Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Методы трансляции

ОТЧЁТ

к лабораторной работе №3

на тему

**СИНТАКСИЧЕСКИЙ АНАЛИЗАТОР**

Выполнил: студент гр.253505 Таргонский Д.А.

Проверил: ассистент кафедры информатики Гриценко Н.Ю.

Минск 2025

**СОДЕРЖАНИЕ**

1 Формулировка задачи 3

2 Краткие теоретические сведения 4

3 Результат выполнения работы 5

Заключение 6

Список использованных источников 7

Приложение А (обязательное) Исходный код программы 8

1 ФОРМУЛИРОВКА ЗАДАЧИ

Целью данной лабораторной работы является освоение работы с существующими синтаксическими анализаторами. Разработка синтаксического анализатора подмножества языка программирования *PHP*

В качестве задачи необходимо разработать свой собственный синтаксический анализатор выбранного подмножества языка программирования. Построить синтаксическое дерево. Определить минимум 4 возможных синтаксических ошибки и показать их корректное выявление. Основной целью работы является написание сценариев, которые задают синтаксические правила для выбранного подмножества языка.

2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Синтаксический анализ – это один из ключевых этапов компиляции, который позволяет компилятору понять структуру исходного кода. Этот процесс включает в себя проверку правильности синтаксиса и построение синтаксического дерева, которое отражает иерархическую структуру программы [1].

Грамматика – это набор правил, которые определяют правильные комбинации токенов в языке программирования. Грамматики используются для проверки правильности синтаксиса и построения синтаксического дерева. Грамматики могут быть контекстно-свободными (CFG) или контекстно-зависимыми. В большинстве случаев используются контекстно-свободные грамматики, так как они проще для анализа.

Контекстно-свободные грамматики (CFG) позволяют описывать синтаксис языка программирования с помощью правил, которые не зависят от контекста, в котором они применяются. Это делает их удобными для автоматического анализа и генерации парсеров. Контекстно-зависимые грамматики, с другой стороны, могут учитывать контекст, что позволяет описывать более сложные синтаксические конструкции, но усложняет процесс анализа.

Синтаксическое дерево (или дерево разбора) – это иерархическая структура, которая отображает синтаксические отношения между токенами. Вершины дерева представляют синтаксические конструкции, а листья – токены.

Синтаксическое дерево позволяет компилятору понять структуру программы и выявить синтаксические ошибки. Оно также служит основой для дальнейших этапов компиляции, таких как семантический анализ и генерация кода. Синтаксическое дерево может быть использовано для оптимизации кода, так как оно позволяет выявить избыточные или неэффективные конструкции [1].

Синтаксический анализатор – это часть компилятора, которая отвечает за выявление и проверку синтаксических конструкций входного языка. Синтаксический анализатор получает строку токенов от лексического анализатора и проверяет, может ли эта строка токенов порождаться грамматикой входного языка. Ещё одной функцией синтаксического анализатора является генерация сообщений обо всех выявленных ошибках, причём достаточно внятных и полных, а кроме того, синтаксический анализатор должен уметь обрабатывать обычные, часто встречающиеся ошибки и продолжать работу с оставшейся частью программы. В случае корректной программы синтаксический анализатор строит дерево разбора и передаёт его следующей части компилятора для дальнейшей обработки [2]

3 РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ РАБОТЫ

В результате выполнения лабораторной работы был написан синтаксический анализатор подмножества языка программирования *PHP*.

На рисунке 3.1 представлен код с заведомо написанными синтаксическими ошибками.

Изображение выглядит как текст, Шрифт, снимок экрана, линия

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 3.1 – Код с заведомо написанными синтаксическими ошибками

На рисунке 3.2 изображен результат работы анализатора для программного кода.



Рисунок 3.2 – Результат работы синтаксического анализатора в виде вывода абстрактного синтаксического дерева

На рисунке 3.3 изображены синтаксические ошибки, найденные при синтаксическом анализе.

Изображение выглядит как текст, Шрифт, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 3.3 – Синтаксические ошибки, найденные при синтаксическом анализ

ЗАКЛЮЧЕНИЕ

В результате выполнения лабораторной работы был разработан собственный лексический анализатор языка программирования *PHP* на языке программирования *VB .NET*.

В ходе работы были изучены существующие синтаксические анализаторы, определены синтаксические правила для выбранного подмножества языка *PHP* и реализован синтаксический анализатор. В процессе реализации были определены синтаксические правила, создано синтаксическое дерево, позволяющее наглядно представить структуру кода, а также реализован механизм выявления ошибок. Было определено и продемонстрировано корректное обнаружение не менее четырех различных типов синтаксических ошибок, что подтверждает работоспособность анализатора.

Разработанный анализатор позволяет обрабатывать входные программы, проверять их соответствие заданным синтаксическим правилам и формировать структурированное представление кода. Полученные результаты демонстрируют возможность дальнейшего расширения функциональности анализатора, включая поддержку более сложных конструкций языка, оптимизацию обработки ошибок и интеграцию с другими инструментами анализа кода.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Синтаксический анализ: как компиляторы понимают структуру кода [Электронный ресурс]. – Режим доступа: https://sky.pro/wiki/javascript/sintaksicheskij-analiz-kak-kompilyatory-ponimayut-strukturu-koda

[2] Лабораторная работа №4. Синтаксический анализатор [Электронный ресурс]. – Режим доступа: https://csc.sibsutis.ru/sites/csc.sibsutis.ru/files/courses/trans/LabWork4.pdf

ПРИЛОЖЕНИЕ А

(обязательное)

**Исходный код программы**

' Класс узла синтаксического дерева

Public Class ASTNode

Public Property NodeType As String

Public Property Value As String

Public Property Children As New List(Of ASTNode)

Public Sub New(type As String, Optional value As String = "")

Me.NodeType = type

Me.Value = value

End Sub

Public Overrides Function ToString() As String

Return NodeType & ": " & Value

End Function

End Class

' Синтаксический анализатор

Public Class SyntacticAnalyzer

Private tokens As List(Of Token)

Private currentPosition As Integer = 0

Private errors As New List(Of String) ' Список для хранения ошибок

' Основной метод анализа – возвращает корневой узел синтаксического дерева

Public Function Analyze(tokens As List(Of Token)) As ASTNode

Console.WriteLine("SyntacticAnalyzer Version: 2025-04-23 v4")

Me.tokens = tokens

Me.currentPosition = 0

errors.Clear()

Dim root As ASTNode = ParseProgram()

Return root

End Function

' Получение списка ошибок

Public ReadOnly Property SyntaxErrors As List(Of String)

Get

Return errors

End Get

End Property

' Разбор программы

Private Function ParseProgram() As ASTNode

Dim node As New ASTNode("Program")

If Match("OPEN\_TAG") Then

node.Children.Add(New ASTNode("OPEN\_TAG", PreviousToken().Value))

ElseIf Not IsAtEnd() Then

AddError("Ожидался открывающий тег <?php в начале программы.")

node.Children.Add(New ASTNode("Error", "Ожидался открывающий тег <?php"))

End If

While Not IsAtEnd() AndAlso Peek().Type <> "CLOSE\_TAG"

Dim stmt As ASTNode = ParseStatement()

If stmt IsNot Nothing Then

node.Children.Add(stmt)

Else

Synchronize()

End If

End While

If Match("CLOSE\_TAG") Then

node.Children.Add(New ASTNode("CLOSE\_TAG", PreviousToken().Value))

End If

Return node

End Function

' Разбор оператора

Private Function ParseStatement() As ASTNode

If IsAtEnd() Then Return Nothing

Dim stmt As New ASTNode("Statement")

Try

If Check("KEYWORD") Then

Select Case Peek().Value

Case "for"

stmt.Children.Add(ParseForStatement())

Case "while"

stmt.Children.Add(ParseWhileStatement())

Case "do"

stmt.Children.Add(ParseDoWhileStatement())

Case "if"

stmt.Children.Add(ParseIfStatement())

Case "switch"

stmt.Children.Add(ParseSwitchStatement())

Case "function"

stmt.Children.Add(ParseFunctionDeclaration())

Case "echo"

stmt.Children.Add(ParseEchoStatement())

Case "else"

stmt.Children.Add(ParseElseStatement())

Case Else

AddError("Неизвестное ключевое слово: " & Peek().Value)

stmt.Children.Add(New ASTNode("Error", "Неизвестное ключевое слово: " & Peek().Value))

Advance()

End Select

ElseIf Check("IDENTIFIER") Then

If PeekNext() IsNot Nothing AndAlso PeekNext().Type = "OPERATOR" AndAlso PeekNext().Value = "=" Then

stmt.Children.Add(ParseAssignment())

Else

stmt.Children.Add(ParseExpressionStatement())

End If

Else

AddError("Ожидался оператор или выражение, найден: " & Peek().Value)

Advance()

Return Nothing

End If

Catch ex As Exception

AddError("Ошибка разбора оператора: " & ex.Message)

Synchronize()

Return Nothing

End Try

Return stmt

End Function

' Разбор echo

Private Function ParseEchoStatement() As ASTNode

Dim echoNode As New ASTNode("EchoStatement")

Advance() ' Пропускаем echo

Dim args As New ASTNode("Arguments")

If Not IsAtEnd() AndAlso Not (Check("DELIMITER") AndAlso Peek().Value = ";") Then

Dim expr As ASTNode = ParseExpression()

args.Children.Add(expr)

End If

echoNode.Children.Add(args)

If Not Match("DELIMITER") OrElse PreviousToken().Value <> ";" Then

AddError("Ожидался ';' после выражения echo.")

echoNode.Children.Add(New ASTNode("Error", "Ожидался ';' после выражения echo"))

End If

Return echoNode

End Function

' Разбор else

Private Function ParseElseStatement() As ASTNode

Dim elseNode As New ASTNode("Else")

Advance()

Dim body As ASTNode = ParseBlock()

elseNode.Children.Add(body)

Return elseNode

End Function

' Разбор выражения с точкой с запятой

Private Function ParseExpressionStatement() As ASTNode

Dim exprStmt As New ASTNode("ExpressionStatement")

If IsAtEnd() Then

AddError("Неожиданный конец ввода в выражении.")

exprStmt.Children.Add(New ASTNode("Error", "Неожиданный конец ввода"))

Return exprStmt

End If

Dim expr As ASTNode = ParseExpression()

exprStmt.Children.Add(expr)

If Not Match("DELIMITER") OrElse PreviousToken().Value <> ";" Then

AddError("Ожидался ';' после выражения.")

exprStmt.Children.Add(New ASTNode("Error", "Ожидался ';' после выражения"))

End If

Return exprStmt

End Function

' Разбор присваивания

Private Function ParseAssignment() As ASTNode

Dim assignNode As New ASTNode("Assignment")

Dim idToken As Token = Advance()

assignNode.Children.Add(New ASTNode("Identifier", idToken.Value))

If Match("OPERATOR") AndAlso PreviousToken().Value = "=" Then

If Not IsAtEnd() AndAlso Not (Check("DELIMITER") AndAlso Peek().Value = ";") Then

Dim expr As ASTNode = ParseExpression()

assignNode.Children.Add(expr)

If expr.NodeType = "Error" OrElse (expr.NodeType = "BinaryExpression" AndAlso expr.Children.Any(Function(c) c.NodeType = "Error")) Then

AddError("Ожидался ';' после присваивания.")

End If

Else

AddError("Ожидалось выражение после '='.")

assignNode.Children.Add(New ASTNode("Error", "Ожидалось выражение после '='"))

AddError("Ожидался ';' после присваивания.")

End If

Else

AddError("Ожидался '=' после идентификатора.")

assignNode.Children.Add(New ASTNode("Error", "Ожидался '=' после идентификатора"))

AddError("Ожидался ';' после присваивания.")

End If

If Not Match("DELIMITER") OrElse PreviousToken().Value <> ";" Then

AddError("Ожидался ';' после присваивания.")

End If

Return assignNode

End Function

' Разбор блока кода

Private Function ParseBlock() As ASTNode

Dim blockNode As New ASTNode("Block")

If Not Match("DELIMITER") OrElse PreviousToken().Value <> "{" Then

AddError("Ожидалась '{' для начала блока.")

blockNode.Children.Add(New ASTNode("Error", "Ожидалась '{' для начала блока"))

Return blockNode

End If

While Not IsAtEnd() AndAlso Not (Check("DELIMITER") AndAlso Peek().Value = "}")

Dim stmt As ASTNode = ParseStatement()

If stmt IsNot Nothing Then

blockNode.Children.Add(stmt)

Else

Synchronize()

If IsAtEnd() OrElse (Check("DELIMITER") AndAlso Peek().Value = "}") Then

Exit While

End If

End If

End While

If Not Match("DELIMITER") OrElse PreviousToken().Value <> "}" Then

AddError("Ожидалась '}' для завершения блока.")

blockNode.Children.Add(New ASTNode("Error", "Ожидалась '}' для завершения блока"))

End If

Return blockNode

End Function

' Разбор цикла for

Private Function ParseForStatement() As ASTNode

Dim forNode As New ASTNode("ForStatement")

Advance()

If Not Match("DELIMITER") OrElse PreviousToken().Value <> "(" Then

AddError("Ожидалась '(' после 'for'.")

forNode.Children.Add(New ASTNode("Error", "Ожидалась '(' после 'for'"))

Return forNode

End If

Dim init As ASTNode

If Not (Check("DELIMITER") AndAlso Peek().Value = ";") Then

init = ParseExpression()

Else

init = New ASTNode("Empty")

End If

Dim initNode As New ASTNode("Initialization")

initNode.Children.Add(init)

forNode.Children.Add(initNode)

If Not Match("DELIMITER") OrElse PreviousToken().Value <> ";" Then

AddError("Ожидался ';' после инициализации в 'for'.")

forNode.Children.Add(New ASTNode("Error", "Ожидался ';' после инициализации в 'for'"))

End If

Dim condition As ASTNode

If Not (Check("DELIMITER") AndAlso Peek().Value = ";") Then

condition = ParseExpression()

Else

condition = New ASTNode("Empty")

End If

Dim conditionNode As New ASTNode("Condition")

conditionNode.Children.Add(condition)

forNode.Children.Add(conditionNode)

If Not Match("DELIMITER") OrElse PreviousToken().Value <> ";" Then

AddError("Ожидался ';' после условия в 'for'.")

forNode.Children.Add(New ASTNode("Error", "Ожидался ';' после условия в 'for'"))

End If

Dim increment As ASTNode

If Not (Check("DELIMITER") AndAlso Peek().Value = ")") Then

Try

increment = ParseExpression()

If increment.NodeType = "Error" Then

forNode.Children.Add(New ASTNode("Error", "Некорректное выражение в инкременте 'for'"))

End If

Catch ex As Exception

AddError("Ошибка разбора инкремента в 'for': " & ex.Message)

increment = New ASTNode("Error", "Ошибка инкремента")

End Try

Else

increment = New ASTNode("Empty")

End If

Dim incrementNode As New ASTNode("Increment")

incrementNode.Children.Add(increment)

forNode.Children.Add(incrementNode)

If Not Match("DELIMITER") OrElse PreviousToken().Value <> ")" Then

AddError("Ожидалась ')' после 'for'.")

forNode.Children.Add(New ASTNode("Error", "Ожидалась ')' после 'for'"))

End If

Dim body As ASTNode = ParseBlock()

Dim bodyNode As New ASTNode("Body")

bodyNode.Children.Add(body)

forNode.Children.Add(bodyNode)

Return forNode

End Function

' Разбор цикла while

Private Function ParseWhileStatement() As ASTNode

Dim whileNode As New ASTNode("WhileStatement")

Advance()

If Not Match("DELIMITER") OrElse PreviousToken().Value <> "(" Then

AddError("Ожидалась '(' после 'while'.")

whileNode.Children.Add(New ASTNode("Error", "Ожидалась '(' после 'while'"))

Return whileNode

End If

Dim condition As ASTNode = ParseExpression()

Dim conditionNode As New ASTNode("Condition")

conditionNode.Children.Add(condition)

whileNode.Children.Add(conditionNode)

If Not Match("DELIMITER") OrElse PreviousToken().Value <> ")" Then

AddError("Ожидалась ')' после условия 'while'.")

whileNode.Children.Add(New ASTNode("Error", "Ожидалась ')' после условия 'while'"))

End If

Dim body As ASTNode = ParseBlock()

Dim bodyNode As New ASTNode("Body")

bodyNode.Children.Add(body)

whileNode.Children.Add(bodyNode)

Return whileNode

End Function

' Разбор цикла do-while

Private Function ParseDoWhileStatement() As ASTNode

Dim doWhileNode As New ASTNode("DoWhileStatement")

Advance()

Dim body As ASTNode = ParseBlock()

Dim bodyNode As New ASTNode("Body")

bodyNode.Children.Add(body)

doWhileNode.Children.Add(bodyNode)

If Not Match("KEYWORD") OrElse PreviousToken().Value <> "while" Then

AddError("Ожидалось 'while' после тела 'do'.")

doWhileNode.Children.Add(New ASTNode("Error", "Ожидалось 'while' после тела 'do'"))

Return doWhileNode

End If

If Not Match("DELIMITER") OrElse PreviousToken().Value <> "(" Then

AddError("Ожидалась '(' после 'while' в 'do-while'.")

doWhileNode.Children.Add(New ASTNode("Error", "Ожидалась '(' после 'while' в 'do-while'"))

Return doWhileNode

End If

Dim condition As ASTNode = ParseExpression()

Dim conditionNode As New ASTNode("Condition")

conditionNode.Children.Add(condition)

doWhileNode.Children.Add(conditionNode)

If Not Match("DELIMITER") OrElse PreviousToken().Value <> ")" Then

AddError("Ожидалась ')' после условия 'do-while'.")

doWhileNode.Children.Add(New ASTNode("Error", "Ожидалась ')' после условия 'do-while'"))

End If

If Not Match("DELIMITER") OrElse PreviousToken().Value <> ";" Then

AddError("Ожидался ';' после 'do-while'.")

doWhileNode.Children.Add(New ASTNode("Error", "Ожидался ';' после 'do-while'"))

End If

Return doWhileNode

End Function

' Разбор условного оператора if

Private Function ParseIfStatement() As ASTNode

Dim ifNode As New ASTNode("IfStatement")

Advance()

If Not Match("DELIMITER") OrElse PreviousToken().Value <> "(" Then

AddError("Ожидалась '(' после 'if'.")

ifNode.Children.Add(New ASTNode("Error", "Ожидалась '(' после 'if'"))

Return ifNode

End If

Dim condition As ASTNode = ParseExpression()

Dim conditionNode As New ASTNode("Condition")

conditionNode.Children.Add(condition)

ifNode.Children.Add(conditionNode)

If Not Match("DELIMITER") OrElse PreviousToken().Value <> ")" Then

AddError("Ожидалась ')' после условия 'if'.")

ifNode.Children.Add(New ASTNode("Error", "Ожидалась ')' после условия 'if'"))

End If

Dim thenBranch As ASTNode = ParseBlock()

Dim thenNode As New ASTNode("Then")

thenNode.Children.Add(thenBranch)

ifNode.Children.Add(thenNode)

If Check("KEYWORD") AndAlso Peek().Value = "else" Then

Dim elseStmt As ASTNode = ParseElseStatement()

ifNode.Children.Add(elseStmt)

End If

Return ifNode

End Function

' Разбор оператора switch

Private Function ParseSwitchStatement() As ASTNode

Dim switchNode As New ASTNode("SwitchStatement")

Advance()

If Not Match("DELIMITER") OrElse PreviousToken().Value <> "(" Then

AddError("Ожидалась '(' после 'switch'.")

switchNode.Children.Add(New ASTNode("Error", "Ожидалась '(' после 'switch'"))

Return switchNode

End If

Dim expression As ASTNode = ParseExpression()

Dim exprNode As New ASTNode("Expression")

exprNode.Children.Add(expression)

switchNode.Children.Add(exprNode)

If Not Match("DELIMITER") OrElse PreviousToken().Value <> ")" Then

AddError("Ожидалась ')' после выражения 'switch'.")

switchNode.Children.Add(New ASTNode("Error", "Ожидалась ')' после выражения 'switch'"))

End If

If Not Match("DELIMITER") OrElse PreviousToken().Value <> "{" Then

AddError("Ожидалась '{' после 'switch'.")

switchNode.Children.Add(New ASTNode("Error", "Ожидалась '{' после 'switch'"))

Return switchNode

End If

While Not IsAtEnd() AndAlso Not (Check("DELIMITER") AndAlso Peek().Value = "}")

If Check("KEYWORD") AndAlso Peek().Value = "case" Then

switchNode.Children.Add(ParseCase())

ElseIf Check("KEYWORD") AndAlso Peek().Value = "default" Then

switchNode.Children.Add(ParseDefault())

Else

AddError("Ожидался 'case' или 'default' в 'switch'.")

switchNode.Children.Add(New ASTNode("Error", "Ожидался 'case' или 'default' в 'switch'"))

Advance()

End If

End While

If Not Match("DELIMITER") OrElse PreviousToken().Value <> "}" Then

AddError("Ожидалась '}' для завершения 'switch'.")

switchNode.Children.Add(New ASTNode("Error", "Ожидалась '}' для завершения 'switch'"))

End If

Return switchNode

End Function

' Разбор case в switch

Private Function ParseCase() As ASTNode

Dim caseNode As New ASTNode("Case")

Advance()

Dim value As ASTNode = ParseExpression()

caseNode.Children.Add(value)

If Not Match("DELIMITER") OrElse PreviousToken().Value <> ":" Then

AddError("Ожидалось ':' после 'case'.")

caseNode.Children.Add(New ASTNode("Error", "Ожидалось ':' после 'case'"))

End If

While Not IsAtEnd() AndAlso Not (Check("KEYWORD") AndAlso (Peek().Value = "case" Or Peek().Value = "default")) AndAlso

Not (Check("DELIMITER") AndAlso Peek().Value = "}")

Dim stmt As ASTNode = ParseStatement()

If stmt IsNot Nothing Then

caseNode.Children.Add(stmt)

Else

Synchronize()

End If

End While

Return caseNode

End Function

' Разбор default в switch

Private Function ParseDefault() As ASTNode

Dim defaultNode As New ASTNode("Default")

Advance()

If Not Match("DELIMITER") OrElse PreviousToken().Value <> ":" Then

AddError("Ожидалось ':' после 'default'.")

defaultNode.Children.Add(New ASTNode("Error", "Ожидалось ':' после 'default'"))

End If

While Not IsAtEnd() AndAlso Not (Check("DELIMITER") AndAlso Peek().Value = "}")

Dim stmt As ASTNode = ParseStatement()

If stmt IsNot Nothing Then

defaultNode.Children.Add(stmt)

Else

Synchronize()

End If

End While

Return defaultNode

End Function

' Разбор объявления функции

Private Function ParseFunctionDeclaration() As ASTNode

Dim funcNode As New ASTNode("FunctionDeclaration")

Advance()

If Check("IDENTIFIER") Then

Dim nameToken As Token = Advance()

funcNode.Children.Add(New ASTNode("FunctionName", nameToken.Value))

Else

AddError("Ожидалось имя функции после 'function'.")

funcNode.Children.Add(New ASTNode("Error", "Ожидалось имя функции после 'function'"))

End If

If Not Match("DELIMITER") OrElse PreviousToken().Value <> "(" Then

AddError("Ожидалась '(' после имени функции.")

funcNode.Children.Add(New ASTNode("Error", "Ожидалась '(' после имени функции"))

End If

Dim params As New ASTNode("Parameters")

While Not IsAtEnd() AndAlso Not (Check("DELIMITER") AndAlso Peek().Value = ")")

If Check("IDENTIFIER") Then

Dim paramToken As Token = Advance()

params.Children.Add(New ASTNode("Parameter", paramToken.Value))

Else

AddError("Ожидался параметр функции.")

params.Children.Add(New ASTNode("Error", "Ожидался параметр функции"))

Advance()

End If

If Check("DELIMITER") AndAlso Peek().Value = "," Then

Advance()

End If

End While

funcNode.Children.Add(params)

If Not Match("DELIMITER") OrElse PreviousToken().Value <> ")" Then

AddError("Ожидалась ')' после параметров функции.")

funcNode.Children.Add(New ASTNode("Error", "Ожидалась ')' после параметров функции"))

End If

Dim body As ASTNode = ParseBlock()

Dim bodyNode As New ASTNode("Body")

bodyNode.Children.Add(body)

funcNode.Children.Add(bodyNode)

Return funcNode

End Function

' Разбор выражения

Private Function ParseExpression() As ASTNode

If IsAtEnd() Then

AddError("Неожиданный конец ввода в выражении.")

Return New ASTNode("Error", "Неожиданный конец ввода")

End If

Return ParseBinaryExpression(0)

End Function

' Рекурсивный разбор бинарных выражений

Private Function ParseBinaryExpression(minPrecedence As Integer) As ASTNode

Dim left As ASTNode = ParsePrimary()

While Not IsAtEnd() AndAlso CurrentTokenIsBinaryOperator(minPrecedence)

Dim opToken As Token = Advance()

Dim opPrecedence As Integer = GetOperatorPrecedence(opToken)

Dim right As ASTNode

If IsAtEnd() OrElse (Check("DELIMITER") AndAlso (Peek().Value = ";" Or Peek().Value = ")" Or Peek().Value = "}")) Then

AddError("Неполное бинарное выражение: " & opToken.Value)

right = New ASTNode("Error", "Неполное выражение")

Else

right = ParseBinaryExpression(opPrecedence + 1)

End If

Dim binNode As New ASTNode("BinaryExpression", opToken.Value)

binNode.Children.Add(left)

binNode.Children.Add(right)

left = binNode

End While

Return left

End Function

' Проверка бинарного оператора

Private Function CurrentTokenIsBinaryOperator(minPrecedence As Integer) As Boolean

If IsAtEnd() Then Return False

Dim token As Token = Peek()

If token.Type <> "OPERATOR" Then Return False

If token.Value = "++" OrElse token.Value = "--" Then Return False ' Исключаем унарные операторы

Dim precedence As Integer = GetOperatorPrecedence(token)

Return precedence >= minPrecedence

End Function

' Приоритет операторов

Private Function GetOperatorPrecedence(ByVal token As Token) As Integer

If token.Type <> "OPERATOR" Then Return -1

Select Case token.Value

Case "\*", "/"

Return 3

Case "+", "-"

Return 2

Case "==", "!=", "<=", ">=", "<", ">"

Return 1

Case Else

Return 0

End Select

End Function

' Разбор первичных выражений

Private Function ParsePrimary() As ASTNode

If IsAtEnd() Then

AddError("Неожиданный конец ввода в первичном выражении.")

Return New ASTNode("Error", "Конец ввода")

End If

Dim token As Token = Peek()

' Обработка префиксного инкремента (++$i)

If token.Type = "OPERATOR" AndAlso (token.Value = "++" OrElse token.Value = "--") Then

Advance()

Dim operand As ASTNode = ParsePrimary()

Dim unaryNode As New ASTNode("UnaryExpression", token.Value)

unaryNode.Children.Add(operand)

Return unaryNode

End If

token = Advance()

Select Case token.Type

Case "NUMBER", "STRING"

Return New ASTNode("Literal", token.Value)

Case "IDENTIFIER"

' Обработка постфиксного инкремента ($i++)

If Not IsAtEnd() AndAlso Peek().Type = "OPERATOR" AndAlso (Peek().Value = "++" OrElse Peek().Value = "--") Then

Dim opToken As Token = Advance()

Dim unaryNode As New ASTNode("UnaryExpression", opToken.Value)

unaryNode.Children.Add(New ASTNode("Identifier", token.Value))

Return unaryNode

End If

Return New ASTNode("Identifier", token.Value)

Case "DELIMITER"

If token.Value = "(" Then

Dim expr As ASTNode = ParseExpression()

If Match("DELIMITER") AndAlso PreviousToken().Value = ")" Then

Return expr

Else

AddError("Ожидалась ')' после группирующего выражения.")

expr.Children.Add(New ASTNode("Error", "Ожидалась ')' после группирующего выражения"))

Return expr

End If

Else

AddError("Неожиданный разделитель в выражении: " & token.Value)

Return New ASTNode("Error", "Неожиданный разделитель: " & token.Value)

End If

Case "OPERATOR"

AddError("Неожиданный оператор в выражении: " & token.Value)

Return New ASTNode("Error", "Неожиданный оператор: " & token.Value)

Case Else

AddError("Неожиданный токен в выражении: " & token.Value)

Return New ASTNode("Error", "Неожиданный токен: " & token.Value)

End Select

End Function

' Вспомогательные методы

Private Function Match(tokenType As String) As Boolean

If Check(tokenType) Then

Advance()

Return True

End If

Return False

End Function

Private Function Check(tokenType As String) As Boolean

If IsAtEnd() Then Return False

Return Peek().Type = tokenType

End Function

Private Function Advance() As Token

If Not IsAtEnd() Then currentPosition += 1

Return PreviousToken()

End Function

Private Function IsAtEnd() As Boolean

Return currentPosition >= tokens.Count

End Function

Private Function Peek() As Token

If IsAtEnd() Then Return Nothing

Return tokens(currentPosition)

End Function

Private Function PeekNext() As Token

If currentPosition + 1 < tokens.Count Then

Return tokens(currentPosition + 1)

End If

Return Nothing

End Function

Private Function PreviousToken() As Token

If currentPosition > 0 Then

Return tokens(currentPosition - 1)

End If

Return Nothing

End Function

Private Sub AddError(message As String)

If Not errors.Contains(message) Then

errors.Add(message)

Console.WriteLine("Синтаксическая ошибка: " & message)

End If

End Sub

' Метод синхронизации

Private Sub Synchronize()

While Not IsAtEnd()

Dim current As Token = Peek()

If current.Type = "DELIMITER" AndAlso (current.Value = ";" Or current.Value = "}" Or current.Value = ")") Then

Advance()

Exit Sub

End If

If current.Type = "KEYWORD" AndAlso IsStatementStarter(current.Value) Then

Exit Sub

End If

If current.Type = "IDENTIFIER" OrElse current.Type = "OPEN\_TAG" OrElse current.Type = "CLOSE\_TAG" Then

Exit Sub

End If

Advance()

End While

End Sub

Private Function IsStatementStarter(keyword As String) As Boolean

Select Case keyword

Case "for", "while", "do", "if", "switch", "function", "else", "echo"

Return True

Case Else

Return False

End Select

End Function

End Class