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 №2

на тему

**ЛЕКСИЧЕСКИЙ АНАЛИЗ**

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 7

Список использованных источников 8

Приложение А (обязательное) Исходный код программы 9

1 ФОРМУЛИРОВКА ЗАДАЧИ

Целью данной лабораторной работы является разработка лексического анализатора, который будет обрабатывать подмножество языка программирования *PHP*, определенное в первой лабораторной работе. Лексический анализатор — это важный компонент компилятора, который отвечает за разбиение исходного кода на токены. Эти токены представляют собой базовые элементы языка, которые затем могут быть использованы для дальнейшего анализа и компиляции.

В рамках работы требуется создать и вывести несколько таблиц, которые будут содержать различную информацию о токенах. В частности, необходимо сформировать таблицы идентификаторов, констант, ключевых слов, разделителей, а также логических и математических операторов. Идентификаторы представляют собой имена переменных и функций, константы — фиксированные в ЯП значения переменных, ключевые слова — зарезервированные слова языка, которые имеют специальное значение, разделители — символы, используемые для разделения элементов кода, а операторы обеспечивают выполнение вычислений и логических операций.

Кроме того, в процессе работы необходимо также обнаружить ошибки, связанные с определением неверных последовательностей символов. Это подразумевает, что анализатор должен уметь выявлять ситуации, когда код содержит недопустимые комбинации символов, и информировать пользователя о таких ошибках. Сообщение об ошибках важно для повышения качества программного кода и предотвращения возможных сбоев в его выполнении.

Лексический анализатор был реализован на языке *Visual* *Basic* .*Net*, что позволяет использовать возможности этого языка для создания графического интерфейса и обработки данных. *Visual* *Basic* .*Net* предоставляет удобные инструменты для работы с строками и массивами, что облегчает процесс разработки анализатора и его интеграцию с другими компонентами системы.

2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Лексический анализатор представляет собой первую фазу компилятора, его основная задача состоит в чтении входных символов исходной программы, их группировании в лексемы и вывод последовательностей токенов для всех лексем исходной программы.

Лексема – это структурная единица языка, которая состоит из элементарных символов языка и не содержит в своём составе других структурных единиц языка [1].

Поток токенов пересылается синтаксическому анализатору для разбора. Обычно при работе лексический анализатор взаимодействует также с таблицей символов. Когда лексический анализатор встречается с лексемой, составляющей идентификатор, эту лексему требуется внести в таблицу символов. В некоторых случаях лексический анализатор может получать из таблицы символов некоторую информацию об идентификаторах, которая может помочь ему верно определить передаваемый синтаксическому анализатору токен. Обычно взаимодействие реализуется как вызов лексического анализатора синтаксическим анализатором.

Этот вызов, представленный как команда *Analyze*, заставляет лексический анализатор читать символы из входного потока, пока он не сможет идентифицировать очередную лексему и вернуть синтаксическому анализатору корректный токен [2].

Поскольку лексический анализатор является частью компилятора, которая читает исходный текст, он может заодно выполнять и некоторые другие действия, помимо идентификации лексем. Одной из таких задач является отбрасывание комментариев и пробельных символов (пробел, символы табуляции и новой строки, а также, возможно, некоторые другие символы, использующиеся для отделения токенов друг от друга во входном потоке). Еще одной задачей является синхронизация сообщений об ошибках, генерируемых компилятором, с исходной программой. Например, лексический анализатор может отслеживать количество символов новой строки, чтобы каждое сообщение об ошибке сопровождалось номером строки, в которой она обнаружена. В некоторых компиляторах лексический анализатор создает копию исходной программы с сообщениями об ошибках, вставленными в соответствующие места исходного текста.

3 РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ РАБОТЫ

В ходе выполнения лабораторной работы был реализован лексический анализатор языка *PHP*. На рисунке 3.1 представлен анализируемый код. Также был реализован отлов различных лексических ошибок в коде.

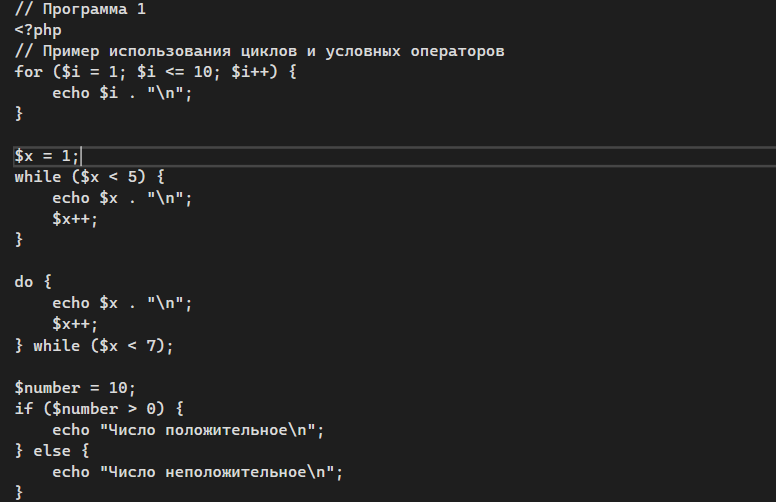


Рисунок 3.1 – Анализируемый код

На рисунке 3.2 представлен вывод таблицы идентификаторов.

Изображение выглядит как текст, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 3.2 – Вывод таблицы идентификаторов

На рисунке 3.3 представлен отлов ошибки неправильной переменной.

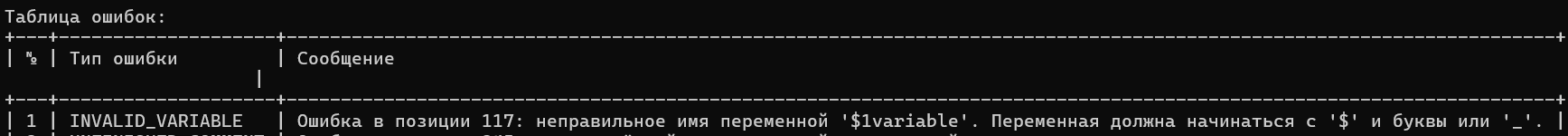


Рисунок 3.3 – Отлов ошибки неправильной переменной.

На рисунке 3.4 представлена отлов ошибки незакрытого многострочного комментария.



Рисунок 3.4 – Отлов ошибки незакрытого многострочного комментария.

.

На рисунке 3.5 представлена отлов ошибки неизвестного символа.



Рисунок 3.5 – Отлов ошибки незакрытого многострочного комментария.

На рисунке 3.6 представлена отлов ошибки незакрытой строки.



Рисунок 3.6 – Отлов ошибки незакрытой строки.

Также стоит заметить что данный лексический анализатор на *VB .NET* воспринимает все символы в незакрытых комментариях и строках считает за неопознанные символы.

Таким образом, был реализован лексический анализатор *PHP* кода на *VB .NET*.

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной лабораторной работы была осуществлена разработка лексического анализатора, который предназначен для обработки подмножества языка программирования *PHP*. Лексический анализатор — это компонент компилятора, который отвечает за преобразование входного текста (в данном случае, кода на *PHP*) в последовательность токенов, где каждый токен представляет собой элемент языка, такой как идентификаторы, ключевые слова, операторы и литералы.

В процессе разработки лексического анализатора особое внимание было уделено обработке различных конструкций языка и правил, определяющих допустимые последовательности символов. При тестировании анализатора были выявлены ошибки, связанные с определением неверных последовательностей символов. Это означает, что программа не всегда корректно распознавала, какие символы можно использовать в коде, и какие комбинации символов являются некорректными. Важно отметить, что выявление таких ошибок является ключевым этапом, поскольку они могут привести к сбоям в работе программы или неправильной интерпретации кода.

Кроме того, в лабораторной работе был продемонстрирован результат работы программы при обнаружении некорректных лексем. Некорректные лексемы — это те элементы кода, которые не соответствуют правилам синтаксиса языка, и их обработка важна для обеспечения надежности и устойчивости программного обеспечения. Результаты демонстрации показали, как анализатор реагирует на ошибки: какие сообщения об ошибках он выводит и как он обрабатывает некорректные входные данные. Это позволяет лучше понять, насколько эффективно и точно работает разработанный лексический анализатор.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Лексический анализатор [Электронный ресурс]. – Режим доступа: *https*://*habr*.*com*/*ru*/*articles*/515420/

[2] *Compilers* [Электронный ресурс]. – Режим доступа: *https*://*web*.*stanford*.*edu*/*class*/*cs*143/

ПРИЛОЖЕНИЕ А

(обязательное)

**Исходный код программы**

*Imports System.Text.RegularExpressions*

*Imports System.Collections.Generic*

*' Класс, представляющий токен (также используется для ошибок)*

*Public Class Token*

*Public Property Type As String ' Тип токена или ошибки*

*Public Property Value As String ' Текст лексемы или сообщение об ошибке*

*Public Property LexemeIndex As Integer ' Уникальный ID токена или ошибки*

*Public Overrides Function ToString() As String*

*Return $"<{Type}>({Value}) [ID = {LexemeIndex}]"*

*End Function*

*End Class*

*' Класс лексического анализатора*

*Public Class LexicalAnalyzer*

*Private ReadOnly tokenPatterns As New List(Of KeyValuePair(Of String, String)) From {*

*New KeyValuePair(Of String, String)("WHITESPACE", "^\s+"),*

*New KeyValuePair(Of String, String)("COMMENT", "^\/\/.\*|^\/\\*[\s\S]\*?\\*\/"),*

*New KeyValuePair(Of String, String)("OPEN\_TAG", "^\<\?php"),*

*New KeyValuePair(Of String, String)("CLOSE\_TAG", "^\?\>"),*

*New KeyValuePair(Of String, String)("IDENTIFIER", "^\$?[a-zA-Z\_]\w\*"),*

*New KeyValuePair(Of String, String)("NUMBER", "^\d+(\.\d+)?([eE][\+\-]?\d+)?"),*

*New KeyValuePair(Of String, String)("STRING", "^""([^""\\]\*(\\.[^""\\]\*)\*)""|^'([^'\\]\*(\\.[^'\\]\*)\*)'"),*

*New KeyValuePair(Of String, String)("OPERATOR", "^(==|!=|<=|>=|\+|\-|\\*|\/|=|<|>)"),*

*New KeyValuePair(Of String, String)("DELIMITER", "^[\(\)\{\}\[\],;:.]")*

*}*

*Private currentUniqueID As Integer = 1*

*Public Tokens As New List(Of Token)*

*Public ErrorTokens As New List(Of Token)*

*' Анализатор возвращает список распознанных токенов, а ошибки сохраняет в ErrorTokens*

*Public Function Analyze(ByVal input As String) As List(Of Token)*

*Dim position As Integer = 0*

*Dim errorCount As Integer = 0*

*While input.Length > 0*

*' --- Проверка 1: Неправильное имя переменной (например, "$1variable") ---*

*If input.StartsWith("$") AndAlso input.Length > 1 AndAlso Char.IsDigit(input, 1) Then*

*Dim mInvalid As Match = Regex.Match(input, "^\$\d\w\*")*

*Dim errorLexeme As String = mInvalid.Value*

*Dim errorMsg As String = $"Ошибка в позиции {position}: неправильное имя переменной '{errorLexeme}'. Переменная должна начинаться с '$' и буквы или '\_'."*

*Dim errorToken As New Token() With {*

*.Type = "INVALID\_VARIABLE",*

*.Value = errorMsg,*

*.LexemeIndex = currentUniqueID*

*}*

*currentUniqueID += 1*

*ErrorTokens.Add(errorToken)*

*errorCount += 1*

*input = input.Substring(mInvalid.Length)*

*position += mInvalid.Length*

*Continue While*

*End If*

*' --- Проверка 2: Незавершённый многострочный комментарий ---*

*If input.StartsWith("/\*") Then*

*Dim endIndex As Integer = input.IndexOf("\*/")*

*If endIndex = -1 Then*

*Dim errorMsg As String = $"Ошибка в позиции {position}: незавершённый многострочный комментарий."*

*Dim errorToken As New Token() With {*

*.Type = "UNFINISHED\_COMMENT",*

*.Value = errorMsg,*

*.LexemeIndex = currentUniqueID*

*}*

*currentUniqueID += 1*

*ErrorTokens.Add(errorToken)*

*errorCount += 1*

*input = input.Substring(2)*

*position += 2*

*Continue While*

*End If*

*End If*

*' --- Проверка 3: Незавершённый строковый литерал ---*

*If input.StartsWith("""") Then*

*Dim closingIndex As Integer = input.IndexOf("""", 1)*

*If closingIndex = -1 Then*

*Dim errorMsg As String = $"Ошибка в позиции {position}: незавершённый строковый литерал."*

*Dim errorToken As New Token() With {*

*.Type = "UNFINISHED\_STRING",*

*.Value = errorMsg,*

*.LexemeIndex = currentUniqueID*

*}*

*currentUniqueID += 1*

*ErrorTokens.Add(errorToken)*

*errorCount += 1*

*input = input.Substring(1)*

*position += 1*

*Continue While*

*End If*

*ElseIf input.StartsWith("'") Then*

*Dim closingIndex As Integer = input.IndexOf("'", 1)*

*If closingIndex = -1 Then*

*Dim errorMsg As String = $"Ошибка в позиции {position}: незавершённый строковый литерал."*

*Dim errorToken As New Token() With {*

*.Type = "UNFINISHED\_STRING",*

*.Value = errorMsg,*

*.LexemeIndex = currentUniqueID*

*}*

*currentUniqueID += 1*

*ErrorTokens.Add(errorToken)*

*errorCount += 1*

*input = input.Substring(1)*

*position += 1*

*Continue While*

*End If*

*End If*

*Dim matched As Boolean = False*

*For Each kvp In tokenPatterns*

*Dim pattern As String = kvp.Value*

*Dim regex As New Regex(pattern)*

*Dim m As Match = regex.Match(input)*

*If m.Success Then*

*Dim lexeme As String = m.Value*

*If kvp.Key <> "WHITESPACE" AndAlso kvp.Key <> "COMMENT" Then*

*Dim token As New Token() With {*

*.Type = kvp.Key,*

*.Value = lexeme,*

*.LexemeIndex = currentUniqueID*

*}*

*currentUniqueID += 1*

*Tokens.Add(token)*

*End If*

*input = input.Substring(lexeme.Length)*

*position += lexeme.Length*

*matched = True*

*Exit For*

*End If*

*Next*

*' --- Проверка 4: Неопознанный символ ---*

*If Not matched Then*

*Dim errorLexeme As String = input.Substring(0, 1)*

*Dim errorMsg As String = $"Ошибка в позиции {position}: неопознанный символ '{errorLexeme}'."*

*Dim errorToken As New Token() With {*

*.Type = "UNRECOGNIZED",*

*.Value = errorMsg,*

*.LexemeIndex = currentUniqueID*

*}*

*currentUniqueID += 1*

*ErrorTokens.Add(errorToken)*

*errorCount += 1*

*input = input.Substring(1)*

*position += 1*

*End If*

*End While*

*Return Tokens*

*End Function*

*End Class*