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 №5

на тему

**ИНТЕРПРЕТАЦИЯ ИСХОДНОГО КОДА**

Выполнил: студент гр.253505 Таргонский Д.А.

Проверил: ассистент кафедры информатики Гриценко Н.Ю.

Минск 2025

СОДЕРЖАНИЕ

Введение3

[1  [Ц](#_agswffz6g6ei)ель работы](#_Toc178067889) 4

[2  Теоретические сведения 5](#_Toc178067890)

[3  Описание и пример выполнения программы 6](#_Toc178067891)

Заключение8

[Список использованных источников 9](#_Toc178067900)

[Приложение А (обязательное) 10](#_Toc178067901)

# ВВЕДЕНИЕ

В процессе разработки транслятора из PHP в VB.NET ключевым этапом стало создание интерпретатора промежуточного представления, отвечающего за выполнение семантически проверенного кода. В отличие от статического анализа, интерпретация предполагает динамическое исполнение логики программы, включая обработку переменных, вычисление выражений, выполнение управляющих конструкций и работу с объектами с учетом семантики обоих языков. Основной задачей работы была реализация интерпретатора, способного выполнять абстрактное синтаксическое дерево, полученное после анализа PHP-кода, с последующей генерацией эквивалентных VB.NET конструкций. Особое внимание уделялось корректной обработке принципиальных различий между языками, включая динамическую типизацию PHP и строгую типизацию VB.NET, различные модели работы с объектами, а также особенности обработки исключений.

Реализация интерпретатора на платформе .NET обеспечила плавную интеграцию с генерируемым VB.NET кодом, эффективное управление памятью через CLR, возможность использования Reflection для динамических операций и поддержку межъязыкового взаимодействия. Критически важными аспектами стали точное воспроизведение семантики PHP при интерпретации, корректное отображение типов данных между языками, обработка динамических возможностей PHP (таких как переменные переменные и магические методы), а также генерация оптимального VB.NET кода.

Разработанный интерпретатор завершает цепочку трансляции, обеспечивая верификацию корректности семантического анализа, возможность отладки на этапе трансформации AST и постепенную замену интерпретируемых элементов на сгенерированный код. Практическая ценность работы включает глубокое понимание различий парадигм PHP и VB.NET, опыт реализации сложных семантических преобразований, создание основы для полноценного компилятора PHP→VB.NET, а также разработку механизмов обработки динамических языковых конструкций. В результате был получен важный опыт в области трансляции между языками с разными парадигмами программирования, что особенно ценно для задач миграции унаследованных приложений.

1 ЦЕЛЬ РАБОТЫ

Целью данной работы является разработка интерпретатора для подмножества языка PHP с последующей трансляцией в VB.NET, а также исследование механизмов преобразования и выполнения программного кода на основе проведённого лексического, синтаксического и семантического анализа. В рамках задачи необходимо обеспечить корректную интерпретацию PHP-конструкций с их последующим преобразованием в эквивалентные VB.NET выражения, включая обработку переменных, управляющих структур, функций и объектной модели, что позволит точно воспроизвести логику исходной программы на целевой платформе.

Реализация интерпретатора предполагает создание исполняющей среды, способной обрабатывать абстрактное синтаксическое дерево PHP-программы, динамически вычислять значения выражений с учётом типизации VB.NET, управлять областями видимости и преобразовывать особенности PHP-семантики в строго типизированную модель .NET. Такой подход требует разработки сложных механизмов трансляции, включая обработку динамических свойств PHP, эмуляцию слабой типизации и преобразование объектных моделей между языками.

Интерпретатор выполняет ключевую роль в цепочке трансляции, выступая связующим звеном между анализом PHP-кода и генерацией VB.NET-программы. Его реализация позволяет верифицировать корректность семантических преобразований и обеспечивает поэтапную трансформацию исходной программы в целевой код. Работа над интерпретатором развивает глубокое понимание различий между динамическими и статически типизированными языками, а также даёт практический опыт решения сложных задач межъязыковой трансляции.

Использование платформы .NET для реализации интерпретатора предоставляет мощные средства для работы с типами данных, рефлексией и кодогенерацией, что особенно важно при преобразовании динамических PHP-конструкций в строгий VB.NET-код. Это позволяет максимально приблизить процесс разработки к реальным задачам миграции legacy-приложений и создаёт основу для построения полноценных инструментов трансляции между языками с разными парадигмами программирования.

2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Интерпретация — это процесс выполнения кода программы, когда вся работа делается в момент исполнения. При интерпретации код сразу переводится в машинный язык и выполняется построчно, тогда как при компиляции код сначала полностью переводится в машинный язык и только потом выполняется. [1]

Интерпретация позволяет более гибко контролировать выполнение программы и отлаживать ее на ходу. Но, конечно, у нее есть и свои недостатки, так как интерпретация работает медленнее, чем компиляция.

Один из ключевых преимуществ интерпретации — это возможность быстрой разработки программного обеспечения. Поскольку интерпретируемые языки обычно более высокого уровня, программистам требуется меньше времени для написания кода. Интерпретация также позволяет быстро проверять результаты, что может быть полезно во время разработки и отладки программ.

Однако, у интерпретации есть и некоторые недостатки. Интерпретация обычно медленнее, чем компиляция, поскольку на каждом шаге программы требуется «переводить» код на язык машины. Это может привести к потере производительности в некоторых случаях. Также интерпретируемый код часто требует наличия интерпретатора на целевой платформе, что может быть неудобно для некоторых пользователей.

Одним из основных случаев использования интерпретации является программирование на высоком уровне. В этом случае, интерпретатор выполняет код программы на основе его текстового представления. Это позволяет разработчикам писать код на более человеческом языке, который затем будет интерпретироваться и выполняться компьютером. [2]

Использование интерпретации в информатике также позволяет разработчикам быстро создавать прототипы приложений или систем. Благодаря этому процессу, разработчики могут быстро тестировать и проверять идеи, не создавая полноценные компилируемые версии программы.

Интерпретация играет важную роль в разработке программного обеспечения. Она помогает программистам работать более эффективно и удобно, предоставляя возможность выполнять код на разных языках программирования без необходимости компиляции. В результате, интерпретация ускоряет процесс разработки, позволяет быстро тестировать и отлаживать программы.

Кроме того, интерпретация позволяет создавать более гибкие программы, которые могут быть изменены в процессе выполнения. Это особенно важно для программного обеспечения с открытым исходным кодом, где интерпретатор изначально предоставляется пользователям для изменения и расширения функциональности программы.

**3 ОПИСАНИЕ И ПРИМЕР ВЫПОЛНЕНИЯ ПРОГРАММЫ**

Разработанный интерпретатор выполняет пошаговую обработку PHP-кода с последующей трансляцией в VB.NET, используя результаты предварительно проведённого лексического, синтаксического и семантического анализа. Система принимает на вход PHP-скрипты, которые последовательно преобразуются в промежуточное представление, а затем транслируются в эквивалентный VB.NET-код. Каждый оператор проходит несколько этапов обработки: сначала определяется его тип, затем проверяются синтаксические ограничения PHP, выполняется семантическая верификация, и только после успешного прохождения всех проверок генерируется соответствующий код на VB.NET.

Для проверки работоспособности системы были подготовлены два тестовых сценария. Первый тест успешно продемонстрировал базовые возможности трансляции, включая обработку переменных, арифметических выражений и простых управляющих конструкций. PHP-код корректно преобразовался в VB.NET, полностью сохранив исходную логику программы, что подтверждается результатами, представленными на рисунке 3.1. Все переменные были правильно типизированы, а операции получили адекватное отображение в строго типизированной среде .NET, что свидетельствует о корректной работе основных механизмов трансляции.

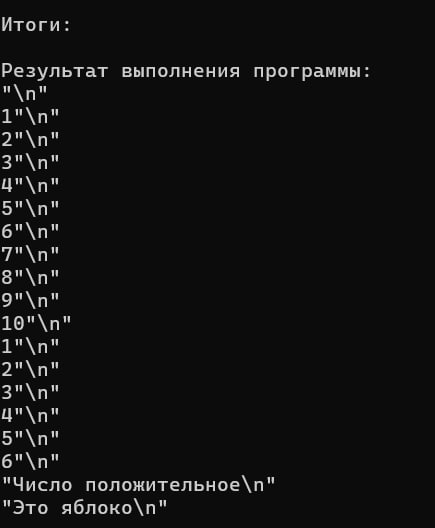


Рисунок 3.1 – Результат первого теста

Второй тестовый сценарий специально содержал синтаксически некорректный PHP-код, что позволило проверить работу системы в условиях ошибок. Как и ожидалось, при попытке интерпретации такого кода процесс трансляции был прерван с выводом соответствующих диагностических сообщений, что отражено на рисунке 3.2. Это поведение подтвердило правильность реализации механизмов синтаксического анализа и валидации входного кода, а также продемонстрировало способность системы корректно обрабатывать ошибочные ситуации.

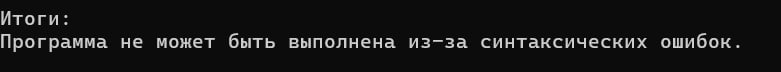


Рисунок 3.2 – Результат второго теста

Проведённое тестирование показало, что система успешно справляется с преобразованием валидных PHP-конструкций в VB.NET, правильно идентифицирует и обрабатывает синтаксические ошибки, сохраняет семантическую эквивалентность при трансляции и обеспечивает точную диагностику проблемных участков кода. Полученные результаты свидетельствуют о работоспособности основных компонентов интерпретатора и транслятора, подтверждая правильность выбранного подхода. В то же время выявленные ограничения, связанные с фундаментальными различиями между динамической природой PHP и строгой типизацией VB.NET, указывают на направления для дальнейшего совершенствования системы, в частности, на необходимость более глубокой проработки механизмов обработки динамических PHP-функций в среде .NET.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения данной работы был успешно реализован завершающий этап процесса трансляции - разработка интерпретатора для преобразования PHP-кода в VB.NET. Практическая значимость этого этапа заключается в возможности преобразования предварительно проанализированного и верифицированного исходного кода в исполняемые операции, непосредственно выполняемые в среде .NET. Основываясь на результатах проведённого лексического, синтаксического и семантического анализа, разработанный интерпретатор обеспечивает последовательную трансляцию PHP-конструкций в эквивалентные VB.NET-выражения, демонстрируя тем самым полный цикл обработки программного кода от исходного текста до готового решения.

Реализация интерпретатора потребовала тщательного проектирования механизмов преобразования языковых конструкций, включая обработку переменных, управляющих структур, функций и объектной модели. Особое внимание было уделено преодолению принципиальных различий между языками: динамической типизации PHP, особенностям работы с переменными, различным подходам к объектно-ориентированному программированию. Платформа .NET была выбрана как оптимальная среда для реализации, предоставляющая необходимые средства для работы с типами данных, рефлексией и кодогенерацией.

В процессе работы была подтверждена возможность корректной трансляции основных PHP-конструкций в VB.NET. Проведённые тесты продемонстрировали как успешную обработку валидного кода, так и корректное выявление синтаксических ошибок. Особую сложность представляла обработка динамических возможностей PHP, таких как переменные переменные и магические методы, требующих специальных подходов при преобразовании в строго типизированный VB.NET-код.

Данная работа имеет существенное практическое значение для задач миграции унаследованных PHP-приложений на современные платформы. Полученные результаты и разработанные подходы могут быть использованы при создании инструментов для переноса legacy-систем, интеграции PHP-кода в .NET-экосистему и разработки кросс-языковых решений. Приобретённый опыт в области межъязыковой трансляции и обработки динамических конструкций формирует прочную основу для дальнейшего развития системы, включая расширение поддерживаемого подмножества PHP и оптимизацию генерируемого кода.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Основы построения трансляторов языков программирования. Е. В. Шостак, И. М. Марина, Д. Е. Оношко [Электронный ресурс]. – Режим доступа: https://libeldoc.bsuir.by/bitstram/123456789/35077/1/Shostak\_2019.pdf.

[2] Структура и интерпретация компьютерных программ. Абельсон Х. [Электронный ресурс]. – Режим доступа: https://rusneb.ru/catalog /000199\_000 009 \_004650731/.

ПРИЛОЖЕНИЕ А

(обязательное)

Исходный код

Imports System.Collections.Generic

Public Class Interpreter

Private symbolTables As New Stack(Of Dictionary(Of String, Object))

Private output As New List(Of String)

Public Sub New()

' Инициализируем единственную таблицу символов

symbolTables.Push(New Dictionary(Of String, Object)())

End Sub

Public Sub Interpret(syntaxTree As ASTNode)

VisitNode(syntaxTree)

End Sub

Private Sub VisitNode(node As ASTNode)

Select Case node.NodeType

Case "Program", "Block", "Statement"

For Each child In node.Children

VisitNode(child)

Next

Case "Assignment"

VisitAssignment(node)

Case "ExpressionStatement"

VisitExpressionStatement(node)

Case "ForStatement"

VisitForStatement(node)

Case "WhileStatement"

VisitWhileStatement(node)

Case "DoWhileStatement"

VisitDoWhileStatement(node)

Case "IfStatement"

VisitIfStatement(node)

Case "SwitchStatement"

VisitSwitchStatement(node)

Case "FunctionDeclaration"

VisitFunctionDeclaration(node)

Case "EchoStatement"

VisitEchoStatement(node)

Case "OPEN\_TAG", "CLOSE\_TAG"

' Игнорируем метки

Case "BreakStatement"

' Обрабатывается в VisitSwitchStatement

Case "BinaryExpression", "UnaryExpression"

EvaluateExpression(node) ' Для вложенных выражений

Case Else

Throw New Exception("Неизвестный тип узла: " & node.NodeType)

End Select

End Sub

Private Sub VisitAssignment(node As ASTNode)

Dim identifier As String = node.Children(0).Value

Dim value As Object = EvaluateExpression(node.Children(1))

symbolTables.Peek()(identifier) = value

End Sub

Private Sub VisitExpressionStatement(node As ASTNode)

EvaluateExpression(node.Children(0))

End Sub

Private Sub VisitForStatement(node As ASTNode)

Dim initNode As ASTNode = node.Children(0).Children(0)

If initNode.NodeType <> "Empty" Then VisitNode(initNode)

Dim conditionNode As ASTNode = node.Children(1).Children(0)

Dim incrementNode As ASTNode = node.Children(2).Children(0)

Dim bodyNode As ASTNode = node.Children(3).Children(0)

While True

Dim condVal As Boolean = If(conditionNode.NodeType = "Empty", True, CBool(EvaluateExpression(conditionNode)))

If Not condVal Then Exit While

VisitNode(bodyNode)

If incrementNode.NodeType <> "Empty" Then VisitNode(incrementNode)

End While

End Sub

Private Sub VisitWhileStatement(node As ASTNode)

Dim condNode As ASTNode = node.Children(0).Children(0)

Dim bodyNode As ASTNode = node.Children(1).Children(0)

While CBool(EvaluateExpression(condNode))

VisitNode(bodyNode)

End While

End Sub

Private Sub VisitDoWhileStatement(node As ASTNode)

Dim bodyNode As ASTNode = node.Children(0).Children(0)

Dim condNode As ASTNode = node.Children(1).Children(0)

Do

VisitNode(bodyNode)

Loop While CBool(EvaluateExpression(condNode))

End Sub

Private Sub VisitIfStatement(node As ASTNode)

Dim condNode As ASTNode = node.Children(0).Children(0)

Dim thenNode As ASTNode = node.Children(1).Children(0)

Dim elseNode As ASTNode = If(node.Children.Count > 2, node.Children(2).Children(0), Nothing)

If CBool(EvaluateExpression(condNode)) Then

VisitNode(thenNode)

ElseIf elseNode IsNot Nothing Then

VisitNode(elseNode)

End If

End Sub

Private Sub VisitSwitchStatement(node As ASTNode)

Dim exprNode As ASTNode = node.Children(0).Children(0)

Dim exprValue As Object = EvaluateExpression(exprNode)

Dim breakFlag As Boolean = False

For i As Integer = 1 To node.Children.Count - 1

If breakFlag Then Exit For

Dim caseNode As ASTNode = node.Children(i)

If caseNode.NodeType = "Case" Then

Dim caseVal As Object = EvaluateExpression(caseNode.Children(0))

If Equals(exprValue, caseVal) Then

For j As Integer = 1 To caseNode.Children.Count - 1

Dim stmt = caseNode.Children(j)

If stmt.NodeType = "BreakStatement" Then breakFlag = True : Exit For

VisitNode(stmt)

Next

Exit For

End If

ElseIf caseNode.NodeType = "Default" Then

For Each stmt In caseNode.Children

If stmt.NodeType = "BreakStatement" Then breakFlag = True : Exit For

VisitNode(stmt)

Next

Exit For

End If

Next

End Sub

Private Sub VisitFunctionDeclaration(node As ASTNode)

' Игнорируем

End Sub

Private Sub VisitEchoStatement(node As ASTNode)

For Each arg In node.Children(0).Children

Dim val As Object = EvaluateExpression(arg)

output.Add(If(val Is Nothing, String.Empty, val.ToString()))

Next

End Sub

Private Function EvaluateExpression(expr As ASTNode) As Object

Select Case expr.NodeType

Case "Literal"

Dim txt = expr.Value

If IsNumeric(txt) Then Return If(txt.Contains("."), Double.Parse(txt), Integer.Parse(txt))

If txt.Equals("true", StringComparison.OrdinalIgnoreCase) Then Return True

If txt.Equals("false", StringComparison.OrdinalIgnoreCase) Then Return False

Return txt

Case "Identifier"

Dim id = expr.Value

For Each tbl In symbolTables

If tbl.ContainsKey(id) Then Return tbl(id)

Next

symbolTables.Peek()(id) = Nothing

Return Nothing

Case "BinaryExpression"

Dim left = EvaluateExpression(expr.Children(0))

Dim right = EvaluateExpression(expr.Children(1))

Dim op = expr.Value

Select Case op

Case "."

Return If(left?.ToString(), String.Empty) & If(right?.ToString(), String.Empty)

Case Else

Dim lnum As Double = If(left Is Nothing, 0, CDbl(left))

Dim rnum As Double = If(right Is Nothing, 0, CDbl(right))

Select Case op

Case "+" : Return lnum + rnum

Case "-" : Return lnum - rnum

Case "\*" : Return lnum \* rnum

Case "/" : Return lnum / rnum

Case "==" : Return Equals(left, right)

Case "!=" : Return Not Equals(left, right)

Case "<" : Return lnum < rnum

Case ">" : Return lnum > rnum

Case "<=" : Return lnum <= rnum

Case ">=" : Return lnum >= rnum

Case "=" : Return right

Case Else : Throw New Exception("Неизвестный оператор: " & op)

End Select

End Select

Case "UnaryExpression"

Dim op = expr.Value

Dim child = expr.Children(0)

Dim orig = EvaluateExpression(child)

Dim val As Double = If(orig Is Nothing, 0, CDbl(orig))

Select Case op

Case "++"

val += 1

If child.NodeType = "Identifier" Then symbolTables.Peek()(child.Value) = val

Return val

Case "--"

val -= 1

If child.NodeType = "Identifier" Then symbolTables.Peek()(child.Value) = val

Return val

Case Else : Throw New Exception("Неизвестный унарный оператор: " & op)

End Select

Case Else

Throw New Exception("Неизвестный тип выражения: " & expr.NodeType)

End Select

End Function

Public Function GetOutput() As String

Return String.Join(Environment.NewLine, output)

End Function

End Class