МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования ф

«Ижевский государственный технический университет имени М. Т. Калашникова» (ФГБОУ ВО «ИжГТУ имени М.Т. Калашникова»)

Факультет «Информатика и вычислительная техника»

Кафедра «Автоматизированные системы обработки информации и управления»

ОТЧЁТ

По лабораторной работе №1

по дисциплине «Формальные языки и трансляторы»

на тему: «Разработка лексического анализатора.»

Выполнил

Студент группы Б19-782-2 И.С. Пономарёв

Руководитель

К.т.н. доцент кафедры «АСОИУ» Д. Р. Касимов

г. Ижевск 2022

# Постановка задачи

Задание по лабораторной работе заключается в разработке лексического анализатора для предложенного варианта слов. На вход подается многострочный текст, содержащий два вида слов, следующих в произвольном порядке. Слова одного вида должны быть разделены любым количеством пробелов. Слова разного вида могут записываться слитно или разделяться любым количеством пробелов. Текст может содержать комментарии. Комментарии и пробелы должны пропускаться. Результатом работы программы является диагностическое сообщение о первом неверном слове (если такое встретится), сообщение о правильности поступившего текста.

Первое слово - (111)\*011(100)\*.

Второе слово - (a|b|c|d)+ .

Комментарий - (\* многостр. комментарий \*) (как в Delphi).

Условие для второго слова - Нет подстроки ab.

# Описание регулярной грамматики

В таблице 2.1 показаны правила регулярной грамматики для первого, второго слова и комментария

|  |  |  |
| --- | --- | --- |
| Первое слово | Второе слово | Комментарий |
| S→1A | 0C  A→1B  B→ 1S  C→1D  D→1E | 1  E→1F  F→0J  J→0 | 0E | S→ aA | bB|cB | dB | a | b | c | d  A → aA | cB | dB | a | c | d  B → aA | bB | cB | dB | a | b | c | d | S → (A  A → \*B  B→ ( любой символ кроме \*)B |\*C  C→ ) | любой символ кроме \* B | \*C |

Таблица 2.1 – Правила регулярной грамматики

# Недетерминированные конечные автоматы

На рисунке 3.1 показана диаграмма НКА для первого слова.

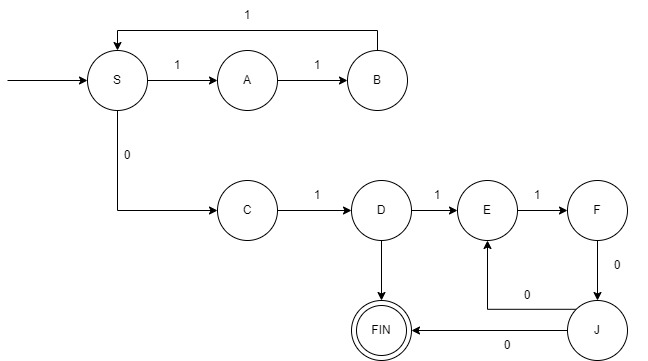


Рисунок 3.1 – Диаграмма НКА для первого слова

В таблице 3.1 представлена матрица НКА для первого слова

|  |  |  |  |
| --- | --- | --- | --- |
|  | 1 | 0 | Явл. фин? |
| S | A | C | Нет |
| A | B |  | Нет |
| B | S |  | Нет |
| C | D |  | Нет |
| D | E,FIN |  | Нет |
| E | F |  | Нет |
| F |  | J | Нет |
| J | E,FIN |  | Нет |
| FIN |  |  | Да |

Таблица 3.1 – Матрица НКА для первого слова

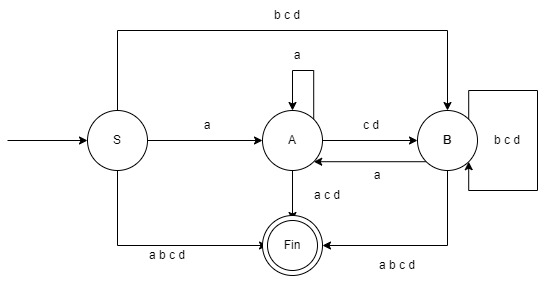
На рисунке 3.2 показана диаграмма НКА для второго слова.

Рисунок 3.2 – Диаграмма НКА для второго слова

В таблице 3.2 показана матрица НКА для второго слова.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | a | b | c | d | Явл. Фин? |
| S | A,Fin | B,Fin | B,Fin | B,Fin | Нет |
| A | A,Fin |  | B,Fin | B,Fin | Нет |
| B | A,Fin | B,Fin | B,Fin | B,Fin | Нет |
| Fin |  |  |  |  | Да |

Таблица 3.2 – Матрица НКА для второго слова

# Детерминированные конечные автоматы

На рисунке 4.1 показана диаграмма ДКА для первого слова.

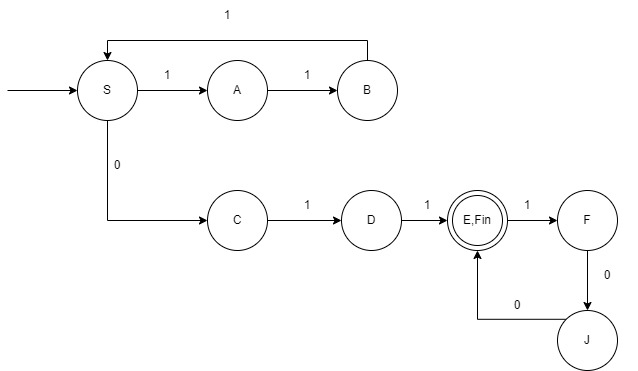


Рисунок 4.1 – Диаграмма ДКА для первого слова

В таблице 4.1 показана матрица ДКА для первого слова

|  |  |  |  |
| --- | --- | --- | --- |
|  | 1 | 0 | Явл. фин? |
| { S } | { A } | { C } | Нет |
| { A } | { B } |  | Нет |
| { B } | { S } |  | Нет |
| { C } | { D } |  | Нет |
| { D } | { E,Fin} |  | Нет |
| { E,Fin } | { F } |  | Да |
| { F } |  | { J} | Нет |
| { J } | { E,Fin } |  | Нет |

Таблица 4.1 – Матрица ДКА для первого слова

На рисунке 4.2 показана диаграмма ДКА для второго слова.

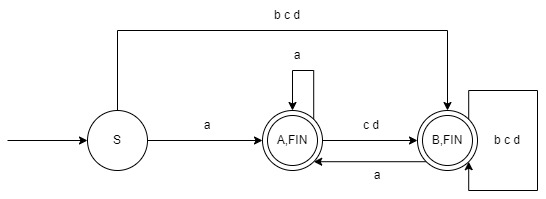


Рисунок 4.2 – Диаграмма ДКА для второго слова

В таблице 4.2 показана матрица ДКА для второго слова.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | a | b | c | d | Явл. Фин? |
| { S } | { A,Fin } | { B,Fin } | { B,Fin } | { B,Fin } | Нет |
| { A,Fin } | { A,Fin } |  | { B,Fin } | { B,Fin } | Да |
| { B,Fin } | { A,Fin } | { B,Fin } | { B,Fin } | { B,Fin } | Да |

Таблица 4.2 – Матрица ДКА для второго слова

На рисунке 4.3 показана диаграмма ДКА для комментария.

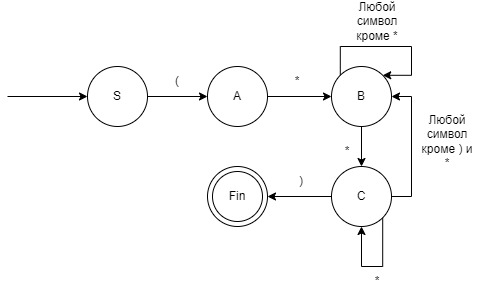


Рисунок 4.3 – Диаграмма ДКА для комментария

В таблице 4.3 показана матрица ДКА для комментария.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | ( | \* | Любой кроме \* | Любой кроме \* и ) | Явл.фин? |
| { S } | { A } |  |  |  | Нет |
| { A } |  | { B } |  |  | Нет |
| { B } |  | { C } | { B } |  | Нет |
| { C } | { FIN } | { C } |  | { B } | Нет |
| { FIN } |  |  |  |  | Да |

Таблица 4.3 – Матрица ДКА для комментария

На рисунке 4.4 представлен объединенный алгоритм лексического анализа.

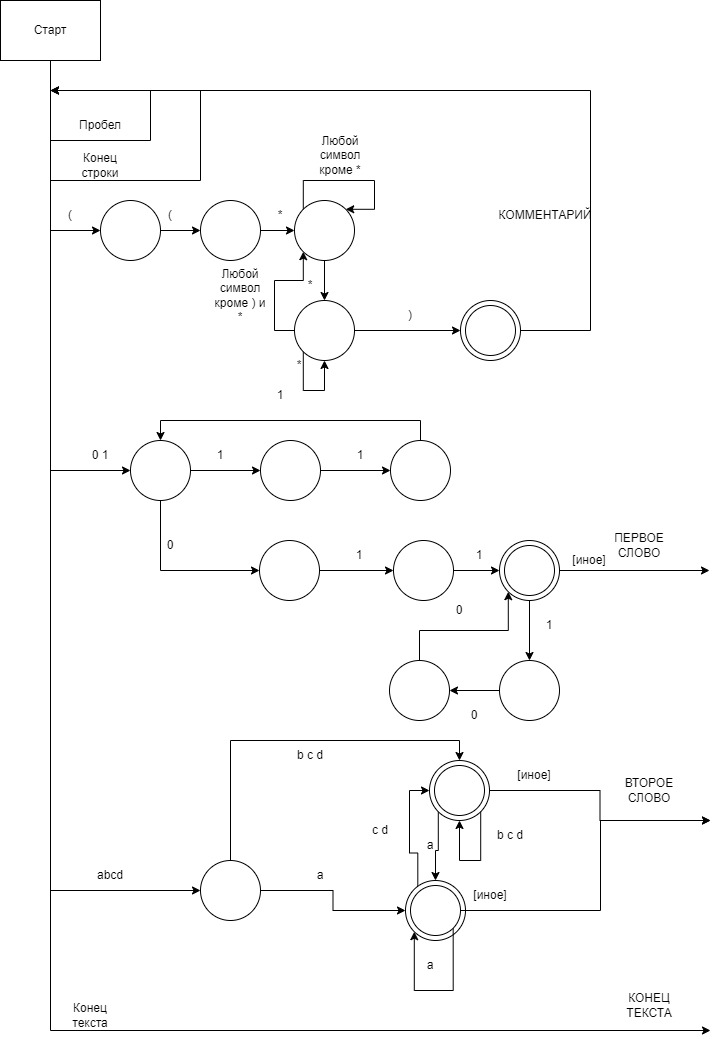


Рисунок 4.4 – Объединенный алгоритм лексического анализа

# Код программы

Ниже представлен код программы:

using System;

namespace LAB1

{

// Тип токена.

enum TokenKind

{

FirstWord, // Первое слово.

SecondWord, // Второе слово.

EndOfText, // Конец текста.

Unknown // Неизвестный.

};

// Класс "Токен".

class Token

{

private string value; // Значение токена (само слово).

private TokenKind type; // Тип токена.

// Позиция токена в исходном тексте.

private int lineIndex; // Индекс строки.

private int symStartIndex; // Индекс символа в строке lineIndex, с которого начинается токен.

// Значение токена (само слово).

public string Value

{

get { return value; }

set { this.value = value; }

}

// Тип токена.

public TokenKind Type

{

get { return type; }

set { this.type = value; }

}

// Индекс строки в исходном тексте, на которой расположен токен.

public int LineIndex

{

get { return lineIndex; }

set { this.lineIndex = value; }

}

// Индекс символа в строке LineIndex в исходном тексте, с которого начинается токен.

public int SymStartIndex

{

get { return symStartIndex; }

set { this.symStartIndex = value; }

}

// Сбросить значения полей токена.

public void Reset()

{

this.value = "";

this.type = TokenKind.Unknown;

this.lineIndex = -1;

this.symStartIndex = -1;

}

// Конструктор токена.

public Token()

{

Reset(); // Сбрасываем значения полей токена.

}

}

// Класс исключительных ситуаций лексического анализа.

class LexAnException : Exception

{

// Позиция возникновения исключительной ситуации в анализируемом тексте.

private int lineIndex; // Индекс строки.

private int symIndex; // Индекс символа.

// Индекс строки, где возникла исключительная ситуация - свойство только для чтения.

public int LineIndex

{

get { return lineIndex; }

}

// Индекс символа, на котором возникла исключительная ситуация - свойство только для чтения.

public int SymIndex

{

get { return symIndex; }

}

// Конструктор исключительной ситуации.

// message - описание исключительной ситуации.

// lineIndex и symIndex - позиция возникновения исключительной ситуации в анализируемом тексте.

public LexAnException(string message, int lineIndex, int symIndex) : base(message)

{

this.lineIndex = lineIndex;

this.symIndex = symIndex;

}

}

// Класс "Лексический анализатор".

// При обнаружении ошибки в исходном тексте он генерирует исключительную ситуацию LexAnException.

class LexicalAnalyzer

{

// Тип символа.

enum SymbolKind

{

Letter, // Буква от а до d.

Digit, // Цифра 0 или 1.

Space, // Пробел.

Reserved, // Зарезервированный.

Other, // Другой.

EndOfLine, // Конец строки.

EndOfText // Конец текста.

};

private const char commentSymbol1 = '('; // Первый символ комментария.

private const char commentSymbol2 = '\*'; // Второй символ комментария.

private const char commentSymbol3 = ')'; // символ комментария.

private string inputLines; // Входной текст - массив строк.

private int curLineIndex; // Индекс текущей строки.

private int curSymIndex; // Индекс текущего символа в текущей строке.

private int pointer;

private char curSym; // Текущий символ.

private SymbolKind curSymKind; // Тип текущего символа.

private Token token; // Токен, распознанный при последнем вызове метода RecognizeNextToken().

// Обработать лексическую ошибку.

// msg - описание ошибки.

private void LexicalError(string msg)

{

// Генерируем исключительную ситуацию, тем самым полностью прерывая процесс анализа текста.

throw new LexAnException(msg, curLineIndex, curSymIndex);

}

// Классифицировать текущий символ.

private void ClassifyCurrentSymbol()

{

if (((int)curSym >= (int)'a') && ((int)curSym <= (int)'d')) // Если текущий символ лежит в диапазоне строчных латинских букв.

{

curSymKind = SymbolKind.Letter; // Тип текущего символа - буква.

}

else if (((int)curSym >= (int)'0') && ((int)curSym <= (int)'1')) // Если текущий символ лежит в диапазоне цифр.

{

curSymKind = SymbolKind.Digit; // Тип текущего символа - цифра.

}

else

{

switch (curSym)

{

case ' ': // Если текущий символ - пробел.

curSymKind = SymbolKind.Space; // Тип текущего символа - пробел.

break;

// Если текущий символ - точка или первый символ комментария или второй символ комментария или символ подчеркивания.

case commentSymbol1:

case commentSymbol2:

case commentSymbol3:

curSymKind = SymbolKind.Reserved; // Тип текущего символа - зарезервированный.

break;

default:

curSymKind = SymbolKind.Other; // Тип текущего символа - другой.

break;

}

}

}

// Считать следующий символ.

private void ReadNextSymbol()

{

if (pointer >= inputLines.Length-1) // Если индекс текущей строки выходит за пределы текстового поля.

{

curSym = (char)0; // Обнуляем значение текущего символа.

curSymKind = SymbolKind.EndOfText; // Тип текущего символа - конец текста.

return;

}

curSymIndex++; // Увеличиваем индекс текущего символа.

pointer++; // Увеличиваем указатель

curSym = inputLines[pointer]; // Считываем текущий символ.

if (curSym == '\n')

{

curSymIndex = -1;

curSym = (char)0;

curSymKind = SymbolKind.EndOfLine;

curLineIndex++;

return;

}

ClassifyCurrentSymbol(); // Классифицируем текущий символ.

}

// Распознать идентификатор.

private void RecognizeFirstWord()

{

goto S; // Запускаем конечный автомат.

// Конечный автомат для идентификатора.

// S - начальное состояние.

//----------------------------------------------------//

S:

if (curSymKind == SymbolKind.Digit && curSym=='1')

{

token.Value += curSym; // Наращиваем значение текущего токена.

ReadNextSymbol(); // Читаем следующий символ в тексте.

goto A; // Переходим в указанное состояние.

}

else if (curSymKind==SymbolKind.Digit && curSym == '0')

{

token.Value += curSym; // Наращиваем значение текущего токена.

ReadNextSymbol(); // Читаем следующий символ в тексте.

goto C; // Переходим в указанное состояние.

}

else LexicalError("Ожидалась 0 или 1"); // Обнаружена ошибка в тексте.

A:

if (curSymKind == SymbolKind.Digit && curSym == '1')

{

token.Value += curSym; // Наращиваем значение текущего токена.

ReadNextSymbol(); // Читаем следующий символ в тексте.

goto B; // Переходим в указанное состояние.

}

else LexicalError("Ожидалась 1");

B:

if (curSymKind == SymbolKind.Digit && curSym == '1')

{

token.Value += curSym; // Наращиваем значение текущего токена.

ReadNextSymbol(); // Читаем следующий символ в тексте.

goto S; // Переходим в указанное состояние.

}

else LexicalError("Ожидалась 1");

C:

if (curSymKind==SymbolKind.Digit && curSym == '1')

{

token.Value += curSym; // Наращиваем значение текущего токена.

ReadNextSymbol(); // Читаем следующий символ в тексте.

goto D; // Переходим в указанное состояние.

}

else LexicalError("Ожидалась 1");

D:

if(curSymKind == SymbolKind.Digit && curSym == '1')

{

token.Value += curSym; // Наращиваем значение текущего токена.

ReadNextSymbol(); // Читаем следующий символ в тексте.

goto E\_FIN; // Переходим в указанное состояние.

}

else LexicalError("Ожидалась 1");

E\_FIN:

if (curSymKind == SymbolKind.Digit && curSym == '1')

{

token.Value += curSym; // Наращиваем значение текущего токена.

ReadNextSymbol(); // Читаем следующий символ в тексте.

goto F; // Переходим в указанное состояние.

}

else

goto Quit; // Выходим из конечного автомата.

//----------------------------------------------------//

// Конец конечного автомата для идентификатора.

F:

if (curSymKind == SymbolKind.Digit && curSym == '0')

{

token.Value += curSym; // Наращиваем значение текущего токена.

ReadNextSymbol(); // Читаем следующий символ в тексте.

goto J; // Переходим в указанное состояние.

}

else LexicalError("Ожидался 0");

J:

if (curSymKind == SymbolKind.Digit && curSym == '0')

{

token.Value += curSym; // Наращиваем значение текущего токена.

ReadNextSymbol(); // Читаем следующий символ в тексте.

goto E\_FIN; // Переходим в указанное состояние.

}

else LexicalError("Ожидался 0");

Quit:

token.Type = TokenKind.FirstWord; // Тип распознанного токена - идентификатор.

return;

}

// Распознать число (целое или вещественное).

private void RecognizeSecondWord()

{

goto S; // Запускаем конечный автомат.

// Конечный автомат для числа.

// S - начальное состояние.

//----------------------------------------------------//

S:

if (curSymKind == SymbolKind.Letter && curSym=='a')

{

token.Value += curSym; // Наращиваем значение текущего токена.

ReadNextSymbol(); // Читаем следующий символ в тексте.

goto A\_FIN; // Переходим в указанное состояние.

}

else if(curSymKind == SymbolKind.Letter)

{

token.Value += curSym; // Наращиваем значение текущего токена.

ReadNextSymbol(); // Читаем следующий символ в тексте.

goto B\_FIN; // Переходим в указанное состояние.

}

else LexicalError("Ожидалась буква a b c или d"); // Обнаружена ошибка в тексте.

A\_FIN:

if (curSymKind == SymbolKind.Letter && curSym == 'b')

{

LexicalError("Не может быть подстроки ab");

}

else if (curSymKind == SymbolKind.Letter && curSym=='a')

{

token.Value += curSym; // Наращиваем значение текущего токена.

ReadNextSymbol(); // Читаем следующий символ в тексте.

goto A\_FIN; // Переходим в указанное состояние.

}

else if (curSymKind == SymbolKind.Letter)

{

token.Value += curSym; // Наращиваем значение текущего токена.

ReadNextSymbol(); // Читаем следующий символ в тексте.

goto B\_FIN; // Переходим в указанное состояние.

}

else

goto Quit; // Выходим из конечного автомата.

B\_FIN:

if (curSymKind == SymbolKind.Letter && curSym!='a')

{

token.Value += curSym; // Наращиваем значение текущего токена.

ReadNextSymbol(); // Читаем следующий символ в тексте.

goto B\_FIN; // Переходим в указанное состояние.

}

else if (curSymKind == SymbolKind.Letter && curSym == 'a')

{

token.Value += curSym; // Наращиваем значение текущего токена.

ReadNextSymbol(); // Читаем следующий символ в тексте.

goto A\_FIN; // Переходим в указанное состояние.

}

else

goto Quit;

//----------------------------------------------------//

// Конец конечного автомата для второго слова.

Quit:

token.Type = TokenKind.SecondWord; // Тип распознанного токена - второе слово.

return;

}

// Пропустить комментарий.

private void SkipComment()

{

goto S; // Запускаем конечный автомат.

// Конечный автомат для комментария.

// S - начальное состояние.

//----------------------------------------------------//

S:

if (curSym == commentSymbol1)

{

ReadNextSymbol(); // Читаем следующий символ в тексте.

goto A; // Переходим в указанное состояние.

}

else

LexicalError("Ожидалось " + commentSymbol1); // Обнаружена ошибка в тексте.

A:

if (curSym == commentSymbol2)

{

ReadNextSymbol(); // Читаем следующий символ в тексте.

goto B; // Переходим в указанное состояние.

}

else

LexicalError("Ожидалось " + commentSymbol2); // Обнаружена ошибка в тексте.

B:

if (curSym == commentSymbol2)

{

ReadNextSymbol(); // Читаем следующий символ в тексте.

goto C; // Переходим в указанное состояние.

}

else if (curSymKind == SymbolKind.EndOfText)

LexicalError("Незаконченный комментарий"); // Обнаружена ошибка в тексте.

else

{

ReadNextSymbol(); // Читаем следующий символ в тексте.

goto B; // Переходим в указанное состояние.

}

C:

if (curSym == commentSymbol3)

{

ReadNextSymbol(); // Читаем следующий символ в тексте.

goto Fin; // Переходим в указанное состояние.

}

else if (curSym == commentSymbol2)

{

ReadNextSymbol(); // Читаем следующий символ в тексте.

goto C; // Переходим в указанное состояние.

}

else if (curSymKind == SymbolKind.EndOfText)

LexicalError("Незаконченный комментарий"); // Обнаружена ошибка в тексте.

else

{

ReadNextSymbol(); // Читаем следующий символ в тексте.

goto B; // Переходим в указанное состояние.

}

Fin:

goto Quit; // Выходим из конечного автомата.

//----------------------------------------------------//

// Конец конечного автомата для комментария.

Quit:

return;

}

// Конструктор лексического анализатора.

// В качестве параметра передается исходный текст.

public LexicalAnalyzer(string inputLines)

{

this.inputLines = inputLines;

// Обнуляем поля.

curLineIndex = 0;

curSymIndex = -1;

pointer = -1;

curSym = (char)0;

token = null;

// Считываем первый символ входного текста.

ReadNextSymbol();

}

// Токен, распознанный при последнем вызове метода RecognizeNextToken() - свойство только для чтения.

public Token Token

{

get { return token; }

}

// Распознать следующий токен в тексте.

public void RecognizeNextToken()

{

// На данный момент уже прочитан символ, следующий за токеном, распознанным в прошлом вызове этого метода.

// Если же это первый вызов, то на данный момент уже прочитан первый символ текста (в конструкторе).

// Цикл пропуска пробелов, переходов на новую строку, комментариев.

while ( (curSymKind == SymbolKind.Space) ||

(curSymKind == SymbolKind.EndOfLine) ||

(curSym == commentSymbol1) )

{

if (curSym == commentSymbol1) // Если текущий символ - первый символ комментария.

SkipComment(); // Пропускаем комментарий.

else

ReadNextSymbol(); // Пропускаем пробел или переход на новую строку.

}

// Создаем новый экземпляр токена.

token = new Token();

// Запоминаем позицию начала токена в исходном тексте.

token.LineIndex = curLineIndex;

token.SymStartIndex = curSymIndex;

switch (curSymKind) // Анализируем текущий символ.

{

case SymbolKind.Digit: // Если текущий символ - буква.

RecognizeFirstWord(); // Вызываем процедуру распознавания идентификатора.

break;

case SymbolKind.Letter: // Если текущий символ - цифра.

RecognizeSecondWord(); // Вызываем процедуру распознавания числа (целого или вещественного).

break;

case SymbolKind.EndOfText: // Если текущий символ - конец текста.

token.Type = TokenKind.EndOfText; // Тип распознанного токена - конец текста.

break;

default: // Если текущий символ - какой-то другой.

LexicalError("Недопустимый символ"); // Обнаружена ошибка в тексте.

break;

}

}

}

}

# Пример работы программы

На рисунке 6.1 показана работа программы с верными входными данными.

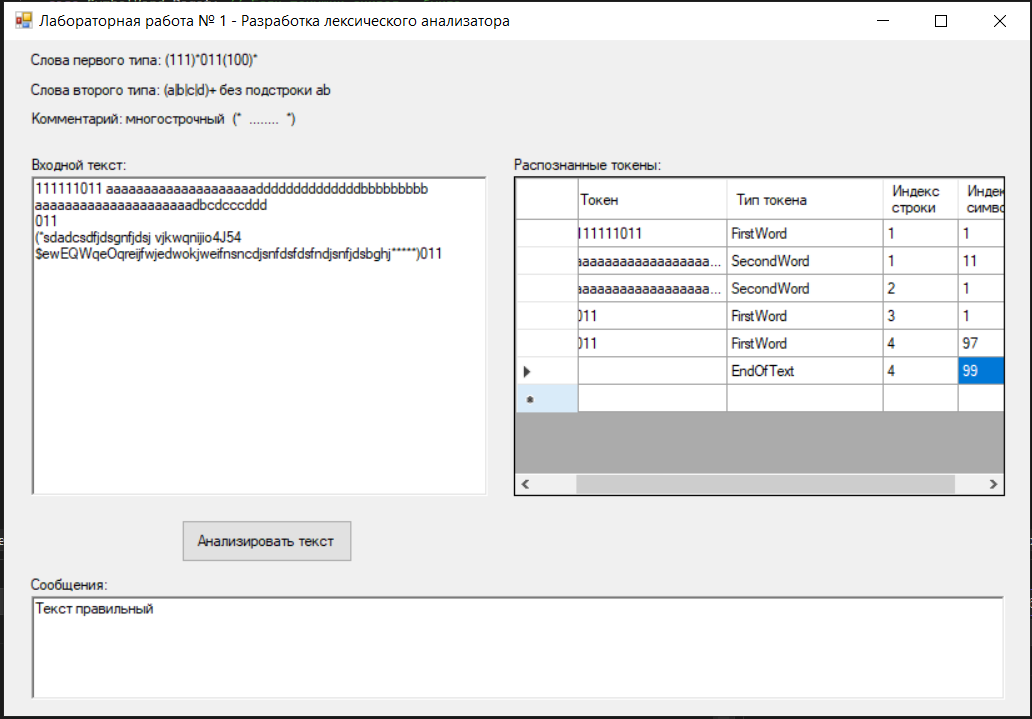


Рисунок 6.1 - Работа программы с верными входными данными

На рисунке 6.2 показана работа программы с неверными входными данными.

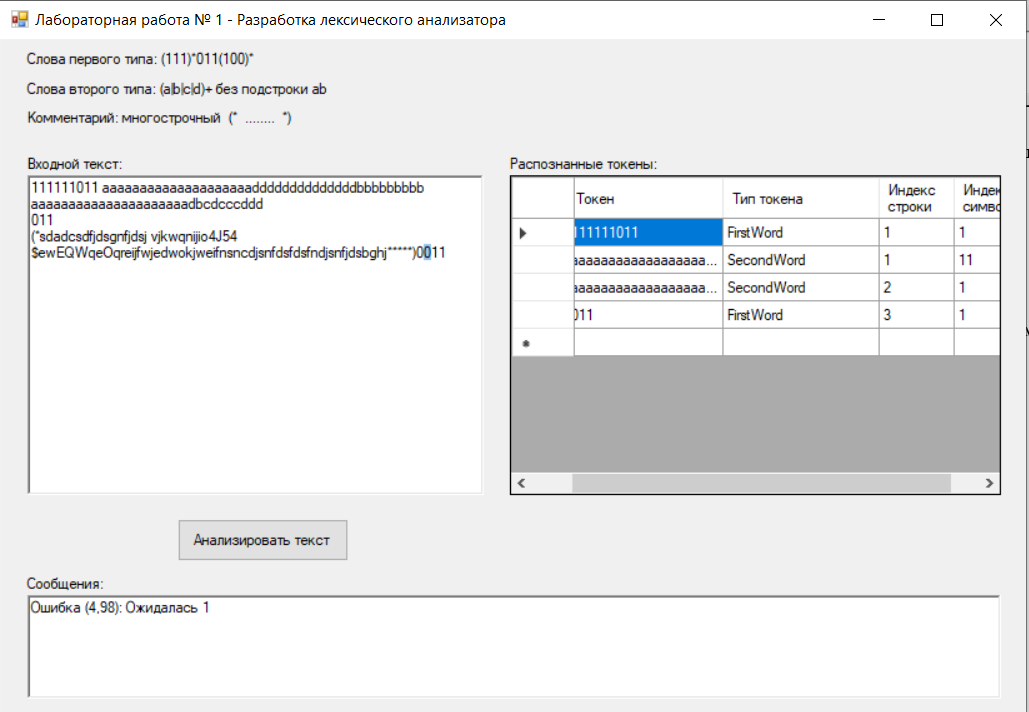


Рисунок 6.2 - Работа программы с неверными входными данными