Лабораторна робота №1

з дисципліни

Теорія формальних мов та компіляція

на тему

«Розробка лексичного аналізатора»

Варіант №17

|  |  |
| --- | --- |
| Виконав студент  групи КВ-64М  Подольський С. В.  залікова книжка № КВ6415 | Перевірив:  Марченко О. І.  \_\_\_\_\_\_\_\_\_ |

1. **Постановка задачі**

Розробити програму лексичного аналізатора (ЛА) для підмножини мови програмування SIGNAL.

Програма має забезпечувати наступне (якщо це передбачається граматикою варіанту):

* згортання ідентифікаторів;
* згортання ключових слів;
* згортання цілих десяткових констант;
* згортання дійсних десяткових констант;
* згортання строкових констант, якщо вони визначені в заданій мові;

Також у всіх варіантах необхідно забезпечити:

* видалення коментарів, заданих у вигляді (\*<текст коментарю>\*)

Для кодування лексем необхідно використовувати числові діапазони, вказані в Таблиці 1.

Діапазони кодування лексем

|  |  |
| --- | --- |
| Вид лексеми | Числовий діапазон |
| односимвольні роздільники та знаки операцій (: / ; + і т.д.) | 0 – 255,  Тобто коди ASCII |
| багатосимвольні роздільники ( := ,<= ,<=, і т.д.) | 301 - 400 |
| цілі десяткові константи | 401 - 500 |
| символьні константи | 501 - 600 |
| рядкові константи | 601 - 700 |
| ключові слова (BEGIN, END, FOR) та ідентифікатори | 701 – 1000 |

**Входом** ЛА має бути наступне:

* вихідна програма, написана підмножиною мови SIGNAL відповідно до варіанту;
* таблиця кодів ASCII з атрибутами для визначення токенів;
* таблиця багато символьних роздільників;
* таблиця ідентифікаторів, в яку попередньо занесені ключові слова з атрибутом ключового слова;

**Вихід** ЛА має бути наступним:

* закодований рядок лексем;
* таблиці ідентифікаторів, числових, символьних та рядкових констант, сформовані для конкретної програми;

1. **Індивідуальне завдання – граматика відповідно до варіанту**

<signal-program> → <program>

<program> → PROGRAM <procedure-identifier> ; <block>.

<block> → BEGIN <statements-list> END

<statements-list> → <statement> <statements-list> | <empty>

<statement> → LOOP <statements-list> ENDLOOP ; | FOR <variable-identifier> := <loop-declaration> ENDFOR ;

<loop-declaration> → <expression> TO <expression> DO <statements-list>

<expression> → <summand> <summands-list> | - <summand> <summands-list>

<summands-list> → <add-instruction> <summand> <summands-list> | <empty>

<add-instruction> → + | -

<summand> → <variable-identifier> | <unsigned-integer>

<variable-identifier> → <identifier>

<procedure-identifier> → <identifier>

<identifier> → <letter><string>

<string> → <letter><string> | <digit><string> | <empty>

<unsigned-integer> → <digit><digits-string>

<digits-string> → <digit><digits-string> | <empty>

<digit> → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<letter> → A | B | C | D | ... | Z

1. **Таблиця переходів автомату (реалізована програмно)**

// Ready Identifier Number Delimiter Assignment CommentBegin Comment CommentEnd

{/\* Letter \*/ State.Identifier, State.Identifier, State.Identifier, State.Identifier, State.Error, State.Error, State.Comment, State.Comment },

{/\* Digit \*/ State.Number, State.Identifier, State.Number, State.Number, State.Error, State.Error, State.Comment, State.Comment },

{/\* Delimiter \*/ State.Delimiter, State.Delimiter, State.Delimiter, State.Delimiter, State.Error, State.Error, State.Comment, State.Comment },

{/\* DoubleDot \*/ State.Assignment, State.Assignment, State.Assignment, State.Assignment, State.Error, State.Error, State.Comment, State.Comment },

{/\* Equality \*/ State.Error, State.Error, State.Error, State.Error, State.Delimiter,State.Error, State.Comment, State.Comment },

{/\* LeftBracket \*/ State.CommentBegin, State.CommentBegin, State.CommentBegin, State.CommentBegin, State.Error, State.Error, State.Comment, State.Comment },

{/\* RightBracket \*/ State.Error, State.Error, State.Error, State.Error, State.Error, State.Error, State.Comment, State.Ready },

{/\* Asterisk \*/ State.Error, State.Error, State.Error, State.Error, State.Error, State.Comment, State.CommentEnd, State.CommentEnd},

{/\* Whitespace\*/ State.Ready, State.Ready, State.Ready, State.Ready, State.Error, State.Error, State.Comment, State.Comment },

{/\* Other \*/ State.Error, State.Error, State.Error, State.Error, State.Error, State.Error, State.Comment, State.Comment },

{/\* EndOfFile \*/ State.Ready, State.Error, State.Error, State.Ready, State.Error, State.Error, State.Error, State.Error }

1. **Лістинг вихідної програми лексичного аналізатора**

using System;

using System.Collections.Generic;

namespace Scanner

{

class StateMachine

{

enum Attribute

{

Letter,

Digit,

Delimiter,

DoubleDot,

Equality,

LeftBracket,

RightBracket,

Asterisk,

Whitespace,

Other,

EndOfFile

}

enum State

{

Ready,

Identifier,

Number,

Delimiter,

Assignment,

CommentBegin,

Comment,

CommentEnd,

Error,

Done

}

static readonly State[,] TransitionTable =

{

// Ready Identifier Number Delimiter Assignment CommentBegin Comment CommentEnd

{/\* Letter \*/ State.Identifier, State.Identifier, State.Identifier, State.Identifier, State.Error, State.Error, State.Comment, State.Comment },

{/\* Digit \*/ State.Number, State.Identifier, State.Number, State.Number, State.Error, State.Error, State.Comment, State.Comment },

{/\* Delimiter \*/ State.Delimiter, State.Delimiter, State.Delimiter, State.Delimiter, State.Error, State.Error, State.Comment, State.Comment },

{/\* DoubleDot \*/ State.Assignment, State.Assignment, State.Assignment, State.Assignment, State.Error, State.Error, State.Comment, State.Comment },

{/\* Equality \*/ State.Error, State.Error, State.Error, State.Error, State.Delimiter,State.Error, State.Comment, State.Comment },

{/\* LeftBracket \*/ State.CommentBegin, State.CommentBegin, State.CommentBegin, State.CommentBegin, State.Error, State.Error, State.Comment, State.Comment },

{/\* RightBracket \*/ State.Error, State.Error, State.Error, State.Error, State.Error, State.Error, State.Comment, State.Ready },

{/\* Asterisk \*/ State.Error, State.Error, State.Error, State.Error, State.Error, State.Comment, State.CommentEnd, State.CommentEnd},

{/\* Whitespace\*/ State.Ready, State.Ready, State.Ready, State.Ready, State.Error, State.Error, State.Comment, State.Comment },

{/\* Other \*/ State.Error, State.Error, State.Error, State.Error, State.Error, State.Error, State.Comment, State.Comment },

{/\* EndOfFile \*/ State.Ready, State.Error, State.Error, State.Ready, State.Error, State.Error, State.Error, State.Error }

};

readonly Action[,] ActionTable;

public StateMachine()

{

ActionTable = new Action[,]

{

// Ready Identifier Number Delimiter Assignment CommentBegin Comment CommentEnd

{/\* Letter \*/ () => {}, null, AddNumber, AddDelimiter, Error2, Error0, null, null },

{/\* Digit \*/ () => {}, null, null, AddDelimiter, Error2, Error0, null, null },

{/\* Delimiter \*/ () => {}, AddIdentifier, AddNumber, AddDelimiter, Error2, Error0, null, null },

{/\* DoubleDot \*/ () => {}, AddIdentifier, AddNumber, AddDelimiter, Error2, Error0, null, null },

{/\* Equality \*/ Error1, Error1, Error1, Error1, null, Error0, null, null },

{/\* LeftBracket \*/ null, AddIdentifier, AddNumber, AddDelimiter, Error2, Error0, null, null },

{/\* RightBracket \*/ Error1, Error1, Error1, Error1, Error2, Error0, null, null },

{/\* Asterisk \*/ Error1, Error1, Error1, Error1, Error2, null, null, null },

{/\* Whitespace\*/ null, AddIdentifier, AddNumber, AddDelimiter, Error2, Error0, null, null },

{/\* Other \*/ Error1, Error1, Error1, Error1, Error2, Error0, null, null },

{/\* EndOfFile \*/ null, Error3, Error3, AddDelimiter, Error3, Error3, Error3, Error3 }

};

#region Define table of attributes

for (var c = 0; c < attributes.Length; c++)

attributes[c] = Attribute.Other;

for (var c = '0'; c <= '9'; c++)

attributes[c] = Attribute.Digit;

for (var c = 'A'; c <= 'Z'; c++)

attributes[c] = attributes[char.ToLower(c)] = Attribute.Letter;

attributes['+'] = attributes['-'] = attributes[';'] = attributes['.'] = Attribute.Delimiter;

attributes[':'] = Attribute.DoubleDot;

attributes['='] = Attribute.Equality;

attributes['\*'] = Attribute.Asterisk;

attributes['('] = Attribute.LeftBracket;

attributes[')'] = Attribute.RightBracket;

attributes[' '] = attributes['\t'] = attributes['\n'] = attributes['\r'] = Attribute.Whitespace;

#endregion

#region Define keywords

foreach (string s in new[] { "program", "begin", "end", "loop", "endloop", "for", "to", "do", "endfor" })

identifiers[s] = identifier++;

#endregion

#region Define long delimiters

longDelimiters.Add(":=", longDelimiter++);

#endregion

}

Attribute[] attributes = new Attribute[char.MaxValue + 1];

int longDelimiter = 301, constant = 401, identifier = 701;

public Dictionary<string, int> identifiers = new Dictionary<string, int>();

public Dictionary<string, int> constants = new Dictionary<string, int>();

public Dictionary<string, int> longDelimiters = new Dictionary<string, int>();

string token = string.Empty;

List<int> tokenString = new List<int>();

int line = 1, column = 0;

public List<int> Scan(string text)

{

var state = State.Ready;

foreach (char c in text)

{

// Get current position in file

if (c == '\n')

{

line++;

column = 0;

}

else

column++;

// Increase state machine tact

var attribute = attributes[c];

var action = ActionTable[(int)attribute, (int)state];

state = TransitionTable[(int)attribute, (int)state];

if (action != null)

{

//token = token.Trim();

action();

token = string.Empty;

}

token += c;

}

// Handle end of file

var finalAction = ActionTable[(int)Attribute.EndOfFile, (int)state];

if (finalAction != null)

finalAction();

return tokenString;

}

#region Actions

void AddNumber()

{

if (!constants.ContainsKey(token))

constants[token] = constant++;

tokenString.Add(constants[token]);

}

void AddIdentifier()

{

token = token.ToLower();

if (!identifiers.ContainsKey(token))

identifiers[token] = identifier++;

tokenString.Add(identifiers[token]);

}

void AddDelimiter()

{

if (token.Length > 1)

tokenString.Add(longDelimiters[token]);

else

tokenString.Add(token[0]);

}

#endregion

#region Errors

public class LexicalException : Exception

{

public LexicalException(string message) : base(message) { }

public LexicalException(string message, int line, int column)

: this(string.Format("Error at line {0}, column {1}: {2}", line, column, message)) { }

}

void Error0()

{

throw new LexicalException("'\*' expected after comment begin '('", line, column);

}

void Error1()

{

throw new LexicalException("Unexpected symbol", line, column);

}

void Error2()

{

throw new LexicalException("'=' expected after ':'", line, column);

}

void Error3()

{

throw new LexicalException("Unexpected end of file", line, column);

}

#endregion

}

}

1. **Контрольні приклади**

|  |  |
| --- | --- |
| **Контрольний приклад вхідної програми** | **Згенерований рядок лексем та таблиці** |
| PROGRAM sample;  BEGIN  LOOP  ENDLOOP;    FOR i := 1 TO -4+20+variable DO  ENDFOR;  (\*comment\*)  END. | Program output token string:  701 710 59 702 704 705 59 706 711 301 401 707 45 402 43 403 43 712 708 709 59 703 46  Table of identifiers:  701 program  702 begin  703 end  704 loop  705 endloop  706 for  707 to  708 do  709 endfor  710 sample  711 i  712 variable  Table of numerical constants:  401 1  402 4  403 20  Table of long delimiters:  301 :=  Press any key to continue . . . |
| PROGRAM sample0;  BEGIN  LOOP  ENDLOOP;    FOR i := 1 TO -4+20+variable DO  ENDFOR;  (comment\*)  END. | Error at line 13, column 3: '\*' expected after comment begin '('  Press any key to continue . . . |
| PROGRAM sample1;  BEGIN  LOOP  ENDLOOP;    FOR i = 1 TO -4+20+variable DO  ENDFOR;  (\*comment\*)  END. | Error at line 9, column 8: Unexpected symbol  Press any key to continue . . . |
| PROGRAM sample2;  BEGIN  LOOP  ENDLOOP;    FOR i : 1 TO -4+20+variable DO  ENDFOR;  (\*comment\*)  END. | Error at line 9, column 9: '=' expected after ':'  Press any key to continue . . . |
| PROGRAM sample3;  BEGIN  LOOP  ENDLOOP;    FOR i := 1 TO -4+20+variable DO  ENDFOR;  (\*comment | Error at line 13, column 10: Unexpected end of file  Press any key to continue . . . |