**C#**

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

ФАКУЛЬТЕТ ІНФОРМАТИКИ І ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

КАФЕДРА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

**Лабораторна робота №5**

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

**Варіант 11**

Виконала:

студентка 2 курсу гр. ІВ-71

Молчанова В.С.

Перевірив:

Павлов В.Г.

Київ 2019 р.

**Тема:** Створення і настроювання низхідних синтаксичних аналізаторів на базі використання метамов Бекуса

**Мета**: Одержання навичок створення механізмів синтаксичного розбору методом рекурсивного спуска і створення обробника синтаксичних помилок вхідного тексту

**Варіант:** 

**Лістинг програми:**

public abstract class SyntacticAnalysis

{

public IGlossary CurrentGlossary { get; }

public List<string> Errors;

public Node Root;

public delegate Node Handler(Token token, Node lastNode);

public Dictionary<TokenType, Handler> TokenTypesHandlers;

protected SyntacticAnalysis(IGlossary currentGlossary)

{

CurrentGlossary = currentGlossary;

}

public Node Analyze(List<Token> tokens)

{

var rootNode = new Node("root", NodeType.Root, null);

var lastNode = rootNode;

for (var i = 0; i < tokens.Count; i++)

{

var token = tokens[i];

PreHandlerCheck(token, lastNode);

lastNode = TokenTypesHandlers[token.Type](token, lastNode);

lastNode.PreviousToken = i == 0 ? null : tokens[i - 1].Value;

lastNode.NextToken = i + 1 == tokens.Count ? null : tokens[i + 1].Value;

}

Errors.AddRange(GetErrors(rootNode));

return rootNode;

}

//All checks that cannot be implemented in handlers

protected abstract void PreHandlerCheck(Token token, Node lastNode);

#region Handlers

protected Node HandleOpeningBracket(Token token, Node lastNode)

{

Node newNode;

if (token.Value == "[")

{

newNode = HandleIndexer(token, lastNode);

}

else if (token.Value == "(" && lastNode.Type == NodeType.Name)

{

newNode = HandleFunctionCall(token, lastNode);

}

else

{

newNode = new Node(token.Value, NodeType.Brackets, lastNode);

}

newNode.AddError(ErrorType.BracketNotClosed);

return newNode;

}

protected Node HandleIndexer(Token token, Node lastNode)

{

var newNode = new Node(token.Value, NodeType.Indexer, lastNode);

if (!lastNode.CanBeRvalue ) newNode.AddError(ErrorType.ArrayNameNotFound);

return newNode;

}

protected Node HandleFunctionCall(Token token, Node lastNode)

{

var newNode = new Node(token.Value, NodeType.FunctionCall, lastNode);

return newNode;

}

protected Node HandleClosingBracket(Token token, Node lastNode)

{

var correspondingOpeningBracket = CurrentGlossary.Brackets.First(b => b.Contains(token.Value))[0].ToString();

var openingNode = lastNode.FindUp(correspondingOpeningBracket);

if (openingNode != null)

{

openingNode.ResolveError(ErrorType.BracketNotClosed);

if (openingNode.Type == NodeType.Brackets)

{

openingNode.Parent.RemoveChild(openingNode);

return openingNode.Parent;

}

return openingNode;

}

var newNode = new Node(token.Value, NodeType.Brackets, lastNode);

newNode.AddError(ErrorType.UnexpectedToken);

return newNode;

}

protected Node HandleNumericalConstant(Token token, Node lastNode)

{

return new Node(token.Value, NodeType.NumericalConstant, lastNode);

}

protected Node HandleName(Token token, Node lastNode)

{

return new Node(token.Value, NodeType.Name, lastNode);

}

protected Node HandleQuote(Token token, Node lastNode)

{

var openingNode = lastNode.FindUp(token.Value);

if (openingNode != null)

{

openingNode.ResolveError(ErrorType.QuoteNotClosed

);

openingNode.Value += token.Value;

openingNode.Left.Update(openingNode.Left.Value,

token.Value == "'" ? NodeType.CharLiteral : NodeType.StringLiteral);

return openingNode;

}

var newNode = new Node(token.Value, NodeType.Quotes, lastNode);

newNode.AddError(ErrorType.QuoteNotClosed);

return newNode;

}

protected Node HandleErrorLexeme(Token token, Node lastNode)

{

if (lastNode.Value == "\"") return new Node(token.Value, NodeType.StringLiteral, lastNode);

if (lastNode.Value == "'") return new Node(token.Value, NodeType.CharLiteral, lastNode);

return new Node(token.Value, NodeType.ErrorLexeme, lastNode);

}

protected Node HandleUnaryOperator(Token token, Node lastNode)

{

var newNode = new Node(token.Value, NodeType.UnaryOperator, lastNode);

return newNode;

}

protected Node HandleBinaryOperator(Token token, Node lastNode)

{

if (CurrentGlossary.AssignationOperators.Contains(token.Value))

return HandleAssignationOperator(token, lastNode);

var result = new Node(token.Value, NodeType.BinaryOperator, null);

result.AddError(ErrorType.BinaryOperatorMustTakeTwoOperands);

var lastPriority = CurrentGlossary.BinaryOperatorsByPriority.IndexOf(lastNode.Parent.Value);

var newPriority = CurrentGlossary.BinaryOperatorsByPriority.IndexOf(token.Value);

if (lastNode.Parent.Type != NodeType.Brackets && lastPriority != -1 && newPriority > lastPriority)

{

lastNode.Parent.Parent.InsertBeforeChild(lastNode.Parent, result);

}

else

{

lastNode.Parent.InsertBeforeChild(lastNode, result);

}

return result;

}

protected Node HandleAssignationOperator(Token token, Node lastNode)

{

lastNode = SkipUnaryToParent(lastNode);

var newNode = new Node(token.Value, NodeType.AssignationOperator, null);

lastNode.Parent.InsertBeforeChild(lastNode, newNode);

newNode.AddError(ErrorType.BinaryOperatorMustTakeTwoOperands);

if (!newNode.Left.CanBeLvalue) newNode.AddError(ErrorType.LeftSideOfAssignationOperator);

return newNode;

}

protected abstract Node HandleOperator(Token token, Node lastNode);

protected abstract Node HandlePunctuation(Token token, Node lastNode);

protected abstract Node HandleReservedWord(Token token, Node lastNode);

#endregion

protected Node SkipUnaryToChild(Node current)

{

while (CurrentGlossary.ConsiderAsUnary.Contains(current.Value) && current.Children.Any())

{

current = current.Children[0];

}

return current;

}

protected Node SkipUnaryToParent(Node current)

{

while (CurrentGlossary.ConsiderAsUnary.Contains(current.Parent.Value))

{

current = current.Parent;

}

return current;

}

protected Node FindBeforeAction(Node currentNode)

{

if (currentNode.Type == NodeType.Root) return currentNode;

while (currentNode.Type != NodeType.Root)

{

if (CurrentGlossary.BeforeActionMarkers.Contains(currentNode.Value) && currentNode.Right == null)

return currentNode;

currentNode = currentNode.Parent;

}

return null;

}

protected abstract bool IsStatement(Node node);

protected void Initialize()

{

TokenTypesHandlers = new Dictionary<TokenType, Handler>

{

{TokenType.ClosingBracket, HandleClosingBracket},

{TokenType.OpeningBracket, HandleOpeningBracket},

{TokenType.NumericalConstant, HandleNumericalConstant },

{TokenType.Name, HandleName },

{TokenType.Operator, HandleOperator },

{TokenType.Punctuation, HandlePunctuation },

{TokenType.ReservedWord,HandleReservedWord },

{TokenType.Quote, HandleQuote },

{TokenType.ErrorLexeme, HandleErrorLexeme }

};

}

public List<string> GetErrors(Node node)

{

var result = new List<string>();

if (node.Left != null) result.AddRange(GetErrors(node.Left));

if (node.Right != null) result.AddRange(GetErrors(node.Right));

foreach (var nodeError in node.Errors)

{

var firstRow = nodeError;

var secondRow = $"\tError token: '{node.Value}' ";

if (node.PreviousToken == null)

{

secondRow += $"before '{node.NextToken}'";

}

else if (node.NextToken == null)

{

secondRow += $"after '{node.PreviousToken}'";

}

else

{

secondRow += $"between '{node.PreviousToken}' and '{node.NextToken}'";

}

result.Add(firstRow + "\n" + secondRow);

}

return result;

}

}

public class PascalSyntacticAnalysis : SyntacticAnalysis

{

public new PascalGlossary CurrentGlossary;

public PascalSyntacticAnalysis(List<Token> tokens, PascalGlossary currentGlossary) : base(currentGlossary)

{

CurrentGlossary = currentGlossary;

Errors = new List<string>();

Initialize();

Root = Analyze(tokens);

}

public PascalSyntacticAnalysis(string expression, PascalGlossary currentGlossary) : base(currentGlossary)

{

CurrentGlossary = currentGlossary;

Errors = new List<string>();

Initialize();

var lexicalAnalysis = new LexicalAnalysis(expression, currentGlossary);

Root = Analyze(lexicalAnalysis.Result);

}

protected override void PreHandlerCheck(Token token, Node lastNode)

{

if (CurrentGlossary.BinaryOperatorsByPriority.Contains(lastNode.Value))

lastNode.ResolveError(ErrorType.BinaryOperatorMustTakeTwoOperands);

var forNode = lastNode.FindUp("for");

if (CurrentGlossary.AssignationOperators.Contains(token.Value))

{

forNode?.ResolveError(ErrorType.AssignationOfCycleVariableExpected);

}

else if (token.Value == "to")

{

forNode?.ResolveError(ErrorType.ToExpected);

}

}

#region OverridedHandlers

protected override Node HandleOperator(Token token, Node lastNode)

{

if (CurrentGlossary.UnaryOperators.Contains(token.Value) && !lastNode.CanBeRvalue) return HandleUnaryOperator(token, lastNode);

if (CurrentGlossary.BinaryOperatorsByPriority.Contains(token.Value)) return HandleBinaryOperator(token, lastNode);

var newNode = new Node(token.Value, NodeType.Empty, lastNode);

newNode.AddError(ErrorType.UnexpectedToken);

return newNode;

}

protected override Node HandlePunctuation(Token token, Node lastNode)

{

if (token.Value == ";") return HandleSemicolon(token, lastNode);

var newNode = new Node(token.Value, NodeType.Punctuation, lastNode);

newNode.AddError(ErrorType.UnexpectedToken);

return newNode;

}

protected override Node HandleReservedWord(Token token, Node lastNode)

{

if (CurrentGlossary.ConsiderAsUnary.Contains(token.Value)) return HandleUnaryOperator(token, lastNode);

switch (token.Value)

{

//Conditions

case "if": return HandleIf(token, lastNode);

case "then": return HandleThen(token, lastNode);

case "else": return HandleElse(token, lastNode);

//Blocks

case "begin": return HandleBegin(token, lastNode);

case "end": return HandleEnd(token, lastNode);

//Cycles

case "for": return HandleFor(token, lastNode);

case "to": return HandleToAndDownto(token, lastNode);

case "downto": return HandleToAndDownto(token, lastNode);

case "do": return HandleDo(token, lastNode);

case "while": return HandleWhile(token, lastNode);

}

return lastNode;

}

#endregion

protected Node HandleSemicolon(Token token, Node lastNode)

{

while (!lastNode.Value.Contains("end") && lastNode.Parent.Type != NodeType.Root && lastNode.Parent.Value != ";" &&

!CurrentGlossary.ReservedWords.Contains(lastNode.Parent.Value) &&

lastNode.Parent.Value != "begin")

{

lastNode = lastNode.Parent;

}

if (lastNode.Value.Contains("end"))

{

lastNode.Value += ";";

lastNode.ResolveError(ErrorType.SemicolonExpectedAfterToken);

return lastNode;

}

var newNode = new Node(token.Value, NodeType.Punctuation, null);

lastNode.Parent.InsertBeforeChild(lastNode, newNode);

return newNode;

}

protected Node HandleIf(Token token, Node lastNode)

{

var newNode = new Node(token.Value, NodeType.Condition, null);

var beforeActionNode = FindBeforeAction(lastNode);

if (beforeActionNode != null)

{

beforeActionNode.AddChild(newNode);

newNode.AddError(ErrorType.ThenExpected);

}

else

{

newNode.AddError(ErrorType.UnexpectedToken);

lastNode.AddChild(newNode);

}

return newNode;

}

protected Node HandleThen(Token token, Node lastNode)

{

var newNode = new Node(token.Value, NodeType.Condition, null);

var ifNode = lastNode.FindUp("if");

if (ifNode != null)

{

ifNode.AddChild(newNode);

ifNode.ResolveError(ErrorType.ThenExpected);

}

else

{

lastNode.AddChild(newNode);

newNode.AddError(ErrorType.UnexpectedToken);

}

return newNode;

}

protected Node HandleElse(Token token, Node lastNode)

{

var newNode = new Node(token.Value, NodeType.Condition, null);

var ifNode = lastNode.FindUp("if");

if (ifNode != null)

{

ifNode.Parent.InsertBeforeChild(ifNode, newNode);

var afterThen = ifNode.Right.Left;

if (afterThen?.Type == NodeType.BlockOperator)

{

if (!IsEndedBlockOperator(afterThen))

{

afterThen.ResolveError(ErrorType.SemicolonExpectedAfterToken);

}

else

{

newNode.AddError(ErrorType.UnexpectedToken);

}

}

}

else

{

lastNode.AddChild(newNode);

newNode.AddError(ErrorType.UnexpectedToken);

}

return newNode;

}

protected Node HandleBegin(Token token, Node lastNode)

{

var beforeActionNode = FindBeforeAction(lastNode);

var newNode = new Node(token.Value, NodeType.BlockOperator, beforeActionNode ?? lastNode);

newNode.AddError(ErrorType.BlockOperatorNotClosed);

return newNode;

}

protected Node HandleEnd(Token token, Node lastNode)

{

var beginNode = lastNode.FindUp("begin");

if (beginNode != null)

{

beginNode.ResolveError(ErrorType.BlockOperatorNotClosed);

beginNode.AddError(ErrorType.SemicolonExpectedAfterToken);

beginNode.Value += " " + token.Value;

return beginNode.Parent.Value == "if" ? beginNode.Parent : beginNode;

}

var newNode = new Node(token.Value, NodeType.BlockOperator, lastNode);

newNode.AddError(ErrorType.UnexpectedToken);

return newNode;

}

protected Node HandleFor(Token token, Node lastNode)

{

var newNode = new Node(token.Value, NodeType.Cycle, null);

var beforeActionNode = FindBeforeAction(lastNode);

if (beforeActionNode != null)

{

beforeActionNode.AddChild(newNode);

newNode.AddError(ErrorType.DoExpected);

newNode.AddError(ErrorType.AssignationOfCycleVariableExpected);

newNode.AddError(ErrorType.ToExpected);

}

else

{

newNode.AddError(ErrorType.UnexpectedToken);

lastNode.AddChild(newNode);

}

return newNode;

}

protected Node HandleToAndDownto(Token token, Node lastNode)

{

var newNode = new Node(token.Value, NodeType.Cycle, null);

var forNode = lastNode.FindUp("for");

if (forNode != null)

{

forNode.InsertBeforeChild(forNode.Left, newNode);

}

else

{

lastNode.AddChild(newNode);

newNode.AddError(ErrorType.UnexpectedToken);

}

return newNode;

}

protected Node HandleDo(Token token, Node lastNode)

{

var newNode = new Node(token.Value, NodeType.Cycle, null);

var cycleNode = lastNode.FindUp("for", "while");

if (cycleNode != null)

{

cycleNode.AddChild(newNode);

cycleNode.ResolveError(ErrorType.DoExpected);

}

else

{

lastNode.AddChild(newNode);

newNode.AddError(ErrorType.UnexpectedToken);

}

return newNode;

}

protected override bool IsStatement(Node node)

{

return node.Value == ";" || node.Type == NodeType.AssignationOperator || node.Type == NodeType.FunctionCall;

}

private bool IsEndedBlockOperator(Node node) => node.Type == NodeType.BlockOperator && node.Value.Contains(";");

}

**Контрольні приклади:**

**Приклад 1**

Entered expression: For a:= 1 to n begin for b:=1 to n do begin a:=m; end; end;

^

SYNTAX ERROR: DoExpected

**Приклад 2**

Entered expression: For a:= 1 to n do begin for b:=1 to n do begin a:=m; end; end;

No syntactic errors were found

root

/

\_\_\_\_\_\_\_for\_\_\_\_\_\_\_\_

/ \

to do

/ \ /

:= n begin end;

/ \ /

a 1 \_\_\_\_for\_\_\_\_\_

/ \

to do

/ \ /

:= n begin end;

/ \ /

b 1 ;

/

:=

/ \

a m

**Приклад 3**

Entered expression: if a > 5 then begin a:=5; end; else a:=4;

^

SYNTAX ERROR: UnexpectedToken