|  |  |  |
| --- | --- | --- |
|  | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования «Новосибирский государственный технический университет» | | |
|  | | |
| Кафедра прикладной математики | | |
| Лабораторная работа № 2 | | |
| по дисциплине «ЯЗЫКИ ПРОГРАММИРОВАНИЯ И МЕТОДЫ ТРАНСЛЯЦИИ» | | |
| **РАЗРАБОТКА И РЕАЛИЗАЦИЯ БЛОКА ЛЕКСИЧЕСКОГО АНАЛИЗА** | | |
|  | | |
|  | Бригада 4 | затолоцкая юлия |
| Группа ПМ-91 | Барсукова наталья |
|  |  |
|  |  |
|  |  |
|  |  |
| Преподаватели | И.Л. Еланцева |
|  |  |
| Новосибирск, 2022 | | |

1. **Цели и задачи проекта**

Изучить методы лексического анализа. Получить представление о методах обработки лексических ошибок. Научиться проектировать сканер на основе детерминированных конечных автоматов.

1. **Вид, объем и источник исходных данных;**

Исходными данными для постоянных таблиц являются файлы, где перечислены элементы этих таблиц. В файлах каждый идентификатор начинается с новой строки.

Для переменных таблиц явных исходных данных нет. Они формируются динамически.

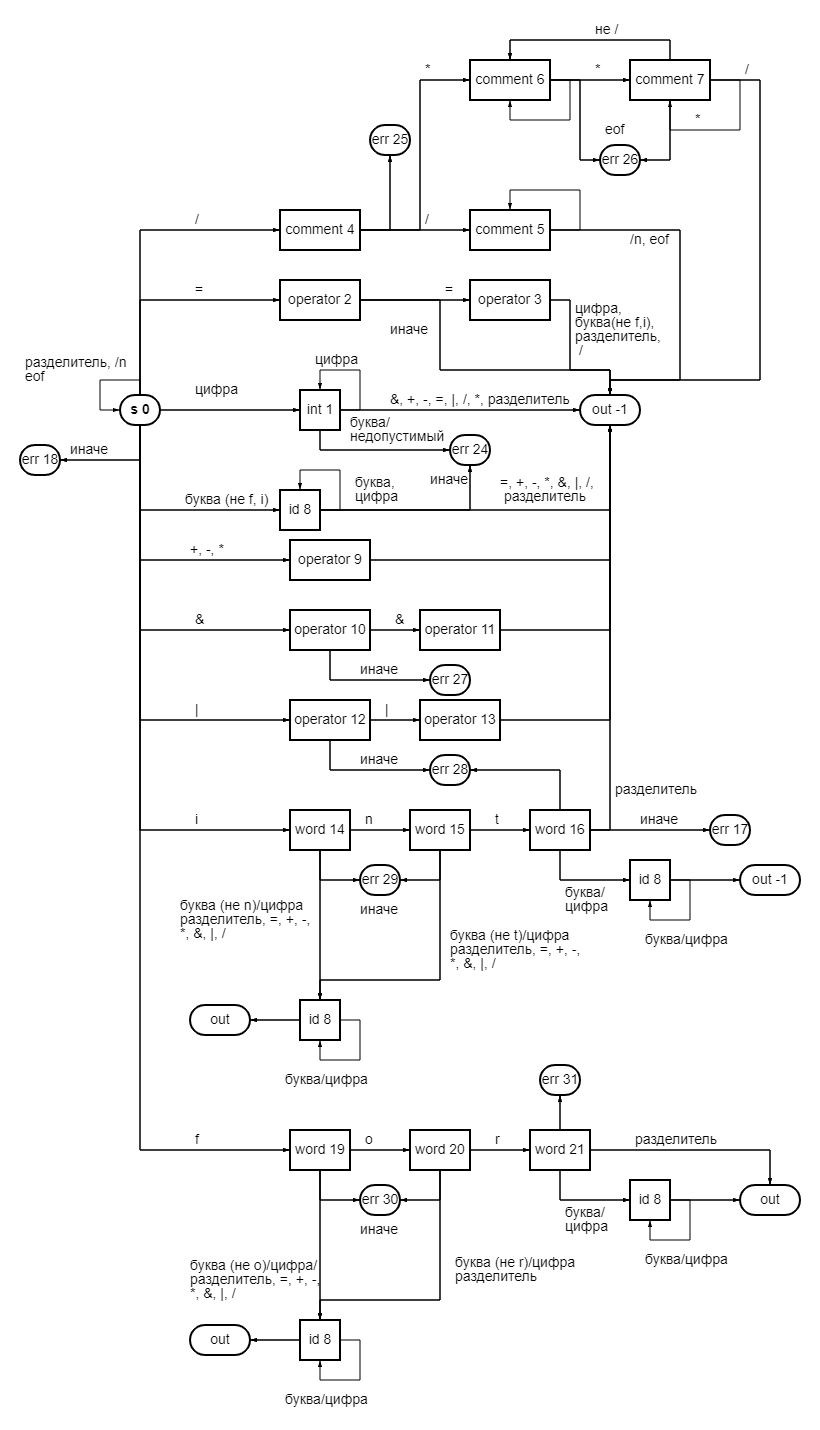
Код для лексического анализатора читается из текстового файла.

1. **Тестовые примеры**

Исходный файл с именем “code.txt”:

|  |  |  |  |
| --- | --- | --- | --- |
| Входные данные | Выходные данные | Пояснение | Комментарий |
| for (int i = 0) | (0, 2, 0)  (6, 5, 0)  (0, 0, 0)  (3, 9, 0)  (1, 0, 0)  (2, 16, 0)  (6, 6, 0) | for  (  int  i  =  0  ) | Проверка распознавания простой структуры |
| int a = 0; | (0, 0, 0)  (3, 1, 0)  (1, 0, 0)  (2, 16, 0)  (6, 2, 0) | int  a  =  0  ; | Проверка распознавания простой структуры |
| for (int i = 0; i = i + 1)  {  k = i;  } | (0, 2, 0)  (6, 5, 0)  (0, 0, 0)  (3, 9, 0)  (1, 0, 0)  (2, 16, 0)  (6, 2, 0)  (3, 9, 0)  (1, 0, 0)  (3, 9, 0)  (1, 3, 0)  (2, 17, 0)  (6, 6, 0)  (6, 3, 0)  (3, 11, 0)  (1, 0, 0)  (3, 9, 0)  (6, 2, 0)  (6, 4, 0) | for  (  int  i  =  0  ;  i  =  i  +  1  )  {  k  =  i  ;  } | Проверка распознавания цикла |
| int k;  //@this is comment int p = 3;  for (;;);  /\* 3l;  this is comment too  int t = 25;  k = k + 1;  \*/  k = k + 1; | (0, 0, 0)  (3, 11, 0)  (6, 2, 0)  (0, 2, 0)  (6, 5, 0)  (6, 2, 0)  (6, 2, 0)  (6, 6, 0)  (6, 2, 0)  (3, 11, 0)  (1, 0, 0)  (3, 11, 0)  (1, 3, 0)  (2, 17, 0)  (6, 2, 0) | int  k  ;  for  (  ;  ;  )  ;  k  =  k  +  1  ; | Проверка работоспособности комментариев |
| /\* 3l;  this is comment too  int t = 25;  k = k + 1; | Error in 52, /\* 3l;  this is comment too  int t = 25;  k = k + 1;  Unclosed comment |  | Проверка ошибки незакрытых комментариев |
| 124hjfjf | Error in 4, h  Unexpected char |  | Проверка недопустимого символа |

1. **Диаграмма состояний**



1. **Тексты программ**

*Tables.h*

#pragma once

#include <string>

#include <vector>

#include <iostream>

#include <fstream>

using namespace std;

struct identifier

{

string name;

int type;

string value;

identifier(string name, int type, string value);

identifier();

};

class const\_table

{

public:

vector<string> table;

const\_table(string filename);// читать из файла

const\_table();

string get(int number);

int search(string value);

};

struct trio

{

int table;

int ind;

int sind;

trio(int t, int i, int si = 0);

trio(int t, pair <int, int> par\_index);

};

/\*---------------------------------\*/

class variable\_table

{

public:

vector<vector<identifier>> table;

variable\_table();

void settype(int type, int hash, int i);

pair <int, int> set(identifier id);

pair <int, int> setname(string name);

void setvalue(string name, string value);

void setvalue(string value, int hash, int i);

pair <int, int> search(string name);

pair <int, int> search(identifier id);

int gettype( int hash, int i);

string getvalue(int hash, int i);

int calc\_hash(string name);

void print\_table(string fname);

};

class Tables

{

public:

const\_table words, operations, separators, alphabet, digits;

variable\_table constants, variables;

Tables();

trio find(string c);

};

*Tables.cpp*

#include "Tables.h"

const\_table::const\_table(string filename)

{

ifstream rfile (filename);

if (rfile.is\_open()) {

string s;

while (!rfile.eof()) {

getline(rfile, s);

table.push\_back(s);

}

}

rfile.close();

}

const\_table::const\_table()

{

}

string const\_table::get(int number)

{

return table[number];

}

int const\_table::search(string value)

{

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

{

if (value == table[i])

{

return i;

}

}

return -1;

}

/\*------------------------------------------\*/

variable\_table::variable\_table()

{

table.resize(32);

}

void variable\_table::settype(int type, int hash, int i)

{

table[hash][i].type = type;

}

pair <int, int> variable\_table::set(identifier id)

{

unsigned int hash = calc\_hash(id.name) % table.size();

pair <int, int> index = search(id.name);

if (index.first == -1)

{

table[hash].push\_back(id);

return { hash, table[hash].size() - 1 };

}

return { index.first, index.second };

}

pair <int, int> variable\_table::setname(string name)

{

unsigned int hash = calc\_hash(name) % table.size();

pair <int, int> index = search(name);

identifier id(name, 0, "");

if (index.first == -1)

{

table[hash].push\_back(id);

return { hash, table[hash].size() - 1 };

}

return { index.first, index.second };

}

void variable\_table::setvalue(string name, string value)

{

pair <int, int> index = search(name);

if (index.first != -1)

{

table[index.first][index.second].value=value;

}

}

void variable\_table::setvalue(string value, int hash, int i)

{

table[hash][i].value = value;

}

pair <int, int> variable\_table::search(string name)

{

int hash = calc\_hash(name) % table.size();

if (table[hash].size() == 0)

{

return { -1, 0 };

}

for (int i = 0; i < table[hash].size(); ++i)

{

if (table[hash][i].name == name)

{

return { hash, i };

}

}

return { -1, 0 };

}

pair<int, int> variable\_table::search(identifier id)

{

return search(id.name);

}

int variable\_table::gettype( int hash, int i)

{

return table[hash][i].type;

}

string variable\_table::getvalue( int hash, int i)

{

return table[hash][i].value;

}

int variable\_table::calc\_hash(string name)

{

unsigned int b = 378551;

unsigned int a = 63689;

unsigned int hash = 0;

for (unsigned int i = 0; i < name.length(); i++)

{

hash = hash \* a + name[i];

a = a \* b;

}

return (hash & 0x7FFFFFFF);//убирает знаковый бит

}

void variable\_table::print\_table(string filename)

{

ofstream rfile(filename);

if (rfile.is\_open()) {

while (!rfile.eof()) {

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

{

for (int j = 0; j < table[i].size(); ++j)

{

rfile << "(" << i << ", " << j << "): (" << table[i][j].name << ", " << table[i][j].type << ", " << table[i][j].value << ");\n";

}

}

}

}

rfile.close();

}

/\*------------------------------------------\*/

identifier::identifier(string nam, int typ, string valu)

{

name = nam;

type = typ;

value = valu;

}

identifier::identifier()

{

name = "";

type = 0;

value = "0";

}

/\*------------------------------------------\*/

Tables::Tables()

{

words = const\_table("words.txt");

operations = const\_table("operations.txt");

separators = const\_table("separators.txt");

alphabet = const\_table("alphabet.txt");

digits = const\_table("digits.txt");

variables = variable\_table();

constants = variable\_table();

}

trio Tables::find(string c)

{

int p;

p = separators.search(c);

if (p != -1)

return trio(0, p);

p = words.search(c);

if ( p!= -1)

return trio(1, p);

p = operations.search(c);

if ( p!= -1)

return trio(2, p);

pair <int, int> pa;

pa = variables.search(c);

if (pa.first != -1)

return trio(3, pa);

pa = constants.search(c);

if (pa.first != -1)

return trio(4, pa);

return trio(-1, 0);

}

/\*------------------------------------------\*/

trio::trio(int t, int i, int si)

{

table = t;

ind = i;

sind = si;

}

trio::trio(int t, pair<int, int> par\_index)

{

table = t;

ind = par\_index.first;

sind = par\_index.second;

}

*Dfa.h*

#pragma once

#include "Tables.h"

enum token {

WORD = 0,

OPERATOR = 1,

CONSTANT = 2,

ID = 3,

ERROR = 4,

COMMENT = 5

};

struct state {

int stat;

string str;

char c;

};

class DFA {

public:

DFA(Tables& tab);

state now;

void read\_matr();

vector<vector<int>> matrix; //матрица перехода

Tables tables;

void print\_tokens();

void read\_code(string filename);

int find\_way(string c);

void next\_state(string c);

int curr = 0;

vector<trio> tokens;

};

*dfa.cpp*

#include "dfa.h"

DFA::DFA(Tables& tab)

{

tables = tab;

read\_matr();

}

void DFA::read\_matr()

{

matrix.resize(32);

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

{

matrix[i].resize(18);

}

ifstream rfile("matr.txt");

if (rfile.is\_open()) {

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

for (int j = 0; j < matrix[i].size(); j++)

rfile >> matrix[i][j];

}

rfile.close();

}

void DFA::print\_tokens()

{

ofstream fout("tokens.txt");

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

{

fout << "(" << tokens[i].table << ", " << tokens[i].ind << ", " << tokens[i].sind << ")\n";

}

fout.close();

//tables.constants.print\_table("constant\_table.txt");

//tables.variables.print\_table("variable\_table.txt");

}

void DFA::read\_code(string filename)

{

now.stat = 0;

now.str = "";

now.c = 0;

tokens.clear();

ifstream rfile(filename);

if (rfile.is\_open()) {

char buff;

string s;

while (rfile.get(buff))

{

s.push\_back(buff);

next\_state(s);

if (now.stat == 0 && s != " ") {

int p = tables.separators.search(s);

if ( p!= -1)

tokens.push\_back(trio(1, p));

}

s = "";

}

s = "eof";

next\_state(s);

s = "";

}

rfile.close();

}

int DFA::find\_way(string c)

{

if (c == "=") {

return 0;

}

if (c == "|") {

return 1;

}

if (c == "&") {

return 2;

}

if (c == "+" || c == "-") {

return 3;

}

if (c == "\*" ) {

return 4;

}

if (c == "f") {

return 5;

}

if (c == "i") {

return 6;

}

if (c == "n") {

return 7;

}

if (c == "t") {

return 8;

}

if (c == "o") {

return 9;

}

if (c == "r") {

return 10;

}

if (tables.separators.search(c) != -1) {

return 12;

}

if (tables.alphabet.search(c) != -1) {

return 11;

}

if (c == "/") {

return 13;

}

if (tables.digits.search(c) != -1) {

return 14;

}

if (c == "eof") {

return 15;

}

if (c == "\n" || c == "\t") {

return 16;

}

return 17;

}

void DFA::next\_state(string s)

{

curr++;

char c = s[0];

int way = find\_way(s);

int next\_st = matrix[now.stat][way];

now.c = c;

if (next\_st == 0)

{

now.str = "";

return;

}

if (next\_st == 26 || next\_st == 25)

{

cerr << "Error in " << curr << ", "<< now.str <<"\nUnclosed comment "<< endl;

}

if (next\_st == 22 || next\_st == 24 || next\_st == 27 || next\_st == 28 || next\_st == 29 || next\_st == 30 || next\_st == 31)

{

cerr << "Error in " << curr << ", " << now.c << "\nUnexpected char" << endl;

}

if (next\_st != -1)

{

now.str.push\_back(c);

now.stat = next\_st;

now.c = 0;

}

if (next\_st == -1)

{

switch (now.stat)

{

default:

break;

case (2):

case (3):

case (9):

case (10):

case (11):

case (12):

case (13):

{

int p = tables.operations.search(now.str);

tokens.push\_back(trio(1,p));

now.str = "";

now.stat = 0;

break;

}

case (8):

case(14):

case(15):

case(19):

case(20):

{

identifier id = identifier(now.str, 0, "");

pair <int, int> p = tables.variables.set(id);

tokens.push\_back(trio(3, p));

now.str = "";

now.stat = 0;

break;

}

case (16):

case (21):

{

int p = tables.words.search(now.str);

tokens.push\_back(trio(0, p));

now.str = "";

now.stat = 0;

break;

}

case(1):

{

identifier id = identifier(now.str, 0, "");

pair <int, int> p = tables.constants.set(id);

tokens.push\_back(trio(2, p));

now.str = "";

now.stat = 0;

break;

}

case (5):

case (7): {

now.str = "";

now.stat = 0;

break;

}

}

}

}

*Lab1.cpp*

#include "Tables.h"

#include "dfa.h"

int main()

{

Tables tables = Tables();

DFA dfa(tables);

dfa.read\_matr();

dfa.read\_code("code.txt");

dfa.print\_tokens();

}