | 20 | 1 | 0 | 0 | 1 |
| **53** | ; | 22 | 1 | 0 | 0 | 1 |
| **54** | : | 55 | 1 | 0 | 0 | 1 |
| **55** | int var switch break | 14 | 0 | 1 | 0 | 1 |
| **56** | break | 57 | 1 | 0 | 0 | 1 |
| **57** | ; | 58 | 1 | 0 | 0 | 1 |
| **58** | case default | 61 | 0 | 0 | 0 | 1 |
| **59** | = | 45 | 0 | 0 | 0 | 0 |
| **60** | , ; | 64 | 0 | 0 | 0 | 1 |
| **61** | case | 50 | 0 | 0 | 0 | 0 |
| **62** | default | 65 | 0 | 0 | 0 | 1 |
| **63** | var const ( | 66 | 0 | 0 | 0 | 1 |
| **64** | , ; | -1 | 0 | 0 | 1 | 1 |
| **65** | default | -1 | 0 | 0 | 1 | 1 |
| **66** | var const | 72 | 0 | 0 | 0 | 0 |
| **67** | ( | 68 | 0 | 0 | 0 | 1 |
| **68** | ( | 69 | 1 | 0 | 0 | 1 |
| **69** | var const ( | 66 | 0 | 1 | 0 | 1 |
| **70** | ) | 71 | 1 | 0 | 0 | 1 |
| **71** | + - \* == != < > , ; ) | 74 | 0 | 0 | 0 | 1 |
| **72** | var const | 76 | 0 | 1 | 0 | 1 |
| **73** | + - \* == != < > , ; ) | 74 | 0 | 0 | 0 | 1 |
| **74** | , ; ) | 78 | 0 | 0 | 0 | 0 |
| **75** | + - \* == != < > | 79 | 0 | 0 | 0 | 1 |
| **76** | var | 81 | 0 | 0 | 0 | 0 |
| **77** | const | 82 | 0 | 0 | 0 | 1 |
| **78** | , ; ) | -1 | 0 | 0 | 1 | 1 |
| **79** | + - \* == != < > | 83 | 0 | 1 | 0 | 1 |
| **80** | var const ( | 66 | 0 | 0 | 0 | 1 |
| **81** | var | -1 | 1 | 0 | 1 | 1 |
| **82** | const | -1 | 1 | 0 | 1 | 1 |
| **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 | 0 |
| **89** | > | 96 | 0 | 0 | 0 | 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** | > | -1 | 1 | 0 | 1 | 1 |

1. **Текст программы**

#include <fstream>

#include <string>

#include <vector>

#include <iostream>

#include <sstream>

#include <algorithm>

#include <stack>

#include <queue>

#include <map>

using namespace std;

struct VarTableElem {

VarTableElem(string name, int value, int typeLevel) :name(name), valueLevel(-1),typeLevel(-1) {}

VarTableElem() : name(""), valueLevel(-1), typeLevel(-1) {}

string name;

int valueLevel;//уровень задания значения переменной, -1 - не задано

int typeLevel; //уровень действия переменной, -1 - не объявлена

};

struct place {

int i;

int j;

place(int i, int j) :i(i), j(j) {}

place() :i(), j() {}

};

struct token {

int table;

int i;

int j;

token(int \_table, int \_i, int \_j) {

table = \_table;

i = \_i;

j = \_j;

}

token() {}

};

class ConstTable

{

public:

vector<string> array;

ConstTable(string fileName)

{

readTable(fileName);

};

~ConstTable()

{

array.~vector();

}

int findElem(string Name)

{

auto res = find(array.begin(), array.end(), Name);

if (res == array.end())

return -1;

else

return res - array.begin();

}

string getElem(int index) {

return array[index];

}

private:

void readTable(string fileName)

{

string tmp;

fstream in(fileName);

while (!(in.eof())) {

in >> tmp;

array.push\_back(tmp);

}

in.close();

}

};

class TableVar

{

public:

vector <vector<VarTableElem>> array;

TableVar()

{

array.resize(52);

};

~TableVar()

{

array.~vector();

}

place findElem(string name)

{

place pl;

auto hash = getHash(name);

pl.i = hash;

if (array[hash].size()) {

auto res = find\_if(array[hash].begin(), array[hash].end(), [&](const VarTableElem& s)-> bool {return s.name == name; });

if (res == array[hash].end())

{

array[hash].push\_back(VarTableElem(name, 0,0));

pl.j = array[hash].size() - 1;

}

else pl.j = res - array[hash].begin();

}

else {

array[hash].push\_back(VarTableElem(name, 0,0));

pl.j = 0;

}

return pl;

}

VarTableElem getElem(place pl) {

return array[pl.i][pl.j];

}

private:

int getHash(string h) {

char len;

if (h[0] >= 'A' && h[0] <= 'Z')

len = h[0] - 'A';

if (h[0] >= 'a' && h[0] <= 'z')

len = h[0] - 'a' + 26;

return (int)len;

}

};

class TableInt

{

public:

vector<vector<int>> array;

TableInt()

{

array.resize(19);

};

~TableInt() {

array.~vector();

}

place findElem(string value)

{

int tmp = stoi(value);

place pl;

auto hash = getHash(value);

pl.i = hash;

if (array[hash].size()) {

auto res = find(array[hash].begin(), array[hash].end(), tmp);

if (res == array[hash].end())

{

array[hash].push\_back(tmp);

pl.j = array[hash].size() - 1;

}

else pl.j = res - array[hash].begin();

}

else {

array[hash].push\_back(tmp);

pl.j = 0;

}

return pl;

}

int getElem(place pl) {

return array[pl.i][pl.j];

}

private:

int getHash(string h) {

int len;

if (h[0] == '-')

len = (int)h[1] - '0' + 9;

else len = (int)h[0] - '0';

return len;

}

};

class Lexeme

{

public:

ConstTable\* keyWords = new ConstTable("KeyWords.txt");

ConstTable\* operations = new ConstTable("Operation.txt");

ConstTable\* delimiters = new ConstTable("Delimiter.txt");

TableInt\* tableInt = new TableInt();

TableVar\* tableVar = new TableVar();

bool syntaxSuccess=false;

Lexeme(string fileName)

{

syntaxSuccess=createTokens(fileName, "tokenFile.txt", "errorFile.txt");

}

Lexeme(){}

string str;

int value;

VarTableElem var;

bool createTokens(string codeName, string tokenName, string errorName)

{

token tmp;

ofstream tokenFile(tokenName), errorFile(errorName);

ifstream code(codeName);

char ch;

string str;

int flag;

bool unexpected = true;

place pl;

if (code.peek() != EOF)

{

code.get(ch);

while (1)

{

unexpected = true;

if (ch >= 'a' && ch <= 'z' || ch >= 'A' && ch <= 'Z')

{

do

{

str += ch;

if (code.peek() == EOF) { return true; }

code.get(ch);

} while (ch >= 'a' && ch <= 'z' || ch >= 'A' && ch <= 'Z' || ch >= '0' && ch <= '9' || ch == '\_');

flag = keyWords->findElem(str);

if (flag != -1)

{

tmp.table = 0;

tmp.i = flag;

tmp.j = -1;

tokenFile << endl <<tmp.table<<" "<<tmp.i<<" "<<tmp.j;

unexpected = false;

}

else

{

pl = tableVar->findElem(str);

tmp.table = 4;

tmp.i = pl.i;

tmp.j = pl.j;

tokenFile << endl<< tmp.table << " " << tmp.i << " " << tmp.j;

unexpected = false;

}

str.clear();

}

if (ch >= '0' && ch <= '9')

{

do

{

str += ch;

if (code.peek() == EOF) { return true; }

code.get(ch);

if (ch >= 'a' && ch <= 'z' || ch >= 'A' && ch <= 'Z')

{

errorFile << "Lexical error: unexpected const "<< str+ch <<endl;

return false;

}

} while (ch >= '0' && ch <= '9');

pl = tableInt->findElem(str);

tmp.table = 3;

tmp.i = pl.i;

tmp.j = pl.j;

tokenFile << endl<< tmp.table << " " << tmp.i << " " << tmp.j;

unexpected = false;

str.clear();

}

if (ch == '=')

{

str = ch;

if (code.peek() == EOF) {return true; }

code.get(ch);

tmp.table = 1;

tmp.i = operations->findElem(str);

tmp.j = -1;

tokenFile << endl << tmp.table << " " << tmp.i << " " << tmp.j;

unexpected = false;

str.clear();

while (ch == ' ')

{

if (code.peek() == EOF) { return true; }

code.get(ch);

}

if (ch == '-' || ch == '+')

{

if (ch == '-') str = ch;

if (code.peek() == EOF) { return true; }

code.get(ch);

}

}

if (ch == '+' || ch == '-' || ch == '\*' || ch == '!' || ch == '<' || ch == '>')

{

str += ch;

if (code.peek() == EOF) { return true; }

code.get(ch);

if (ch == '+' || ch == '-' || ch == '=')

{

str += ch;

if (code.peek() == EOF) { return true; }

code.get(ch);

}

flag = operations->findElem(str);

if (flag != -1)

{

tmp.table = 1;

tmp.i = flag;

tmp.j = -1;

tokenFile << endl<< tmp.table << " " << tmp.i << " " << tmp.j;

unexpected = false;

}

else { errorFile << "Lexical error: unexpected symbol "<< ch<< endl; return false; }

str.clear();

}

if (ch == '(' || ch == ')' || ch == '{' || ch == '}' || ch == ':' || ch == ';' || ch == ',' || ch == '.')

{

str += ch;

flag = delimiters->findElem(str);

if (flag != -1)

{

tmp.table = 2;

tmp.i = flag;

tmp.j = -1;

tokenFile << endl<< tmp.table << " " << tmp.i << " " << tmp.j;

unexpected = false;

}

else { errorFile << "Lexical error: unexpected symbol" << str << endl; return false; }

str.clear();

if (code.peek() == EOF) { return true; }

code.get(ch);

}

if (ch == '/')

{

if (code.peek() == EOF) { errorFile << "Lexical error: unexpected symbol"; return false; }

code.get(ch);

if (ch == '/')

do

{

if (code.peek() == EOF) { return true; }

code.get(ch);

} while (ch != '\n');

else if (ch == '\*')

{

char ch2 = ' ';

do

{

if (code.peek() == EOF) { errorFile << "Lexical error: unclosed comment"; return false; }

code.get(ch);

if (ch == '\*')

{

if (code.peek() == EOF) { errorFile << "Lexical error: unclosed comment"; return false; }

code.get(ch2);

}

} while (!(ch == '\*' && ch2 == '/'));

}

else { errorFile << "Lexical error: open comment error\n"; return false; }

unexpected = false;

if (code.peek() == EOF) { return true; }

code.get(ch);

}

while (ch == '\n' || ch == '\t' || ch == ' ')

{

unexpected = false;

if (code.peek() == EOF) { return true; }

code.get(ch);

}

if (unexpected)

{

errorFile << "Lexical error: unexpected symbol " << ch << endl; return false;

}

}

}

tokenFile.close();

errorFile.close();

code.close();

}

void getElemByToken(token tmp) {

switch (tmp.table) {

case 0: str = keyWords->getElem(tmp.i); break;

case 1: str = operations->getElem(tmp.i); break;

case 2: str = delimiters->getElem(tmp.i); break;

case 3: value = tableInt->getElem(place(tmp.i, tmp.j)); break;

case 4: var = tableVar->getElem(place(tmp.i, tmp.j)); break;

}

}

};

class Translator

{

public:

Lexeme\* oneLexeme;

Translator(string ParseTable, string tokenFile) {

oneLexeme = new Lexeme("code.txt");

if (oneLexeme->syntaxSuccess) {

readParseTable(ParseTable);

LLParse(tokenFile);

ofstream postfix;

postfix.open("postfix.txt");

for (int i = 0; i < toPostfixFile.size(); i++)

postfix << toPostfixFile[i] << " ";

postfix.close();

}

}

vector<string> toPostfixFile;

map<string, int> priority = { {"+", 2}, {"-",2}, {"\*",3},{"=",0},{"==",1},{"!=",1 }, {"<",1}, {">",1},{",",0} };

stack<int> States;

// Структура элемент таблицы разбора

struct tableParseElem

{

vector<string> terminal\_; // Терминалы

int jump\_; // Переход

int accept\_; // Принимать или нет

int stack\_; // Cтек

int return\_; // Возвращать или нет

int error\_; // Oшибка

};

vector<tableParseElem> tableParse;

void readParseTable(string fileName) {

string tmp;

fstream in(fileName);

while (!(in.eof())) {

getline(in, tmp);

string t;

istringstream ss(tmp);

vector<string> v;

while (ss >> t)

v.push\_back(t);

int size = v.size();

tableParseElem one;

int i = 0;

for (i = 0; i < size - 5; i++)

one.terminal\_.push\_back(v[i]);

i = size - 5;

one.jump\_ = stoi(v[i]);

one.accept\_ = stoi(v[i + 1]);

one.stack\_ = stoi(v[i + 2]);

one.return\_ = stoi(v[i + 3]);

one.error\_ = stoi(v[i + 4]);

tableParse.push\_back(one);

}

in.close();

}

string readToken(int tableNum, int i, int j) {

token oneToken(tableNum, i, j);

oneLexeme->getElemByToken(oneToken);

string str;

if (tableNum == 0 || tableNum == 1 | tableNum == 2) {

str = oneLexeme->str;

}

if (tableNum == 3) str = "const";

if (tableNum == 4) str = "var";

return str;

}

string makePostfix(vector<token> infix) {

stack<string> tmp;

string tmpStr;

token curElem;

queue<string> postfix;

for (int i = 0; i < infix.size(); i++) {

curElem = infix[i];

oneLexeme->getElemByToken(curElem);

if (curElem.table == 3) {

int value = oneLexeme->value;

string str = to\_string(value);

postfix.push(str);

}

else {

if (curElem.table == 4) {

string str = oneLexeme->var.name;

postfix.push(str);

}

else { //не числа и не переменные

oneLexeme->getElemByToken(curElem);

string curStr = oneLexeme->str;

if (curStr == "(") {

tmp.push(curStr);

}

else {

if (curStr == ")") {

while (tmp.top() != "(") {

postfix.push(tmp.top());

tmp.pop();

}

tmp.pop();

}

else {

//все операторы кроме скобок

if (tmp.empty() || tmp.top() == "(")

tmp.push(curStr);

else {

int p\_in = priority[curStr];

int p\_top = priority[tmp.top()];

if (p\_in > p\_top) tmp.push(curStr);

else {

while ((tmp.top() != "(" || priority[tmp.top()] >= p\_in)) {

postfix.push(tmp.top());

tmp.pop();

if (tmp.empty()) break;

}

tmp.push(curStr);

}

}

}

}

}

}

}

//перегрузка остатков из стека в очередь

while (!tmp.empty()) {

postfix.push(tmp.top());

tmp.pop();

}

string str;

//печать постфиксной очереди

while (!postfix.empty()) {

str = str + postfix.front();

str = str + " ";

postfix.pop();

}

return str;

}

bool LLParse(string tokenFile) {

bool errorFlag = false;

int curState = 0;

ifstream in(tokenFile);

ofstream err;

err.open("errorFile.txt", ios::app);

string tmp;

int tableNum, i, j;

if (in.peek() == EOF)

{

return true;

}

in >> tableNum >> i >> j;

tmp = readToken(tableNum, i, j);

vector<token> infix;

bool postfixFlag = false;

//структуры для обработки switch

int globalCaseCount = 0;//для номера метки

stack<string>switchCond;//для выражений из switch

stack<string> endCase; //окончания case

stack<int>countCase; //количество case для каждого switch

int curLevel = 0;//текущий уровень действия переменных

stack<vector<int>> admitLevels;//список допустимых уровней для переменных на каждом текущем уровне

vector<int> tmpLevelVec;//для допустимых уровней переменных на текущем уровне

tmpLevelVec.push\_back(0);

admitLevels.push(tmpLevelVec);

//обработка состояний таблицы разбора

do {

//проверка столбца терминалов

auto res = find(tableParse[curState].terminal\_.begin(), tableParse[curState].terminal\_.end(), tmp);

if (res == tableParse[curState].terminal\_.end()) { //если не найден

if (tableParse[curState].error\_) {

err << "Syntax error: unexpected symbol, expected: ";

for (i = 0; i < tableParse[curState].terminal\_.size(); i++)

err << "'" << tableParse[curState].terminal\_[i] << "' ";

return false;

}

else {

curState++;

}

}

else {//если найден

if (tableParse[curState].accept\_) {

if (curState == 46 || curState == 30) {

auto curVar = oneLexeme->tableVar->getElem(place(i, j));

if (curState == 46) {

//если текущий левел есть в списке на вершине стека

if (curVar.typeLevel != -1)

//уже задан - curVar.typeLevel принадлежит admitLevels.top()

{

err << "Syntax error: redefinition of type of variable '" << curVar.name << "'" << endl;

errorFlag = true;

}

else oneLexeme->tableVar->array[i][j].typeLevel = admitLevels.top()[admitLevels.top().size() - 1];

//задание типа переменные

}

else {

//проверка типа

if (find(admitLevels.top().begin(), admitLevels.top().end(), curVar.typeLevel) == admitLevels.top().end())

//еще не задан - curVar.typeLevel не принадлежит admitLevels.top()

{

err << "Syntax error: undeclared variable '" << curVar.name << "'" << endl;

errorFlag = true;

}

}

string tmp2;

int tableNum2, i2, j2;

//считать новый токен

if (in.peek() != EOF) {

in >> tableNum2 >> i2 >> j2;

tmp2 = readToken(tableNum2, i2, j2);

}

if (tmp2 == "=") {

if (oneLexeme->tableVar->array[i][j].valueLevel == -1)//только если значение не было задано

oneLexeme->tableVar->array[i][j].valueLevel = curLevel;

infix.push\_back(token(tableNum, i, j));

postfixFlag = true;

}

tmp = tmp2;

i = i2;

j = j2;

tableNum = tableNum2;

}

else {

if (curState == 81) {

auto curVar = oneLexeme->tableVar->getElem(place(i, j));

if (find(admitLevels.top().begin(), admitLevels.top().end(), curVar.typeLevel) == admitLevels.top().end())

//еще не объявлено - curVar.typeLevel не принадлежит admitLevels.top()

{

err << "Syntax error: undeclared variable'" << curVar.name << "'" << endl;

errorFlag = true;

}

else {

if (find(admitLevels.top().begin(), admitLevels.top().end(), curVar.valueLevel) == admitLevels.top().end())

//еще не задано значение - curVar.typeLevel не принадлежит admitLevels.top()

{

err << "Syntax error: using variable without definition '" << curVar.name << "'" << endl;

errorFlag = true;

}

}

}

if (curState == 25) {

if (postfixFlag)

toPostfixFile.push\_back(makePostfix(infix));

postfixFlag = false;

infix.clear();

}

//скобка от switch

if (curState == 36) {

if (postfixFlag)

switchCond.push(makePostfix(infix));

postfixFlag = false;

infix.clear();

}

//новый case

if (curState == 51) {

oneLexeme->getElemByToken(token(tableNum, i, j));

toPostfixFile.push\_back(switchCond.top() + to\_string(oneLexeme->value) + " == ");

countCase.top()++;

toPostfixFile.push\_back(" L" + to\_string(globalCaseCount) + " CJ");

endCase.push(" L" + to\_string(globalCaseCount + 1) + ":");

endCase.push(" L" + to\_string(globalCaseCount) + ":");

endCase.push(" UJ");

endCase.push(" L" + to\_string(globalCaseCount + 1));

globalCaseCount += 2;

curLevel++;

tmpLevelVec.clear();

tmpLevelVec = admitLevels.top();

tmpLevelVec.push\_back(curLevel);

admitLevels.push(tmpLevelVec);

}

//break от текущего case

if (curState == 56) {

toPostfixFile.push\_back(endCase.top());

endCase.pop();

toPostfixFile.push\_back(endCase.top());

endCase.pop();

toPostfixFile.push\_back(endCase.top());

endCase.pop();

admitLevels.pop();//убрать из стека список допустимых уровней

}

//начало default

if (curState == 39) {

curLevel++;

tmpLevelVec.clear();

tmpLevelVec = admitLevels.top();

tmpLevelVec.push\_back(curLevel);

admitLevels.push(tmpLevelVec);

}

//break от default текущего switch

if (curState == 42) {

for (int ii = 0; ii < countCase.top(); ii++) {

toPostfixFile.push\_back(endCase.top());

endCase.pop();

}

countCase.pop();//удалить подсчет case на текущем switch

switchCond.pop();//удалить условие текущего switch

admitLevels.pop();//убрать из стека список допустимых уровней

}

//записать в инфикс

if (postfixFlag) infix.push\_back(token(tableNum, i, j));

//скобка от switch

if (curState == 34) {

postfixFlag = true;

countCase.push(0);

}

//считать новый токен

if (in.peek() != EOF) {

in >> tableNum >> i >> j;

tmp = readToken(tableNum, i, j);

}

}

}

if (tableParse[curState].stack\_) {

int stTmp;

switch (curState) {

case 19: stTmp = 52; break;

case 21: stTmp = 53; break;

default: stTmp = curState + 1;

}

States.push(stTmp);

}

if (tableParse[curState].jump\_ > 0) {

curState = tableParse[curState].jump\_;

}

else {

if (tableParse[curState].return\_) {

if (!States.empty()) {

curState = States.top();

States.pop();

if (curState == 52 || curState == 53) {

if (postfixFlag)

toPostfixFile.push\_back(makePostfix(infix));

postfixFlag = false;

infix.clear();

}

}

else {

err << "Syntax error: stack is empty!";

}

}

}

}

} while (in.peek() != EOF && !errorFlag);

if (!errorFlag) {

if (curState == 13) {

err << "Success: correct end of making postfix!";

return true;

}

else {

err << "Syntax error: uncorrect end";

return false;

}

}

in.close();

err.close();

}

};

void main()

{

auto trans = new Translator("ParseTable.txt", "tokenFile.txt");

system("pause");

}

1. **Результат работы программы:**

switch(A1){

case e1: C1; break;

case e2: C2; break;

default: D; break;

}

E1 L0 CJ C1 L1 UJ L0: E2 L2 CJ C2< место для вложенных switch> L3 UJ L2: <место для следующих case текущего switch> D L3: L1:

L0, L1, L2, L3 – метки условного оператора с логическими выражениями E1, E2 где Е1: A1 == e1, E2: A1 == e2;

CJ – условный переход по значению «ложь» (УПЛ);

C1, C2, D – операторы;

UJ – безусловный переход (БП);

|  |  |  |
| --- | --- | --- |
| Синтаксически верный исходный код | | |
| Исходный код | Файл postfix.txt | Файл ошибок errorFile.txt |
| int main(){  int a=5;  int b;  switch(a-5){  case 0: a=-1; break;  case 2: a=0; break;  default: a=-5; break;  }  return 0;  } | a 5 = a 5 - 0 == L0 CJ a -1 = L1 UJ L0: a 5 - 2 == L2 CJ a 0 = L3 UJ L2: a -5 = L3: L1: | Success: correct end of making postfix! |
| int main(){  int a=5;  int c;  switch(a-5){  case 0: a=-1; break;  case 2:  switch(a){  case 4: c=0; break;  case 3: int b=3; int d=b; c=3; break;  default: a=-10; break;  }  break;  default: break;  }  return 0;  } | a 5 = a 5 - 0 == L0 CJ a -1 = L1 UJ L0: a 5 - 2 == L2 CJ a 4 == L4 CJ c 0 = L5 UJ L4: a 3 == L6 CJ b 3 = d b = c 3 = L7 UJ L6: a -10 = L7: L5: L3 UJ L2: L3: L1: | Success: correct end of making postfix! |

|  |  |  |  |
| --- | --- | --- | --- |
| Обработка ошибок | | | |
| Исходный код | Ожидаемый код | Файл ошибок errorFile.txt |
| int main(){  int a=5;  int c;  switch(a-5){  case 0: a=-1; break;  case 2:  switch(a){  case 4: c=0; break;  case 3: int b=3; int d=b; c=3; break;  default: a=-10; break; | int main(){  int a=5;  int c;  switch(a-5){  case 0: a=-1; break;  case 2:  switch(a){  case 4: c=0; break;  case 3: int b=3; int d=b; c=3; break;  default: a=-10; break;  }  break;  default: break;  }  return 0;  } | Syntax error: uncorrect end  Неожиданное завершение файла |
| int main(){  int a=5;  int c;  switch(a-5){  case 0: a=-1; break;  case 2:  switch(a){  case 4: c=; break;  case 3: int b=3; int d=b; c=3; break;  default: a=-10; break;  }  break;  default: break;  }  return 0;  } | int main(){  int a=5;  int c;  switch(a-5){  case 0: a=-1; break;  case 2:  switch(a){  case 4: c= 0; break;  case 3: int b=3; int d=b; c=3; break;  default: a=-10; break;  }  break;  default: break;  }  return 0;  } | Syntax error: unexpected symbol, expected: 'var' 'const' '('  Неожиданный символ и список возможных терминалов |
| int main(){  int a=5;  switch(a-5){  case 0: a=-1; break;  case 2:  switch(a){  case 4: c=0; break;  case 3: int b=3; int d=b; c=3; break;  default: a=-10; break;  }  break;  default: break;  }  return 0;  } | int main(){  int a=5;  int c;  switch(a-5){  case 0: a=-1; break;  case 2:  switch(a){  case 4: c=0; break;  case 3: int b=3; int d=b; c=3; break;  default: a=-10; break;  }  break;  default: break;  }  return 0;  } | Syntax error: undeclared variable 'c'  Необъявленная переменная |
| int main(){  int a=5;  int c;  switch(a-5){  case 0: a=-1; break;  case 2:  switch(a){  case 4: a=c+10; break;  case 3: int b=3; int d=b; c=3; break;  default: a=-10; break;  }  break;  default: break;  }  return 0;  } |  | Syntax error: using variable without definition 'c'  Использование переменной без инициализации |
| int main(){  int abc=5;  int abc;  switch(a-5){  case 0: a=-1; break;  case 2:  switch(a){  case 4: a=c+10; break;  case 3: int b=3; int d=b; c=3; break;  default: a=-10; break;  }  break;  default: break;  }  return 0;  } |  | Syntax error: redefinition of type of variable 'abc'  Переопределение переменной |
| int main(){  int a=5;  switch(a-5){  case 0: a=-1; break;  case 2:  switch(a){  case 4: a=10; break;  case 3: int b=3; int d; break;  default: a=-10; break;  }  break;  default: b=10; break;  }  return 0;  } |  | Syntax error: undeclared variable 'b'  Ограничение области действия переменных |
| int main(){  int a=5;  int b;  switch(a-5){  case 0: a=-1; break;  case 2:  switch(a){  case 4: a=10; break;  case 3: b=3; int d; break;  default: a=-10; break;  }  break;  default: a=-5+b; break;  }  return 0;  } |  | Syntax error: using variable without definition 'b'  Ограничение области действия переменных |