Министерство образования и науки РФ

Новосибирский государственный технический университет

Кафедра прикладной математики

Лабораторная работа №2

по языкам программирования и методам трансляции

Факультет: ПМИ

Группа: ПМ-01

Студент: Конев А.М., Ряховский М.И.

Вариант: 6

Преподаватель: Еланцева И.Л., Полетаева И.А.

Новосибирск

2013

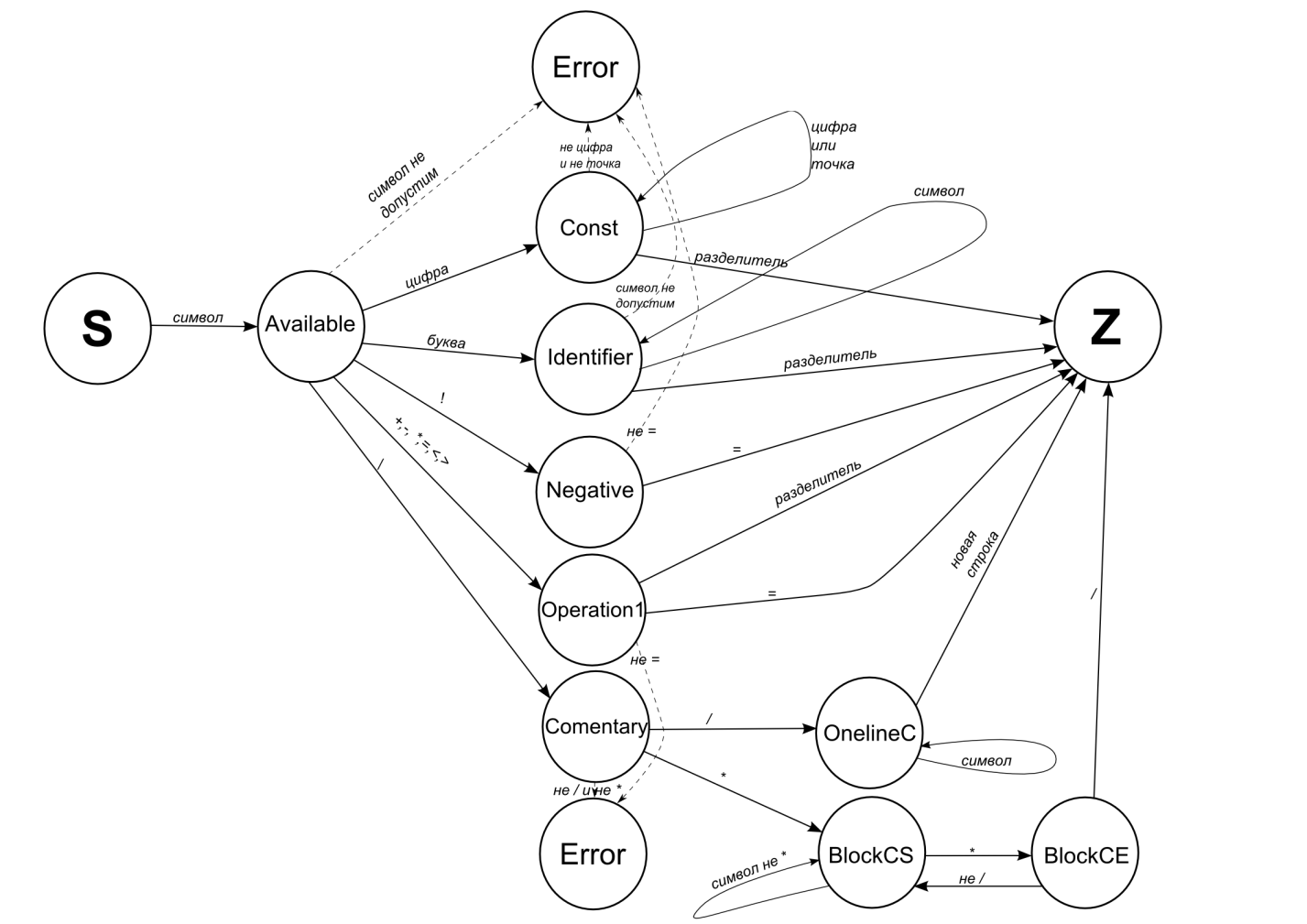
# Цель работы

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

# Входные и выходные данные

Входные данные представляют собой имя файла, содержавшего исходный код, а так же имена файла токенов и файла ошибок. Результатам работы программы являются два файла – файл токенов и файл ошибок.

# Детерминированный конечный автомат



*Примечание:* два поля для ошибок было добавлено, чтобы уменьшить число пересечений рёбер автомата. Пунктирные стрелки ведут к состоянию Error.

*Описание состояний*:

**S** – начальное состояние;

**Available** – проверка символа на допустимость;

**Error** – ошибка;

**Const** – цепочка – константа;

**Identifier** – цепочка – идентификатор или ключевое слов;

**Negative** – операция отрицания;

**Operation1** – одно символьная операция;

**Operation2** – двух символьная операция;

**Comentary** – поверка на комментарий;

**OnelineC** – однострочный комментарий;

**BlockCS** – блочный комментарий;

**BlockCE** – проверка на конец блочного комментария;

**Z** – конечное состояние, разбор закончен.

# Алгоритм разбора

Общий принцип выбранного алгоритма следующий: считывается последовательность символов до пробела(строка), затем она обрабатывается посимвольно и разделятся на лексемы. Если встречается недопустимый символ выдаётся ошибка.

Подробный алгоритм:

1. Считать строку str, если нечего считывать – разбор закончен;
2. Проверить первый символ str, если это не допустимый символ – перейти на шаг 9, иначе на шаг 3. Если трока пустая на шаг 1;
3. Если первый символ – цифра, на шаг 4, если буква или ‘\_’ на шаг 5, если +,-,=,<,> на шаг 6, если / на шаг 7, если ! на шаг 8;
4. Проходим символы, если символ не цифра и не точка, то сформировать лексему из пройдённых символов и вывести токен, оставшиеся символы считатем стройкой str, перейти на шаг 2;
5. Проходим символы, пока они допустимы и не являются разделители или знаками операций, как нашли другой символ формируем строку из пройдённых символов и выводим токен, оставшиеся символы считаем строкой str, прейти на шаг 2;
6. Анализируем второй символ если это ‘=’, то формируем токен о двух символной операции, иначе – об одно символьной. Оставшиеся символы считаем строкой str, прейти на шаг 2;
7. Анализируем второй символ, если второй символ ‘\*’, то ищем подстроку “\*/”, если приходится считываем ещё строки, всё что после указанной подстроки считаем строкой str, перейти на шаг 2. Если второй символ - / считываем строки, пока не закончатся строка файла, перейти на шаг 1. Если второй символ отличается от указанных на шаг 9;
8. Анализируем второй символ, если это ‘=’, то формируем токен и выводи, иначе – перейти на шаг 9;
9. Формируем сообщение об ошибке, разбор закончен;

# Тесты

## Тест 1. Лексически верный исходный код.

|  |  |  |
| --- | --- | --- |
| Исходный код | Файл токенов | Файл ошибок |
| void main(){  float a = 1.0;  int b= 2;  char ac= a+b;  //int bc = 2.01;  /\* комментарий  блочный \*/  } | 3 -1 6  3 -1 4  4 -1 1  4 -1 2  4 -1 4  3 -1 2  5 10 0  2 -1 9  6 1 0  4 -1 3  3 -1 3  5 11 0  2 -1 9  6 2 0  4 -1 3  3 -1 0  5 10 1  2 -1 9  5 10 0  2 -1 3  5 11 0  4 -1 3  4 -1 5 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Таблица идентификаторов | | Таблица констант | |
| Номер цепочки | Содержание цепочки | Номер цепочки | Содержание цепочки |
| 10 | [a, ac] | 1 | 1.0 |
| 11 | b | 2 | 2 |

## Тест 2. Присутствие недопустимых символов

|  |  |  |
| --- | --- | --- |
| Исходный код | Файл токенов | Файл ошибок |
| int main(){  char char\_1 = 2@;  retrun 0;  } | 3 -1 3  3 -1 4  4 -1 1  4 -1 2  4 -1 4  3 -1 0  5 12 0  2 -1 9  6 2 0 | В @; обнаруженны недопустимые символы. Первый символ недопустим.  Ошибка в 2@; , недопустимая цепочка |

|  |  |  |  |
| --- | --- | --- | --- |
| Таблица идентификаторов | | Таблица констант | |
| Номер цепочки | Содержание цепочки | Номер цепочки | Содержание цепочки |
| 12 | char\_1 | 3 | 2 |

## Тест 3. Неправильный комментарий

|  |  |  |
| --- | --- | --- |
| Исходный код | Файл токенов | Файл ошибок |
| void main (){  /неправильный комментарий  } | 3 -1 6  3 -1 4  4 -1 1  4 -1 2  4 -1 4 | Ошибка, символ / не предворяет комментарий  Ошибка в /неправильный , недопустимая цепочка |

Таблицы идентификаторов и констант пусты.

# Тест программы

## Файл «translator.h»

#pragma once

#include "variable\_table.h"

//Класс тарнслятора. Итог всех лаб и РГЗ

class translator{

public:

translator();

//Констуркутор по умолчанию. Ввод постоянных таблиц.

bool lex\_analysis(string sorce\_code\_file, string tokens\_file, string errors\_file);

//Фаза лексического анализа

//sorce\_code\_file - имя файла с исходным кодом

//tokens\_file - имя файла токенов(будет генерироватся)

//errors\_file - имя файла для ошибок

//Возврашает true, если фаза успешно завершенна

void parse(string tokens\_file, string errors\_file);

//Фаза синтаксического анализа

//tokens\_file - имя файла токенов

//errors\_file - имя файла для ошибок

//Струкутра, описывающая токен

class token{

public:

int table\_n; //номер таблицы

int chain\_n; //номер цепочки, если таблица постоянная - -1

int numb; //номер в таблице / в цепочке

token();

token(int t\_n, int ch\_n, int n); //конструктор по типу "сразу всё"

friend ostream& operator << (ostream& os, token& out\_t); //вывод токена

friend istream& operator >> (istream& is, token& inp\_t); //ввод токена

};

private:

//Постоянны таблицы

const\_table<char> admis\_chars; //таблица допустимых символов, используемых для имён (за исключением цифр), номер таблицы - 0

const\_table<char> numbs; //таблица цифр, номер таблицы - 1

const\_table<char> opers\_chars; //таблица символов операций, номер таблицы - 7

const\_table<string> opers; //таблица операций, номер таблицы - 2

const\_table<string> key\_words; //таблица ключевых слов, номер таблицы - 3

const\_table<char> separaters; //таблица разделителей, номер таблицы - 4

//Переменные таблицы

variable\_table identifier; //таблица индетификаторов, номер таблицы - 5

variable\_table consts; //таблица констант, номер таблицы - 6

bool lex\_string\_process(string inp\_str); //Лексическая обработка строки. Возврашает true, если ошибок нет, иначе - false

int check\_symbol(char sym); //проверят допустим ли символ и возврашает:

// -1 - не допустим

// Если допустим, то вернёт номер таблицы

ifstream lex\_sorce\_s; //Лексический анализ. Поток для работы с файлом исходного кода.

ofstream token\_s; //Лексический анализ. Поток для работы с файлом токенов.

ofstream errors\_s; //Лексический анализ. Поток для работы с файлом ошибок.

};

## Файл «translator.cpp»

#include "translator.h"

translator::translator(){

admis\_chars.generate\_from\_file("table\_chars.txt");

numbs.generate\_from\_file("table\_numbs.txt");

opers\_chars.generate\_from\_file("table\_opers\_chars.txt");

opers.generate\_from\_file("table\_opers.txt");

key\_words.generate\_from\_file("table\_key\_words.txt");

separaters.generate\_from\_file("table\_seps.txt");

}

translator::token::token(){

}

translator::token::token(int t\_n, int ch\_n, int n){

table\_n = t\_n;

chain\_n = ch\_n;

numb = n;

}

ostream& operator << (ostream& os, translator::token& out\_t){

os << out\_t.table\_n << " " << out\_t.chain\_n << " " << out\_t.numb << endl;

return os;

}

istream& operator >> (istream& is, translator::token& inp\_t){

is >> inp\_t.table\_n >> inp\_t.chain\_n >> inp\_t.numb;

return is;

}

bool translator::lex\_analysis(string sorce\_code\_file, string tokens\_file, string errors\_file){

lex\_sorce\_s.open(sorce\_code\_file.c\_str(), ios::in); //поток для работы с файлом исходного кода

token\_s.open(tokens\_file.c\_str(), ios::out); //поток для работы с файлом токенов

errors\_s.open(errors\_file.c\_str(), ios::out); //поток для работы с файлом ошибок

string inp\_str; //введёная строка

bool has\_error = false;

while(!lex\_sorce\_s.eof() && !has\_error){

lex\_sorce\_s >> inp\_str;

has\_error = !lex\_string\_process(inp\_str);

};

lex\_sorce\_s.close();

token\_s.close();

errors\_s.close();

return !has\_error;

}

bool translator::lex\_string\_process(string inp\_str){

if(inp\_str.size() != 0){

bool has\_error = false;

int first\_sym\_type = check\_symbol(inp\_str[0]);

bool local\_error = false; //ошибка не в первом символе

string continue\_str;

string basic\_inp = inp\_str;

switch(first\_sym\_type){

case -1:{ //В случае недопустимого символа

errors\_s << "В " << inp\_str << " обнаруженны недопустимые символы. Первый символ недопустим." << endl;

has\_error = true;

cout << "We have a error!" << endl;

}break;

case 0:{ //Если с первого символа может начинатся индентификатор

int i;

//Проверяем можт ли быть такой индетификатор

for(i = 1; i < inp\_str.size() && !local\_error; i++){

local\_error = !(admis\_chars.is\_element\_in\_table(inp\_str[i]) || numbs.is\_element\_in\_table(inp\_str[i]));

}

int start = i;

if(local\_error)

start--;

continue\_str = inp\_str.substr(start);

inp\_str = inp\_str.substr(0, start);

//Если всё хорошо, то проверим ключивое это имя или нет

//Если да создадим токен и пойдём дальше, если нет - добавим в индетификаторы и пойдём дальше)

if(key\_words.is\_element\_in\_table(inp\_str)){

int n;

key\_words.get\_elemen\_number(inp\_str, n);

token\_s << token(3, -1, n);

}

else{

identifier.add\_ind(inp\_str);

int ch\_n, n;

identifier.ind\_adrress(inp\_str, ch\_n, n);

token\_s << token(5, ch\_n, n);

}

local\_error = !lex\_string\_process(continue\_str);

}break;

case 1:{ //Если первый символ - константа

int i;

//Проверяем константа ли это

for(i = 1; i < inp\_str.size() && !local\_error; i++){

local\_error = !(numbs.is\_element\_in\_table(inp\_str[i]) || inp\_str[i] == '.'); //это либо число, либо точка

}

int start = i;

if(local\_error)

start--;

local\_error = false;

continue\_str = inp\_str.substr(start);

inp\_str = inp\_str.substr(0, start);

int points\_n = inp\_str.find\_last\_of('.') - inp\_str.find\_first\_of('.'); //количество вхождения точек

//У нас может быть только одна точка

if(points\_n != 0)

local\_error = true;

//Если нет ошибок, то обновляем таблицу констант и создаём токен

if(!local\_error){

consts.add\_ind(inp\_str);

int ch\_n, n;

consts.ind\_adrress(inp\_str, ch\_n, n);

token\_s << token(6, ch\_n, n);

local\_error = !lex\_string\_process(continue\_str);

}

}break;

case 7:{ //Если первый символ это символ операции

if(inp\_str[0] != '/'){

int i;

string oper\_s = inp\_str.substr(0, 2);

int oper\_l;

//Пытаемся найти нужную двух символьнуюоперацию

local\_error = !opers.is\_element\_in\_table(oper\_s);

//Теперь - односимвольную

if(local\_error){

oper\_s = inp\_str.substr(0,1);

local\_error = !opers.is\_element\_in\_table(oper\_s);

oper\_l = 1;

}

else{

if(inp\_str.size() == 1)

oper\_l = 1;

else

oper\_l = 2;

}

//Если всё хорошо - создаём токен

if(!local\_error){

int n;

opers.get\_elemen\_number(oper\_s, n);

token\_s << token(2, -1, n);

continue\_str = inp\_str.substr(oper\_l, inp\_str.size()-oper\_l);

local\_error = !lex\_string\_process(continue\_str);

}

}

else{ //Если есть подозрение на комментарий

if(inp\_str.size() > 1){

switch(inp\_str[1]){

case '\*':{

string garb; //сюда будем лишние строки

bool comment\_end = false;

while(!comment\_end && !lex\_sorce\_s.eof()){

lex\_sorce\_s >> garb;

int st = garb.find("\*/");

if(st != -1)

comment\_end = true;

}

if(!lex\_sorce\_s.eof()){

int st1 = garb.find\_first\_of("\*/");

continue\_str = garb.substr(st1+2);

local\_error = !lex\_string\_process(continue\_str);

}

else{

local\_error = true;

errors\_s << "Незакрытый коментарий" << endl;

}

}break;

case '/':{

string garb;

getline(lex\_sorce\_s, garb);

}break;

default:{

errors\_s << "Ошибка, символ / не предворяет комментарий" << endl;

local\_error = true;

}break;

};

}

else{

errors\_s << "Ошибка, символ / не предворяет комментарий" << endl;

local\_error = true;

}

}

}break;

case 4 :{ // Если разделитель

continue\_str = inp\_str.substr(1);

int n;

separaters.get\_elemen\_number(inp\_str[0],n);

token\_s << token(4, -1, n);

local\_error = !lex\_string\_process(continue\_str);

}break;

default: {

errors\_s << "Неполучилось определить принадлежность первого символа" << endl;

has\_error = true;

cout << "Обноруженна ошибка" << endl;

}break;

};

if(local\_error){

cout << "We have a error!" << endl;

errors\_s << "Ошибка в " << basic\_inp << " , недопустимая цепочка" << endl;

}

return !(has\_error || local\_error);

}

else

return true;

}

int translator::check\_symbol(char sym){

if(admis\_chars.is\_element\_in\_table(sym))

return 0;

if(numbs.is\_element\_in\_table(sym) || sym == '-')

return 1;

if(opers\_chars.is\_element\_in\_table(sym))

return 7;

if(separaters.is\_element\_in\_table(sym))

return 4;

return -1;

}