**C#**

**11**

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

ФАКУЛЬТЕТ ІНФОРМАТИКИ І ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

КАФЕДРА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

**Лабораторна робота №3**

з дисципліни **«**Системне програмування 2**»**

Виконала:

студентка 2 курсу гр. ІВ-71

Молчанова В.С.

Перевірив:

Павлов В.Г.

Київ 2019 р.

**Тема:** створення та настроювання лексичних аналізаторів на основі автоматної граматики

**Мета**: Вивчення схеми табличного подання автоматної граматики лексичного аналізу. Використання об’єктів стану графів автоматів для формування лексем у форматі внутрішнього подання вузлів графів розбору.

**Варіант:** 

**Лістинг програми:**

public static class Lab3

{

public const string Expression = "if(c)b=sin(2\*a);else b=2\*a;";

public static readonly List<string> Operators = new List<string>

{

"-", "+", "/", "\*", "=", "&", "&&", "|", "||",

"%", "++", "--", "!=", "==", ">", "<", ">=",

"<=", "!", "~", "^", "<<", ">>",

"-=", "+=", "/=", "\*=", "&=", "|=", "^=", "<<=", ">>=", "->", "."

};

public static readonly List<string> ReservedWords = new List<string>

{

"auto", "break", "case", "char", "const", "continue", "default",

"do", "double", "else", "enum", "extern", "float", "for", "goto", "if",

"inline", "int", "long", "register", "restrict", "return", "short",

"signed", "sizeof", "static", "struct", "switch", "typedef", "union",

"unsigned", "void", "volatile", "while"

};

public static readonly List<string> Brackets = new List<string>{"()", "{}", "[]"};

public static readonly List<string> Separators = new List<string>

{

",", ";", ":", "…"

};

public static void DoWork()

{

var noErrors = true;

var bracketsAccumulator = new Dictionary<string, int>();

var withoutBracketsAndSpaces = SplitByBracketsAndSpaces(Expression, bracketsAccumulator);

if (withoutBracketsAndSpaces == null)

{

Console.WriteLine("There is a bracketing error in the input");

}

else

{

var separatorsAccumulator = new Dictionary<string, int>();

var operatorsAccumulator = new Dictionary<string, int>();

var words = new List<string>();

foreach (var part in withoutBracketsAndSpaces)

{

var withoutSeparators = SplitWithCount(part, Separators, separatorsAccumulator);

foreach (var str in withoutSeparators)

{

words.AddRange(SplitWithCount(str, Operators, operatorsAccumulator));

}

}

var reservedWordsAccumulator = new Dictionary<string, int>();

var namesAccumulator = new Dictionary<string, int>();

var numbersAccumulator = new Dictionary<string, int>();

foreach (var word in words.Where(w => w != ""))

{

if (ReservedWords.Contains(word))

{

if (!reservedWordsAccumulator.ContainsKey(word)) reservedWordsAccumulator.Add(word, 0);

reservedWordsAccumulator[word]++;

}

else if (IsNumericalConstant(word))

{

if (!numbersAccumulator.ContainsKey(word)) numbersAccumulator.Add(word, 0);

numbersAccumulator[word]++;

}

else if (CanBeName(word))

{

if (!namesAccumulator.ContainsKey(word)) namesAccumulator.Add(word, 0);

namesAccumulator[word]++;

}

else

{

Console.WriteLine($"{word} is not a reserved word and can't be a variable name");

noErrors = false;

break;

}

}

if (noErrors)

{

Console.WriteLine($"Expression '{Expression}' contains");

PrintResult("Brackets", bracketsAccumulator);

PrintResult("Separators", separatorsAccumulator);

PrintResult("Operators", operatorsAccumulator);

PrintResult("Reserved words", reservedWordsAccumulator);

PrintResult("Variable or method names", namesAccumulator);

PrintResult("Numerical constants", numbersAccumulator);

}

}

Console.ReadLine();

}

private static void PrintResult(string title, Dictionary<string, int> accumulator)

{

if (accumulator.Any())

{

Console.Write($"{title}:\n\t");

foreach (var item in accumulator)

{

Console.Write($"{item.Key} ({item.Value}); ");

}

Console.WriteLine();

}

}

private static List<string> SplitByBracketsAndSpaces(string str, Dictionary<string, int> bracketsAccumulator)

{

var newString = (string)str.Clone();

foreach (var bracketsPair in Brackets)

{

var openedBracket = bracketsPair[0];

var closedBracket = bracketsPair[1];

var opened = 0;

foreach (var symbol in str)

{

if (symbol == openedBracket)

{

opened++;

if (!bracketsAccumulator.ContainsKey(bracketsPair)) bracketsAccumulator.Add(bracketsPair, 0);

bracketsAccumulator[bracketsPair]++;

}

else if (symbol == closedBracket)

{

opened--;

}

if (opened < 0) return null;

}

newString = newString.Replace(openedBracket, ' ');

newString = newString.Replace(closedBracket, ' ');

}

return newString.Split(' ').ToList();

}

private static List<string> SplitWithCount(string str, List<string> separators,

Dictionary<string, int> separatorsAccumulator)

{

separators.Sort(delegate(string x, string y)

{

if (x.Length > y.Length) return -1;

if (x.Length < y.Length) return 1;

return 0;

});

var newStr = str.Clone().ToString();

foreach (var separator in separators)

{

var index = newStr.IndexOf(separator);

while (index != -1)

{

newStr = newStr.Substring(0, index) + " " +

newStr.Substring(index + separator.Length);

index = newStr.IndexOf(separator);

if(!separatorsAccumulator.ContainsKey(separator)) separatorsAccumulator.Add(separator, 0);

separatorsAccumulator[separator]++;

}

}

return newStr.Split(' ').Where(s => s != "").ToList();

}

private static bool CanBeName(string str)

{

return Char.IsLetter(str[0]);

}

private static bool IsNumericalConstant(string str)

{

return str.All(Char.IsDigit);

}

}

**Контрольні приклади:**

1.

Expression 'switch(c){case 0: b=2\*a[n]; break; default: b=d;}' contains

Brackets:

() (1); {} (1); [] (1);

Separators:

: (2); ; (3);

Operators:

= (2); \* (1);

Reserved words:

switch (1); case (1); break (1); default (1);

Variable or method names:

c (1); b (2); n (1);

Numerical constants:

0 (1); 2 (1);

2.

Expression 'do{--n; if(b==a[n]) return n;}while(n); return 0;' contains

Brackets:

() (2); {} (1); [] (1);

Separators:

; (4);

Operators:

-- (1); == (1);

Reserved words:

do (1); if (1); return (2); while (1);

Variable or method names:

n (4); b (1); a (1);

Numerical constants:

0 (1);

3.

Expression 'if(c)b=sin(2\*a);else b=2\*a;' contains

Brackets:

() (2);

Separators:

; (2);

Operators:

= (2); \* (2);

Reserved words:

if (1); else (1);

Variable or method names:

c (1); b (2); sin (1); a (2);

Numerical constants:

2 (2);