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 Снежко М.А.

Проверил: ассистент кафедры информатики

Гриценко Н.Ю.

Минск 2025

**СОДЕРЖАНИЕ**

[1 Цель работы 3](#_dq0nk8qy3txq)

[2 Ход работы 4](#_dmu1pedgw0t7)

[Заключение](#_p2r888fdfiws) 5

[Список использованных источников](#_l80j63tmhb4g) 6

[Приложение А(обязательное) Листинг программного кода](#_fsbz4tnpvuwn) 7

# 

# 1 ЦЕЛЬ РАБОТЫ

Целью данной лабораторной работы является разработка интерпретатора, который будет осуществлять преобразование кода на языке *Swift* в эквивалентный код на языке *F#*. Это включает в себя не только синтаксический и семантический анализ исходного кода, но и его интерпретацию для выполнения в среде *F#*[1].

В рамках работы необходимо:

1 Использовать существующие компоненты. На предыдущих этапах были реализованы лексический анализатор, синтаксический анализатор и семантический анализатор. Эти компоненты будут служить основой для интерпретации, обеспечивая корректное распознавание и анализ исходного кода Java.

2 Перевести синтаксическую структуру: В процессе работы будет разработан алгоритм, который обеспечит преобразование синтаксиса *Swift* в соответствующий синтаксис *F#*[2]. Это включает в себя такие конструкции, как классы, методы, условия и циклы.

Данная работа позволит углубить понимание механизмов работы языков программирования, а также развить навыки в области компиляторов и интерпретаторов.

# 2 ХОД РАБОТЫ

Во ходе выполнения лабораторной работы была выполнена интерпретация языка *F#*. Листинг программного кода приведен в приложении А. Пример кода на *F#* приведен рисунке 2.1.

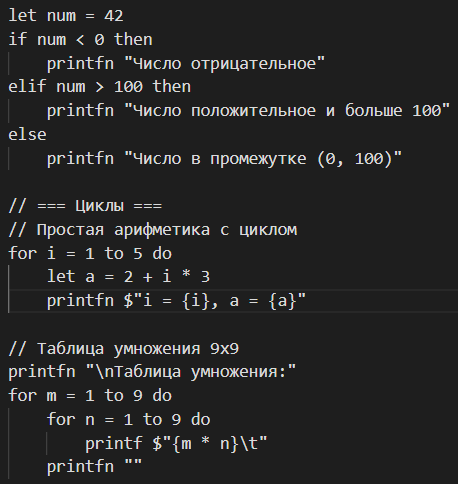


Рисунок 2.1 – Пример ввода на интерпретируемом языке

Результат работы интерпретатора представлен на рисунке 2.2.

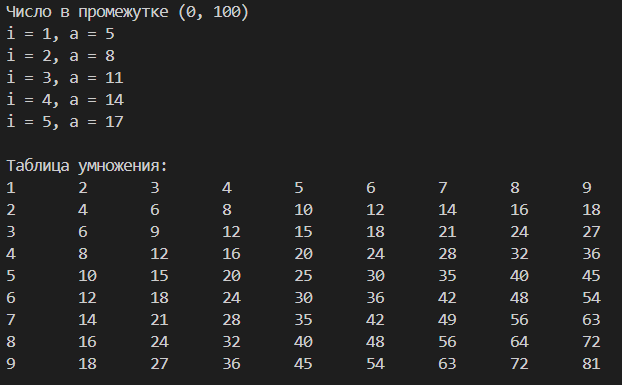


Рисунок 2.2 – Результат работы интерпретатора

Таким образом интерпретатор успешно справляется с генерацией и выполнением кода.

# ЗАКЛЮЧЕНИЕ

В процессе выполнения лабораторной работы по созданию интерпретатора были изучены ключевые аспекты интерпретации программного кода, включая построчное выполнение, динамический анализ и обработку ошибок. Были разработаны алгоритмы для разбора исходного кода, проверки синтаксической и семантической корректности, а также для выполнения операций и вызова функций. Особое внимание было уделено обработке ошибок, чтобы предоставлять пользователю понятные и информативные сообщения о проблемах в коде, что способствует более эффективной отладке и улучшению качества программного обеспечения.

# 

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Medium [Электронный ресурс]. – Режим доступа: https://medium.com/@bhagyesh.patil20/code-generation-in-compiler-design-93591d62efb6. – Дата доступа: 17.04.2025.

[2] GeeksForGeeks [Электронный ресурс]. – Режим доступа: https://www.geeksforgeeks.org/phases-of-a-compiler/. – Дата доступа: 17.04.2025.

# ПРИЛОЖЕНИЕ А

# (обязательное)

# Листинг программного кода

class CodeGenerator {

private var output = ""

private var indentLevel = 0

private var currentIndent: String {

return String(repeating: " ", count: indentLevel \* 4)

}

private let typeMap: [String: String] = [

"int": "Int",

"float": "Float",

"double": "Double",

"char": "Character",

"bool": "Bool",

"string": "String",

"unit": "Void",

"list": "Array",

"array": "Array",

"seq": "Array",

"Map": "Dictionary",

"Set": "Set",

]

func generate(from node: ASTNode) -> String {

output = ""

generateNode(node)

//return output

return cleanGeneratedCode(output)

}

private func generateNode(\_ node: ASTNode) {

switch node.type {

case "Program":

generateProgram(node)

case "FunctionDeclaration":

generateFunction(node)

case "VariableDeclaration":

generateVariableDeclaration(node)

case "IfStatement":

generateIfStatement(node)

case "WhileStatement":

generateWhileStatement(node)

case "ForInStatement":

generateForInStatement(node)

case "ForToStatement":

generateForToStatement(node)

case "FunctionCall":

generateFunctionCall(node)

case "BinaryOp":

generateBinaryOp(node)

case "Argument":

generateNode(node.children.first ?? node)

case "Assignment":

generateAssignment(node)

case "Return":

generateReturn(node)

case "Printfn":

generatePrint(node, newline: true)

case "Printf":

generatePrint(node, newline: false)

case "Class":

generateClass(node)

case "ThenBlock", "ElseBlock":

// Обрабатываем все дочерние узлы блока

indentLevel += 1

for child in node.children {

generateNode(child)

}

indentLevel -= 1

case "Condition":

generateCondition(node)

case "MemberDefinition":

generateMember(node)

case "Number", "Float\_number":

let value = node.value.replacingOccurrences(of: "L", with: "")

output += value

case "String\_literal":

var cleanedValue = node.value

if cleanedValue.hasPrefix("\"") && cleanedValue.hasSuffix("\"") {

cleanedValue = String(cleanedValue.dropFirst().dropLast())}

output += "\"\(cleanedValue)\""

case "Boolean\_literal":

output += node.value.lowercased()

case "Identifier":

output += node.value

case "ArrayAccess":

generateArrayAccess(node)

case "PropertyAccess":

generatePropertyAccess(node)

case "ArrayOrList":

generateArrayOrList(node)

case "StartValue", "EndValue", "Range":

for child in node.children {

generateNode(child)

}

case "Type":

// Тип обычно обрабатывается в родительском узле

break

case "Value":

output += node.value

case "Body":

// Обрабатываем все дочерние узлы тела

indentLevel += 1

for child in node.children {

generateNode(child)

if !child.type.hasPrefix("If") && !child.type.hasPrefix("For")

&& !child.type.hasPrefix("While") && child.type != "Body"

{

output += "\n"}}

indentLevel -= 1

case "SetDeclaration":

generateSetDeclaration(node)

case "SequenceDeclaration":

generateSequenceDeclaration(node)

case "ListDeclaration":

generateListDeclaration(node)

case "MapDeclaration":

generateMapDeclaration(node)

case "ArrayDeclaration":

generateArrayDeclaration(node)

case "KeyValuePair":

generateKeyValuePair(node)

default:

print("Warning: Unknown node type \(node.type)")

for child in node.children {

generateNode(child)

}

}

}