2.2. Разработка лексического анализатора

Во входном тексте программы возможно использование следующих элементов:

1. Идентификаторы или имена переменных
2. Числа
3. Знаки сравнения
4. Разделители
5. Скобки
6. Ключевые слова

Для выделения этих элементов, проверку их на корректность необходимо разработать структуру лексического анализатора, выполнить программную реализацию и провести тестирование.

2.2.1. Разработка структуры лексического анализатора

В качестве структуры лексического анализатора была выбрана диаграмма переходов состояний, так как при помощи ее состояний можно представлять различные ситуации, которые могут возникнуть в процессе сканирования входного текста.

Если в начале разбора нового входного слова встречается цифра, причем она может быть представлена в 16-чном формате, то происходит запись символа в буфер и переход в состояние «число». В этом состоянии происходит дальнейшее сканирование текста и проверка новых символ на соответствие цифре. В случае непрохождения проверки, происходит переход в состояние «ошибка». Проверка и запись в буфер продолжается до тех пор, пока не встретиться разделитель. После происходит создание токена, очистка буфера и переход в начальное состояние. Диаграмма переходов состояний для распознавания числа представлена на рисунке 2.

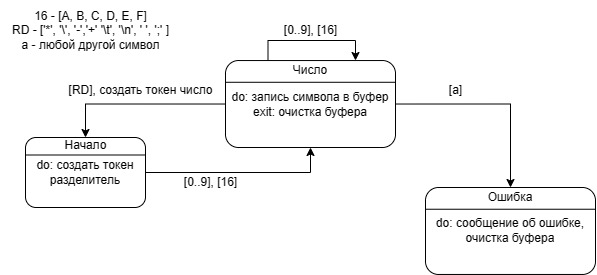


Рисунок 2 – Диаграмма переходов состояний для распознавания числа

Правильной числовой последовательностью можно считать следующие примеры:

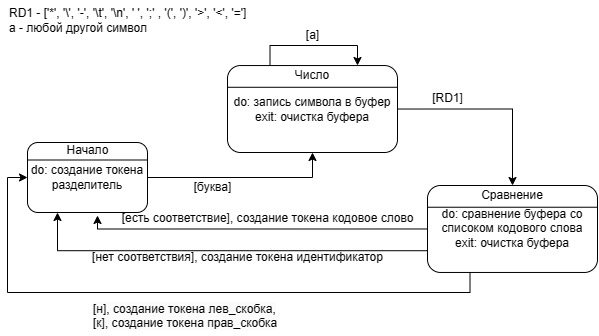
- 856

- AFF

Неправильным примером числовой последовательности является:

- 8ва24

Если при распознавании нового слова встретилась буква, то происходит запись символа в буфер и переход в состояние «слово» с последующим сканированием текста программы и запись в буфер до момента обнаружения символа – разделителя. Далее полученный буфер сравнивается со списком ключевых слов, если находится соответствие, то создается токен с указанием «ключевое слово» в качестве типа или «лев\_скобка» и «прав\_скобка» для слов «н» и «к» соответственно. Если же соответствие не найдено, то создается токен с типом идентификатор. Очистка буфера и переход в начальное состояние. Диаграмма переходов состояний представлена на рисунке 3.



Примеры определения типов следующие :

- определить – ключевое слово

- н – лев\_скобка

- опред - идентификатор

При обнаружении разделителя происходит создание токена с указанием его типа в качестве типа.

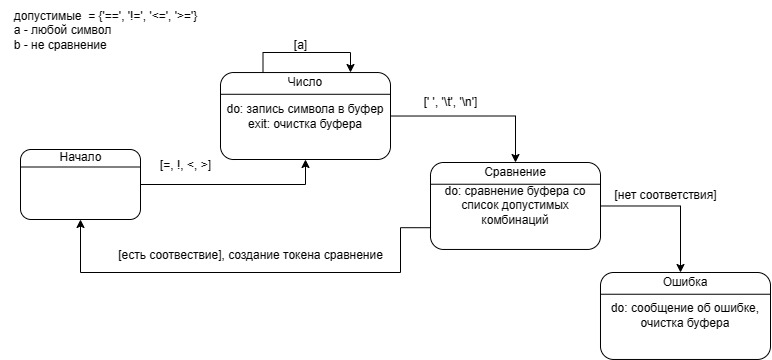
Примеры символов и их типы могут быть следующие:

- “ - ” - минус

- “ \* ” - звезда

В случае получения символов “ ( ” или “ ) ” создаются токены с типом прав\_скобка и лев\_скобка соответственно.

Если при сканировании встречается символ “=” или “!”, то происходит запись в буфер и переход в состояние «разделитель». В нем происходит запись в буфер до момента, пока не входной символ не будет являться числом, или буквой, или пробелом. В этом случае происходит анализ содержимого буфера, если там хранится “=”, то создается токен с типом присваивания. Если же содержимым является “ != ” или “==”, то токен имеет тип сравнение. В любом другом случае происходит переход в состояние «ошибка». Диаграмма переходов состояний представлена на рисунке 4.



Неправильными вариантами последовательности могут служить следующие примеры:

- !в

- ===

В состоянии ошибка бросается исключение, в котором указывается лексема, в которой обнаружена ошибка и № строки, в которой она допущена.

Общая диаграмма переходов состояний представлена на рисунке 5.

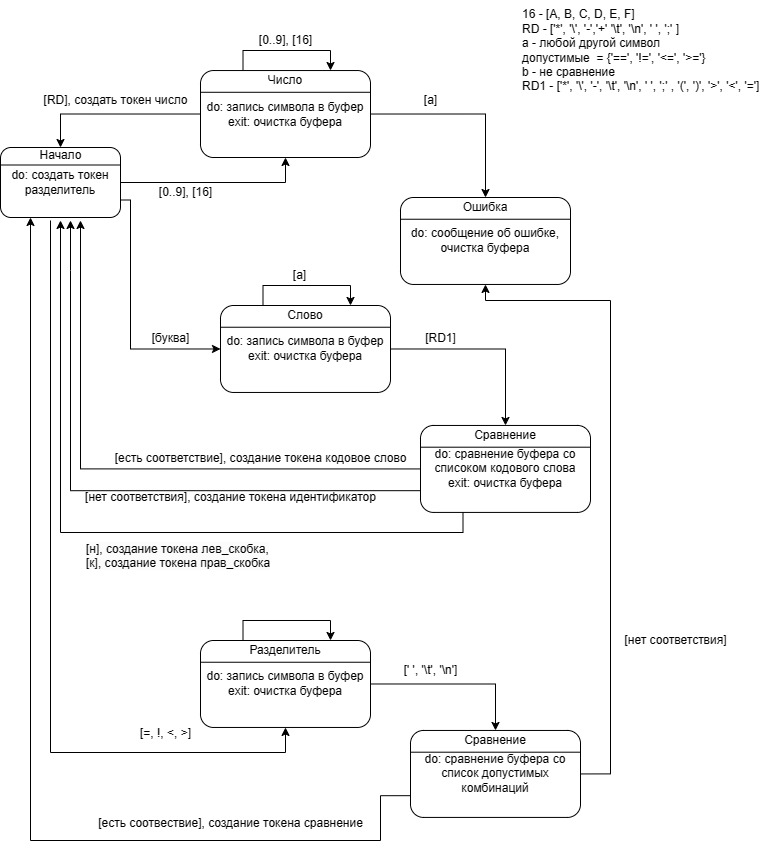


Рисунок 5 – Диаграмма переходов общая

2.2.2. Программная реализация

В ходе выполнения программной реализации было произведено кодирование автомата для распознавания лексем, заданного диаграммой переходов состояний.

В качестве состояний автомата был создан класс перечислений States, в котором указаны названия состояний. Данный класс представляет собой ограниченный список идентификаторов, что позволяет выполнять быстрые переходы между ними. Программный код класса States представлен ниже:

enum class States // состояния автомата

{

S, //начальное

NUM, // число

DLM, //разделитель

WORD, //слово

OPRTN, //операция

ER, // ошибка

FIN, //конечное

};

Такой же класс был разработан для определения типа токена. Его программный код представлен ниже:

enum class token\_type

{

ID, //переменная

NUMBER, //число

KEY\_WORD, //ключевое слово

DLM, // , ;

ASSGN, // =

COM, // != == > <

LPAR, // ( н

RPAR, // ) к

PAR, // "

MINUS, // -

PLUS,

METKA, // <name>:

OR,

AND,

NOT,

STAR,

SHIFT, // << >>

DIV,

};

Для определения является ли символ разделителем был создан массив, в котором описаны все разделители, которые могут встретиться в тексте программы, а также функция, проверяющая наличие символа в массиве разделителей. Их программная реализация представлена ниже:

char delimetr[] { '(', ')', ';', ' ', '-', '\n', '=', ':', '>', '<', '\t', '/', ',',

'!', '+', '\*'};

bool isDelimetr(char symbol)

{

bool exists = find(begin(delimetr), end(delimetr), symbol) != end(delimetr);

if (exists) return true;

else return false;

}

Программная реализация конечного автомата для распознавания числа следующая:

для каждого символа строки:

{

выбор (состояние):

{

состояние:: начальное:

{

если (символ – число)

{

запись в буфер;

состояние: = число;

}

}

состояние:: число:

{

если (символ – число)

{

запись в буфер;

состояние: = число;

}

иначе

{

если (символ

}

}

}

}

for (char symbol : code) //по символу строки

{

if (flag == true)

{

switch (\_state)

{

case States::S: //+

{

if (isDigit(symbol))

{

temp\_token += symbol;

\_state = States::NUM;

}

break;

}

case States::NUM: //+

{

if (isDigit(symbol))

{

temp\_token += symbol;

\_state = States::NUM;

}

else

{

if (isDelimetr(symbol))

{

if (isInvalidDelimetr(symbol))

{

\_state = States::ER;

}

else

{

token\* new\_token = new

token(temp\_token, token\_type::NUMBER);

\_tokens.push\_back(new\_token);

temp\_token.clear();

\_state = States::S;

}

}

else

{

temp\_token += symbol;

\_state = States::ER; }

}

break;

}

case States::ER:

{

string message = "ошибка в строке № " +

to\_string(number\_string) + " " + temp\_token + '\n';

throw Exception{ message };

temp\_token.clear();

\_state = States::FIN;

break;

}

case States::FIN:

{

cout << "End\n";

flag = false;

break;

}

default:

break;

}

}

}

}

Полностью листинг программы лексического анализатора представлен в приложении А.

2.2.3. Тестирование

Тестирование по методу черного ящика представлено в таблице 2.

Таблица 2 – Тестирование методом черного ящика

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Операция | Тест | Ожидаемый рез-т | Фактический рез-т | Рез-т проверки |
| Ввод 10 числа | 98 | Type: NUMBER | Type: NUMBER | Успешно |
| 8к | Ошибка 8к | Ошибка 8к | Успешно |
| Ввод 16 числа | 08FF | Type: NUMBER | Type: NUMBER | Успешно |
| 08ff | Ошибка 08f | Ошибка 08f | Успешно |
| 08АА | Ошибка 08А | Ошибка 08А | Успешно |
| Ввод имени | С\_рег | Type: ID | Type: ID | Успешно |
| 2имя | Ошибка 2и | Ошибка 2и | Успешно |
| Ввод унарной операции | 8 – 7 | Type: MINUS | Type: MINUS | Успешно |
| 8 % 7 | Ошибка % | Ошибка % | Успешно |
| Ввод двойной операции | 8 << 7 | Type: SHIFT | Type: SHIFT | Успешно |
| 8 <+ 7 | Ошибка <+ | Ошибка <+ | Успешно |
| Ввод скобок | ( | Type: LPAR | Type: LPAR | Успешно |
| { | Ошибка { | Ошибка { | Успешно |
| Ввод ключевых слов | Определить | Type: KEY\_WORD | Type: KEY\_WORD | Успешно |
| Опрделить | Type: ID | Type: ID | Успешно |