**

*Институт транспорта и связи*

Лабораторная работа №1

По дисциплине «Проектирование компиляторов»

Тема: **Программирование лексического анализатора**

Студент: Anastasija Mežale

Группа: 4802BD

Рига

2021 г.

**Содержание**

[**Задание** 2](#_Toc84958037)

[**Теоретическое обоснование** 3](#_Toc84958038)

[**Индивидуальный вариант** 4](#_Toc84958039)

[**Таблицы лексем** 5](#_Toc84958040)

[**Грамматические правила** 6](#_Toc84958041)

[**Диаграмма состояний конечного автомата** 7](#_Toc84958042)

[**Код программы** 8](#_Toc84958043)

[**Результаты работы программы** 14](#_Toc84958044)

[**Выводы** 17](#_Toc84958045)

### **Задание**

1. Произвести анализ заданного на языке PASCAL программного фрагмента (индивидуальные варианты фрагментов – номер индивидуального задания выбирается в соответствии со своим номером по списку групп) и выделить все типы имеющихся в нем лексем: ключевые слова, специальные символы (однопозиционные, двухпозиционные разделители), литералы, идентификаторы.
2. Для выделенных типов лексем построить регулярную грамматику. Если в заданном программном фрагменте есть комментарии и текстовые (символьные, строковые) литералы, то грамматические правила для них строить не нужно, однако их распознавание должно осуществляться далее конечным автоматом.
3. Для полученной грамматики построить диаграмму состояний соответствующего конечного автомата, который сможет распознавать все лексемы выделенных типов для заданного программного фрагмента.
4. На основе полученного в пунктах 1-3 написать программу лексического анализатора (сканер), который:

* Принимает заданный программный фрагмент, таблицу ключевых слов и таблицу специальных символов (эти две таблицы могут быть объединены в одну таблицу т.н. служебных символов, которые могут задаваться в тексте программы сканера, например, в виде массива или хранится в файле и фиксироваться как-то иначе);
* Возможно, формирует отдельную таблицу распознанных литералов и таблицу распознанных идентификаторов;
* Формирует таблицу т.н. стандартных символов (все распознанные лексемы в порядке их появления в заданном программном фрагменте и их тип, а также, возможно, индекс в таблице соответствующего типа лексем, если таковые формировались);
* Сообщать об успешном или неуспешном результате лексического анализа (для этого необходимо внести изменения в исходный фрагмент и проверить, что сканер по-прежнему распознает все лексемы, если лексических ошибок нет, и обнаруживает лексические ошибки, если они есть).

1. Подготовить отчет о проделанной работе.

### **Теоретическое обоснование**

**Лексический анализ** – это процесс аналитического разбора последовательности символов на распознанные лексемы (токены). Лексический анализ используется в компиляторах. Грамматика языка задает набор лексем, которые могут встретиться в ходе лексического анализа. Программа, выполняющая лексический анализ, называется лексическим анализатором или лексическим сканером.

**Лексема** – это последовательность допустимых символов языка программирования, обрабатываемая компилятором. Лексемами являются идентификаторы, ключевые слова, разделители, литералы (числовые и строковые).

### **Индивидуальный вариант**

На основании программного фрагмента на языке Pascal, представленном в соответствии с индивидуальным вариантом №11 (рис. 1), необходимо описать таблицы лексем, грамматические правила и отобразить диаграмму состояний конечного автомата, после чего разработать сам лексический анализатор.

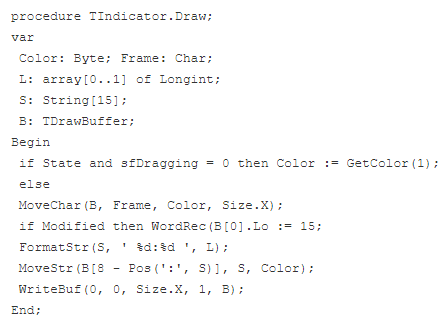


Рис. 1. Программный фрагмент индивидуального варианта

### **Таблицы лексем**

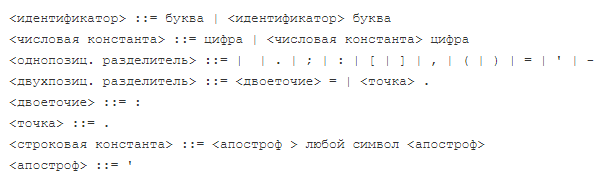
В ходе анализа программного фрагмента были составлены таблицы лексем (таблица 1), содержащие ключевые слова, идентификаторы, специальные символы, числовые константы и строковые константы.

Таблица 1. Таблицы лексем программного фрагмента

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ключевые слова | Идентификаторы | Специальные символы (разделители) | Числовые константы | Строковые константы |
| procedure | TIndicator | . | 0 | ‘%d:%d’ |
| var | Draw | ; | 1 | ‘:’ |
| Byte | Color | : | 15 |  |
| Char | Frame | [ | 8 |  |
| array | L | .. |  |  |
| of | S | ] |  |  |
| Longint | B | = |  |  |
| String | TDrawBuffer | := |  |  |
| Begin | State | ( |  |  |
| if | sfDragging | ) |  |  |
| and | GetColor | , |  |  |
| then | MoveChar | ‘ |  |  |
| else | Size | - |  |  |
| End | X |  |  |  |
|  | Modified |  |  |  |
|  | WordRec |  |  |  |
|  | Lo |  |  |  |
|  | FormatStr |  |  |  |
|  | MoveStr |  |  |  |
|  | Pos |  |  |  |
|  | WriteBuf |  |  |  |

### **Грамматические правила**

В соответствии с составленными таблицами лексем были выявлены грамматические правила (рис. 2), необходимые для реализации конечного автомата.

Рис. 2. Грамматические правила программного фрагмента

### **Диаграмма состояний конечного автомата**

На основании грамматических правил была разработана следующая диаграмма состояний конечного автомата:

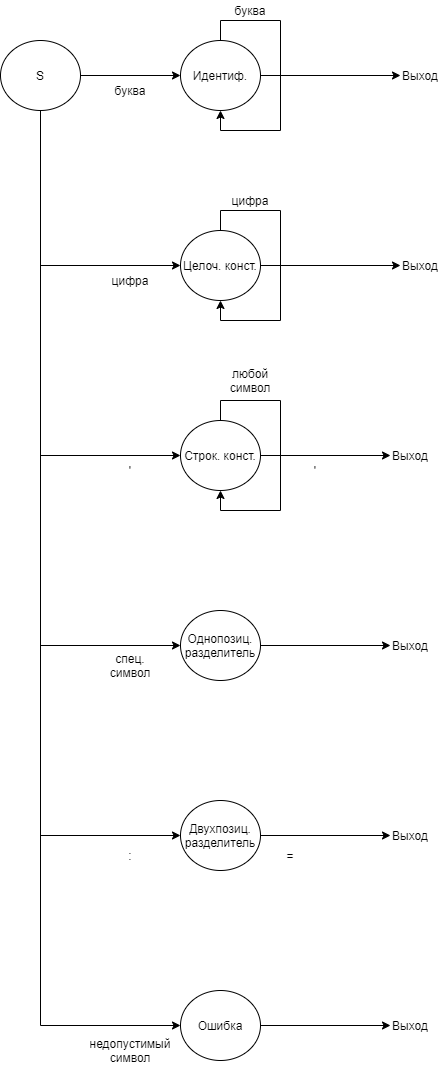


Рис. 3. Диаграмма состояний конечного автомата программного фрагмента

### **Код программы**

#include <iostream>

#include <fstream>

#include <string>

#include <vector>

#include <algorithm>

using namespace std;

bool isID(const string &str);

bool isNumericConstant(const string &str);

bool isStringConstant(const string &str);

bool isKeyword(const string &str);

bool isDelimeter(const string &str);

bool isSpace(const string &str);

void role(const string &token);

void print();

void scan(const string &file);

const vector<string> keywords{ "procedure", "var", "byte", "char", "array", "of", "longint", "string", "begin", "if", "and", "then", "else", "end" };

const vector<string> delimeters{ ".", ";", ":", "[", "]", ",", "(", ")", "=", ":=", "'", "-", "..", "" };

const vector<string> numericConstants{ "0", "1", "15", "8" };

const vector<string> stringConstants{ "'%d:%d'", "':'" };

vector<string> findedID;

vector<string> findedKeywords;

vector<string> findedDelimeters;

vector<string> findedNumericConstants;

vector<string> findedStringConstants;

bool ex = false;

void main()

{

cout << "Scan:\n";

scan("code.txt");

system("PAUSE");

}

bool isID(const string &str)

{

try

{

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

if (!(isalpha(str[i]))) //идентификатор может содержать только буквы

throw (str);

}

catch (const string)

{

if (!ex)

{

ex = true;

cout << "\nError! An identifier can only contain a letters!" << endl;

}

return false;

}

return true;

}

bool isNumericConstant(const string &str)

{

bool flag = false; //переменная для переключения между состояниями

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

{

if (!(isdigit(str[i]))) //число должно содержать только цифры

{

flag = true;

return false;

}

}

if (!flag)

for (auto numericConstant : numericConstants)

if (numericConstant == str)

{

flag = true;

return true;

}

try

{

if (!flag)

throw (str);

}

catch (const string)

{

ex = true;

cout << "\nError! Number is not a registered numeric constant!" << endl;

return false;

}

}

bool isStringConstant(const string &str)

{

bool flag = false;

for (auto stringConstant : stringConstants)

if (stringConstant == str)

{

flag = true;

return true;

}

try

{

if ((!flag) && (str[0] == char(39) && (str[str.size() - 1]) == char(39))) //char(39) - знак апострофа (') по таблице ASCII

throw (str);

}

catch (const string)

{

ex = true;

cout << "\nError! String is not a registered string constant!" << endl;

return false;

}

return false;

}

bool isKeyword(const string &str)

{

string buffer = str;

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

buffer[i] = tolower(str[i]); //буквы могут быть любого регистра

for (auto keyword : keywords)

if (keyword == buffer)

return true;

return false;

}

bool isDelimeter(const string &str)

{

bool flag = false;

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

{

if (!(ispunct(str[i]))) //разделитель может содержать только знаки пунктуации

{

flag = true;

return false;

}

}

if ((!flag) && (str.size() != 1 && str[0] == char(39) && (str[str.size() - 1]) == char(39))) //строковая константа, содержащая только разделитель, не должна обрабатываться

{

flag = true;

return false;

}

if (!flag)

for (auto delimeter : delimeters)

if (delimeter == str)

{

flag = true;

return true;

}

try

{

if (!flag)

throw (str);

}

catch (const string)

{

ex = true;

cout << "\nError! Delimeter is not a registered delimeter!" << endl;

return false;

}

}

bool isSpace(const string &str)

{

if (str == " " || str == "\n")

return true;

return false;

}

void role(const string& str)

{

bool flag = false; //переменная для нахождения совпадений при переборе векторов

string bufferStr = str; //переменная для проверки регистра подаваемой строки

string bufferID; //переменная для проверки регистра хранящихся идентификаторов

if (isKeyword(str))

{

for (auto keyword : findedKeywords)

if (keyword == str)

flag = true;

if (!flag)

findedKeywords.push\_back(str); //если совпадение не было найдено, лексема записывается в вектор и флаг опускается

flag = false;

}

else if (isDelimeter(str))

{

for (auto delimeter : findedDelimeters)

if (delimeter == str)

flag = true;

if (!flag)

findedDelimeters.push\_back(str);

flag = false;

}

else if (isNumericConstant(str))

{

for (auto numericConstant : findedNumericConstants)

if (numericConstant == str)

flag = true;

if (!flag)

findedNumericConstants.push\_back(str);

flag = false;

}

else if (isStringConstant(str))

{

for (auto stringConstant : findedStringConstants)

if (stringConstant == str)

flag = true;

if (!flag)

findedStringConstants.push\_back(str);

flag = false;

}

else if (isID(str))

{

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

bufferStr[i] = tolower(str[i]);

for (auto ID : findedID)

{

bufferID = ID;

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

bufferID[i] = tolower(ID[i]);

if (bufferID == bufferStr)

flag = true;

}

if (!flag)

findedID.push\_back(str);

flag = false;

}

else

{

cout << "Invalid token: " << str << endl;

ex = false;

}

}

void print()

{

//вывод таблиц распознанных лексем

cout << "\n==============\nIDENTIFIERS\n==============\n\n";

for (auto ID : findedID)

{

cout << ID << endl;

}

cout << "\n==============\nKEYWORDS\n==============\n\n";

for (auto keyword : findedKeywords)

{

cout << keyword << endl;

}

cout << "\n==============\nDELIMETERS\n==============\n\n";

for (auto delimeter : findedDelimeters)

{

cout << delimeter << endl;

}

cout << "\n==============\nNUMERIC CONSTANTS\n==============\n\n";

for (auto numericConstant : findedNumericConstants)

{

cout << numericConstant << endl;

}

cout << "\n==============\nSTRING CONSTANTS\n==============\n\n";

for (auto stringConstant : findedStringConstants)

{

cout << stringConstant << endl;

}

}

void scan(const string &fileName)

{

char ch; //переменная для хранения символов

string buffer; //переменная для хранения последовательностей символов

string twoposDelim; //переменная для хранения двухпозиционных разделителей

string strConstant; //переменная для хранения строковых констант

bool flag = false; //переменная для установления начала и конца записи строковых констант

fstream file(fileName, fstream::in);

if (!file.is\_open())

{

cout << "Error while opening the file\n";

exit(0);

}

else

{

while (file >> noskipws >> ch) //чтение символов из файла с учетом начальных пробельных символов

{

if (isSpace(string(1, ch))) //1 - размер строки, ch - символ для заполнения размера. В буфер добавляется по одному символу

{

if (!buffer.empty()) //если буфер не пуст, проверяет содержимое буфера на совпадение с существующими лексемами, после чего буфер обнуляется

{

role(buffer); //определение типа лексемы

buffer = ""; //обнуление буфера

}

continue;

}

if (flag) //пока значение flag равно true, идет запись строковой константы в переменную strConstant

{

if (string(1, ch) == "'") //если в ходе записи строковой константы встречается апостроф, запись заканчивается и содержимое переменной strConstant проверяется на совпадение с лексемами

{

strConstant += string(1, ch);

role(strConstant);

flag = false;

buffer = "";

}

strConstant += string(1, ch);

continue;

}

if (!flag && string(1, ch) == "'") //если встречен апостроф, начинается запись строковой константы

{

strConstant = "";

strConstant += string(1, ch);

flag = true;

}

if (isDelimeter(string(1, ch)) || isDelimeter(buffer)) //если входящий символ или содержимое буфера является разделителем

{

if (isDelimeter(string(1, ch)) && !isDelimeter(buffer)) //если входящий символ является разделителем, а содержимое буфера - нет, то буфер проверяется на совпадение с лексемами и обнуляется

{

if (!buffer.empty())

{

role(buffer);

buffer = "";

}

}

else if (!isDelimeter(string(1, ch)) && isDelimeter(buffer))

{

role(buffer);

buffer = "";

}

else if ((string(1, ch) == "." || string(1, ch) == "=") && (buffer == "." || buffer == ":")) //проверка двухпозиционных разделителей

{

twoposDelim = buffer + string(1, ch);

role(twoposDelim);

twoposDelim = "";

}

else

{

role(buffer);

buffer = "";

}

}

buffer += ch; //формирование буфера, посимвольное добавление

}

print(); //вывод таблиц распознанных лексем

}

file.close();

}

### **Результаты работы программы**

Представленный выше код программы лексического анализатора реализован на языке C++ и обладает следующим функционалом:

* Вывод отдельных таблиц различных типов лексем: ключевых слов, идентификаторов, разделителей, числовых констант и строковых констант;
* Вывод лексем в порядке их появления;
* Вывод только уникальных значений;
* Распознавание ключевых слов и идентификаторов вне зависимости от регистра символов;
* Обработка лексических ошибок, таких как наличие чисел и знаков пунктуации в названиях идентификаторов, а также символы, не соответствующие составленной таблице лексем в зависимости от типа (например, если строка заключена в одинарные кавычки, но не зарегистрирована в векторе строковых констант, то лексический анализатор объявит о неизвестной строковой константе).

Ниже представлена работа программы при отсутствии лексических ошибок (исходный программный фрагмент, представленный в индивидуальном задании):

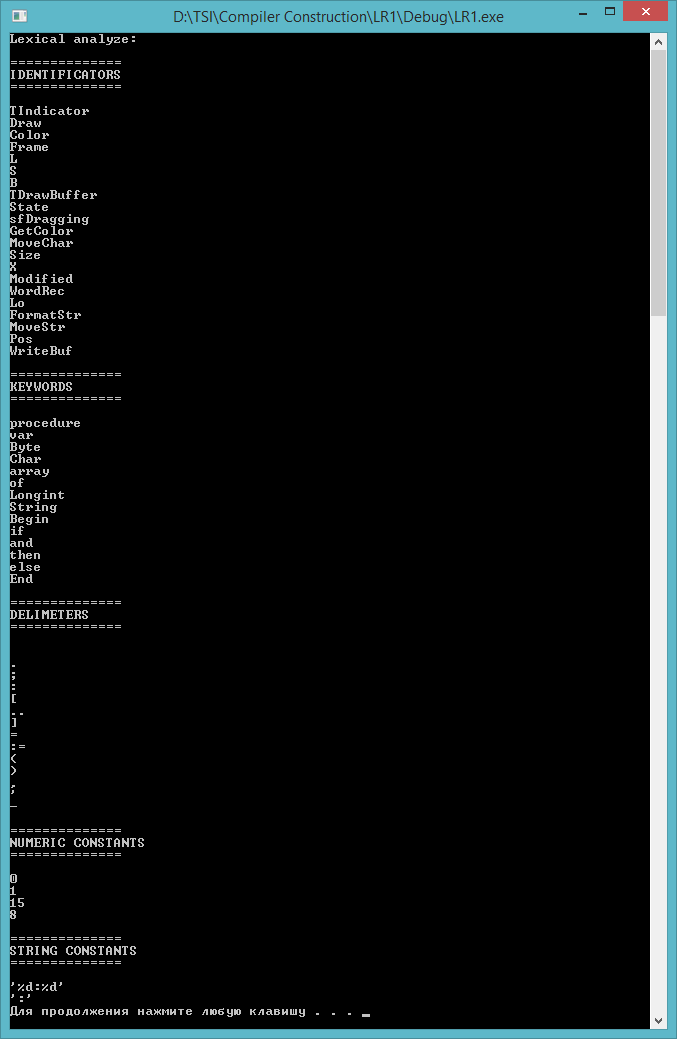


Рис. 4. Результат лексического анализа программного фрагмента, заданного в индивидуальном варианте

Если внести изменения в исходный программный код, например, строковую константу '%d:%d' изменить на '123', программа выдаст исключение (рис. 5 и 6).

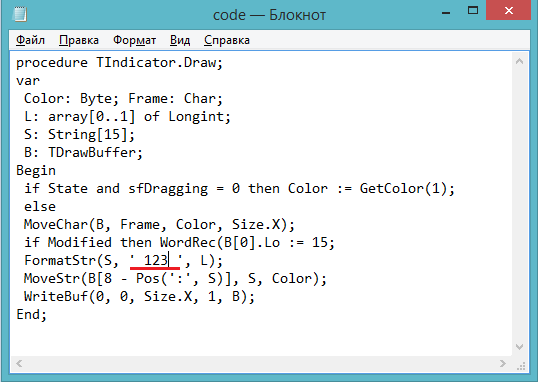


Рис. 5. Измененный программный фрагмент

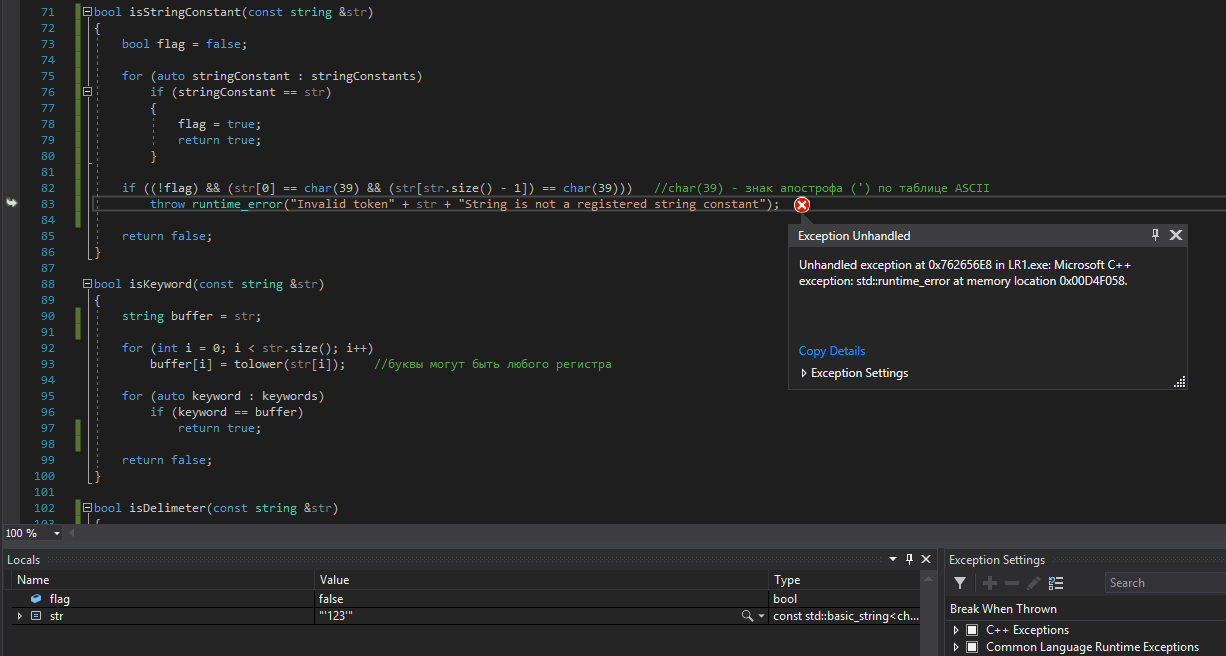


Рис. 6. Обработка лексической ошибки

### **Выводы**

В ходе выполнения лабораторной работы были усвоены теоретические и практические знания по работе лексических анализаторов, написания правил регулярной грамматики и разработке диаграмм состояний конечных автоматов. Для реализации конечного кода был продуман алгоритм посимвольного чтения содержимого файла и его обработки и составлены ограничения в виде исключений на основе грамматических правил фрагмента для обработки лексических ошибок различных типов.

Наибольшую сложность составило разделение ключевых слов и идентификаторов от разделителей для корректного чтения содержимого файла, однако в процессе написания программы данная проблема была решена.

Реализованная программа работает в соответствии с поставленными перед ней требованиями и выполняет поставленную перед ней задачу, проводя лексический анализ программного фрагмента на языке Pascal.