НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ “КПІ”

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп'ютерних систем

Лабораторна робота №3

*з дисципліни*

*“Інженерія програмного забезпечення”*

Виконав студент гр. КВ-22

Ткаченко Роман

Перевірив

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Київ 2015

Індивідуальне завдання:

**Варіант 19**

Постановка задачі

1. Розробити програму синтаксичного аналізатора (СА) для підмножини мови програмування SIGNAL згідно граматики за варіантом.

2. Програма має забезпечувати наступне:

· читання рядка лексем та таблиць, згенерованих лексичним аналізатором, який було розроблено в лабораторній роботі «Розробка лексичного аналізатора»;

· синтаксичний аналіз (розбір) програми, поданої рядком лексем (алгоритм синтаксичного аналізатора вибирається за варіантом);

· побудову дерева розбору;

· формування таблиць ідентифікаторів та різних констант з повною інформацією, необхідною для генерування коду;

· формування лістингу вхідної програми з повідомленнями про лексичні та синтаксичні помилки.

3. Для програмування може бути використана довільна алгоритмічна мова програмування високого рівня. Якщо обрана мова програмування має конструкції або бібліотеки для роботи з регулярними виразами, то використання цих конструкцій та/або бібліотек строго заборонено.

4. Входом синтаксичного аналізатора має бути наступне:

· закодований рядок лексем;

· таблиці ідентифікаторів, числових, символьних та рядкових констант (якщо це передбачено граматикою варіанту), згенеровані лексичним аналізатором;

· вхідна програма на підмножині мови програмування SIGNAL згідно з варіантом (необхідна для формування лістнигу програми).

5. Виходом синтаксичного аналізатора має бути наступне:

· дерево розбору вхідної програми;

· таблиці ідентифікаторів та різних констант з повною інформацією, необхідною для генерування коду;

· лістинг вхідної програми з повідомленнями про лексичні та синтаксичні помилки.

**Низхідний розбір за алгоритмом аналізуючої машини Кнута;**

**Граматика мови:**

1. <signal-program> --> <program>

2. <program> --> PROGRAM <procedure-identifier> ;

<block>.

3. <block> --> <variable-declarations> BEGIN

<statements-list> END

4. <variable-declarations> --> VAR <declarations-list> |

<empty>

5. <declarations-list> --> <declaration> <declarations-list> |

<empty>

6. <declaration> --><variable-identifier>:<attribute>

;

7. <attribute> --> INTEGER |

FLOAT

8. <statements-list> --> <statement> <statements-list> |

<empty>

9. <statement> --> <condition-statement> ENDIF ;

10. <condition-statement> --> <incomplete-conditionstatement><alternative-part>

11. <incomplete-condition-statement> --> IF <conditional-expression> THEN <statements-list>

12. <alternative-part> --> ELSE <statements-list> |

<empty>

13. <conditional-expression> --> <expression> = <expression>

14. <expression> --> <variable-identifier> |

<unsigned-integer>

15. <variable-identifier> --> <identifier>

16. <procedure-identifier> --> <identifier>

17. <identifier> --> <letter><string>

18. <string> --> <letter><string> |

<digit><string> |

<empty>

19. <unsigned-integer> --> <digit><digits-string>

20. <digits-string> --> <digit><digits-string> |

<empty>

21. <digit> --> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

9

22. <letter> --> A | B | C | D | ... | Z

Лістинг програми:

**XMLNode.cs:**

namespace lexer.SyntaxTree

{

public class XMLNode

{

private static int id = 0;

public XMLNode()

{

name = nodesTypes.node;

value = "";

nodes = new List<XMLNode>();

Id = id++.ToString();

}

public XMLNode(nodesTypes name)

{

this.name = name;

value = "";

nodes = new List<XMLNode>();

Id = id++.ToString();

}

[XmlAttribute]

public string Id;

[XmlAttribute]

public nodesTypes name;

[XmlAttribute]

public string value;

public List<XMLNode> nodes;

public XMLNode AddNode(XMLNode node)

{

nodes.Add(node);

return node;

}

}

}

**XMLNodeToDGMLParser.cs:**

namespace lexer.SyntaxTree

{

class XMLNodeToDGMLParser

{

public XMLNodeToDGMLParser()

{

XMLSyntaxTree = SerializeTables.DeseriaizeNode();

nodes = new List<Node>();

links = new List<Link>();

nodes.Add(new Node(XMLSyntaxTree.Id, XMLSyntaxTree.value, XMLSyntaxTree.name.ToString()));

graph = new Graph();

}

private XMLNode XMLSyntaxTree;

private List<Node> nodes;

private List<Link> links;

private Graph graph;

private void ParseNode(XMLNode parentNode)

{

foreach (var item in parentNode.nodes)

{

string label = item.name.ToString();

if (item.value != "")

{

label += ": ";

label += item.value;

}

nodes.Add(new Node() { Id = item.Id, Label = label, Value = item.value });

links.Add(new Link() { Source = parentNode.Id, Target = item.Id });

ParseNode(item);

}

}

public Graph GetGraph()

{

ParseNode(XMLSyntaxTree);

graph.Nodes = nodes.ToArray();

graph.Links = links.ToArray();

return graph;

}

}

}

public static void SeriaizeNodeGraph(SyntaxTree.Graph graph)

{

if (File.Exists(SyntaxTreeGraphPath))

{

File.Delete(SyntaxTreeGraphPath);

}

Type graphType = typeof(SyntaxTree.Graph);

XmlRootAttribute root = new XmlRootAttribute("DirectedGraph");

root.Namespace = "http://schemas.microsoft.com/vs/2009/dgml";

XmlSerializer serializer = new XmlSerializer(graphType, root);

XmlWriterSettings settings = new XmlWriterSettings();

settings.Indent = true;

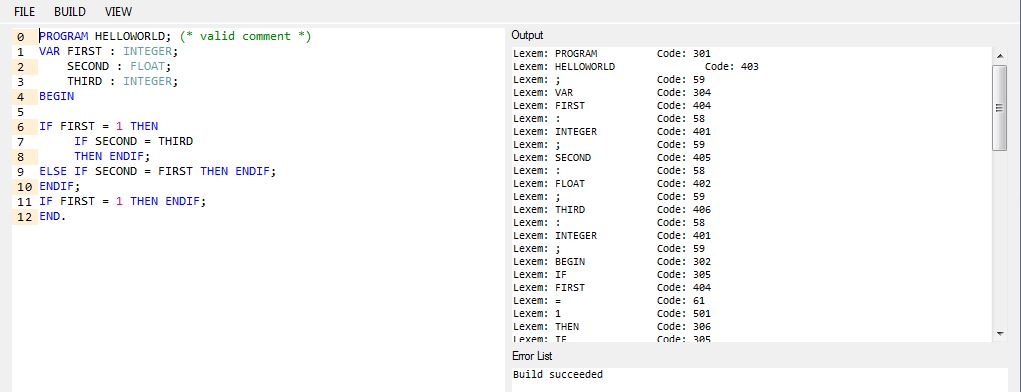
XmlWriter xmlWriter = XmlWriter.Create(SyntaxTreeGraphPath, settings);

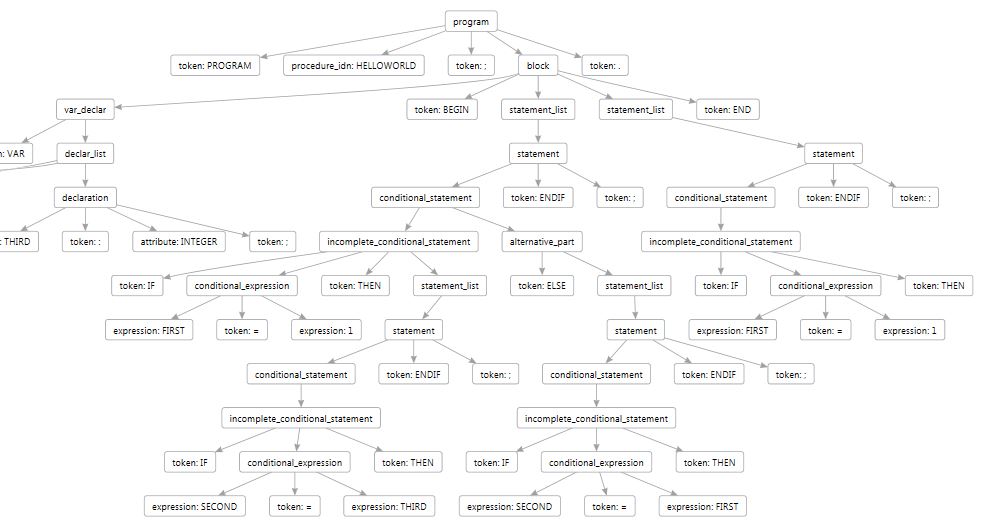
serializer.Serialize(xmlWriter, graph);

}

|  |  |  |  |
| --- | --- | --- | --- |
| **Loc** | **Op-code** | **AT** | **AF** |
| program | PROGRAM |  | F |
|  | [procedure\_idn] |  | F |
|  | ; |  | F |
|  | [block] |  | F |
|  | . | T | F |
| block | [var\_declar] |  | F |
|  | BEGIN |  | F |
|  | [statement\_list] |  | F |
|  | END | T | F |
| var\_declar | VAR |  | empty |
|  | [declar\_list] | T | F |
|  | empty | T | F |
| declar\_list | [declaration] |  | empty |
|  | [declar\_list] |  | F |
|  | empty | T | F |
| declaration | [var\_idn] |  | F |
|  | : |  | F |
|  | attribute |  | F |
|  | ; | T | F |
| attribute | INTEGER | T |  |
|  | FLOAT | T | F |
| var\_idn | [identifier] | T | F |
| procedure\_idn | [identifier] | T | F |
| Signal\_prog | [program] | OK | ERROR |

Приклади:





Test2:

PROGRAM HELLOWORLD; (\* valid comment \*)

VAR FIRST : INTEGER;

SECOND : FLOAT;

THIRD : INTEGER;

BEGIN

IF SECOND = FIRST THEN ENDIF;

IF 1 = FIRST THEN

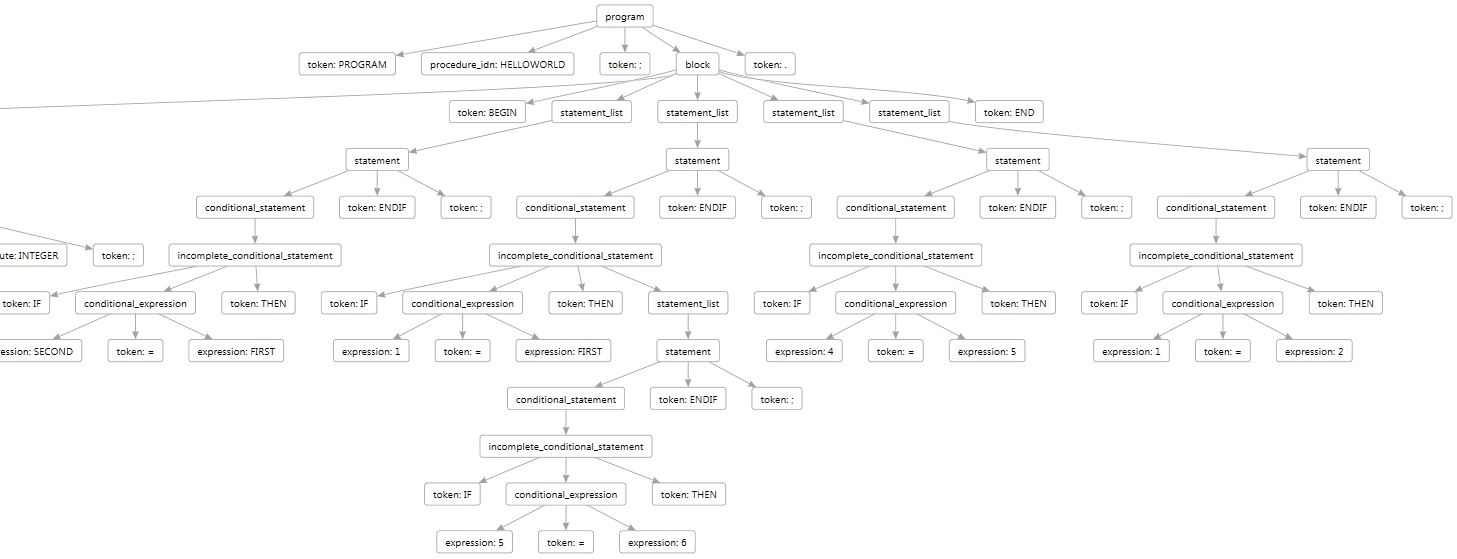
IF 5 = 6 THEN ENDIF;

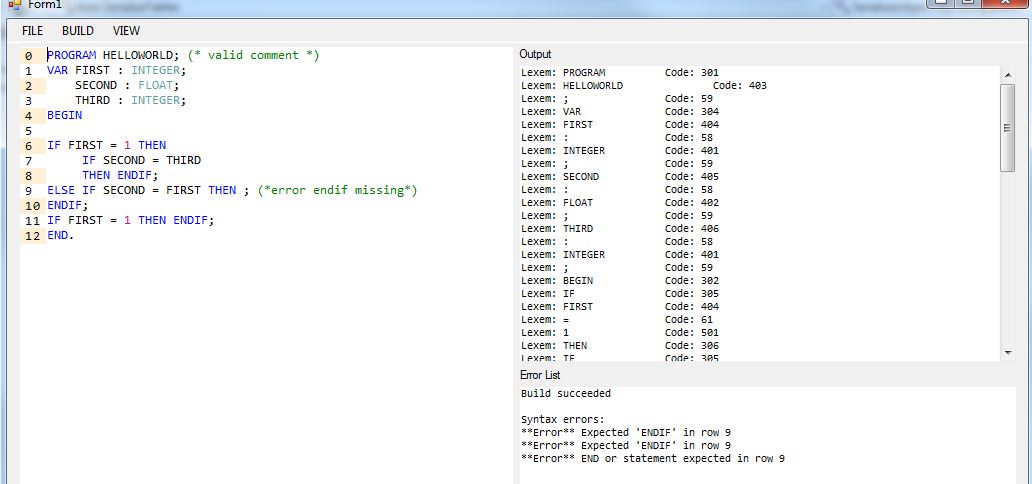
ENDIF;

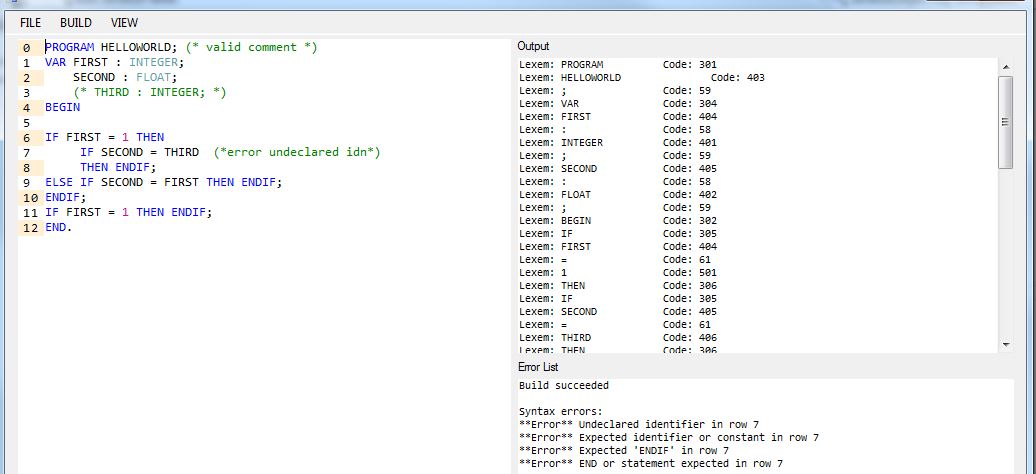
IF 4 = 5 THEN ENDIF;

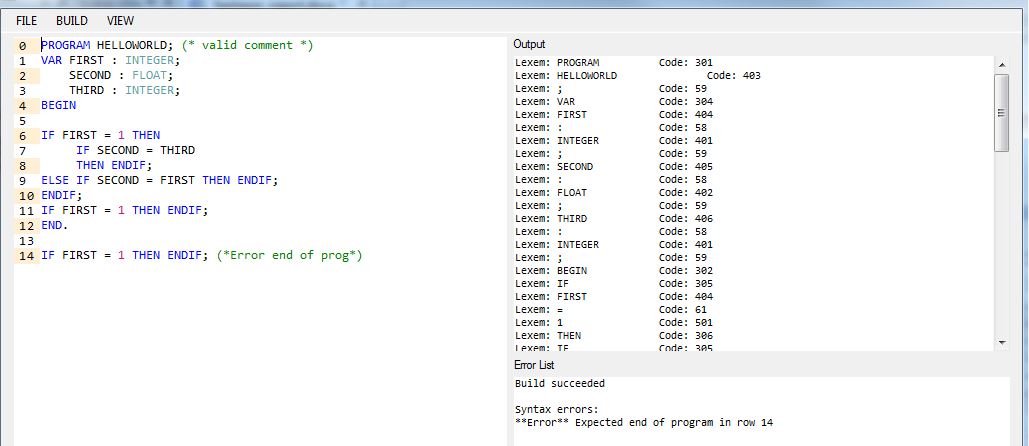
IF 1 = 2 THEN ENDIF;

END.









Приклад дерева розбору:

PROGRAM HELLOWORLD; (\* valid comment \*)

VAR FIRST : INTEGER;

BEGIN

IF FIRST = 1 THEN ENDIF;

END.

<?xml version="1.0" encoding="utf-8"?>

<XMLNode xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" Id="516" name="program" value="">

<nodes>

<XMLNode Id="517" name="token" value="PROGRAM">

<nodes />

</XMLNode>

<XMLNode Id="518" name="procedure\_idn" value="HELLOWORLD">

<nodes />

</XMLNode>

<XMLNode Id="519" name="token" value=";">

<nodes />

</XMLNode>

<XMLNode Id="520" name="block" value="">

<nodes>

<XMLNode Id="521" name="var\_declar" value="">

<nodes>

<XMLNode Id="522" name="token" value="VAR">

<nodes />

</XMLNode>

<XMLNode Id="523" name="declar\_list" value="">

<nodes>

<XMLNode Id="524" name="declaration" value="">

<nodes>

<XMLNode Id="525" name="var\_idn" value="FIRST">

<nodes />

</XMLNode>

<XMLNode Id="526" name="token" value=":">

<nodes />

</XMLNode>

<XMLNode Id="527" name="attribute" value="INTEGER">

<nodes />

</XMLNode>

<XMLNode Id="528" name="token" value=";">

<nodes />

</XMLNode>

</nodes>

</XMLNode>

</nodes>

</XMLNode>

</nodes>

</XMLNode>

<XMLNode Id="529" name="token" value="BEGIN">

<nodes />

</XMLNode>

<XMLNode Id="530" name="statement\_list" value="">

<nodes>

<XMLNode Id="531" name="statement" value="">

<nodes>

<XMLNode Id="532" name="conditional\_statement" value="">

<nodes>

<XMLNode Id="533" name="incomplete\_conditional\_statement" value="">

<nodes>

<XMLNode Id="534" name="token" value="IF">

<nodes />

</XMLNode>

<XMLNode Id="535" name="conditional\_expression" value="">

<nodes>

<XMLNode Id="536" name="expression" value="FIRST">

<nodes />

</XMLNode>

<XMLNode Id="537" name="token" value="=">

<nodes />

</XMLNode>

<XMLNode Id="538" name="expression" value="1">

<nodes />

</XMLNode>

</nodes>

</XMLNode>

<XMLNode Id="539" name="token" value="THEN">

<nodes />

</XMLNode>

</nodes>

</XMLNode>

</nodes>

</XMLNode>

<XMLNode Id="545" name="token" value="ENDIF">

<nodes />

</XMLNode>

<XMLNode Id="546" name="token" value=";">

<nodes />

</XMLNode>

</nodes>

</XMLNode>

</nodes>

</XMLNode>

<XMLNode Id="551" name="token" value="END">

<nodes />

</XMLNode>

</nodes>

</XMLNode>

<XMLNode Id="552" name="token" value=".">

<nodes />

</XMLNode>

</nodes>

</XMLNode>