Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

ОТЧЕТ

к лабораторной работе №2

на тему

**ЛЕКСИЧЕСКИЙ АНАЛИЗ**

Выполнил: студент гр. 253505

Снежко М. А.

Проверил: ассистент кафедры

информатики Гриценко Н. Ю.

Минск 2025

СОДЕРЖАНИЕ

[1 Постановка задачи](#_Toc188619154) 3

[2 Описание работы программы 4](#_Toc188619155)

[3 Ход выполнения программы 5](#_Toc188619156)

[Заключение 6](#_Toc188619157)

[Список литературных источников 7](#_Toc188619158)

[Приложение А](#_Toc188619159) [(обязательное) Листинг программного кода 8](#_Toc188619160)

1. постановка задачи

Цель данной лабораторной работы заключается в разработке лексического анализатора на *Swift* для подмножества языка программирования *F#*, с использованием которого будет производиться разбор исходных текстов программ [1].

Лексический анализатор должен читать поток символов, представляющих программу, и выделять из него значащие последовательности, называемые лексемами. Эти лексемы могут включать идентификаторы, операторы, константы, знаки математических операций, скобки и другие элементы, характерные для базового синтаксиса языка. Основной задачей является правильное определение лексем, их категоризация и преобразование в понятную структуру, которая может быть использована для дальнейшего синтаксического анализа.

Лексический анализатор должен работать с входными данными, представленными в виде текстового файла, содержащего программу, написанную на заданном подмножестве языка.

В рамках работы нужно создать таблицы констант, ключевых слов, разделителей, логических/математических операторов. Эти таблицы должны быть использованы для поиска ранее встреченных лексем, что позволит избежать дублирования данных. Для констант таблица должна включать информацию о типе (например, переменная с плавающей точкой). При каждом новом обращении к идентификатору нужно проверять его наличие в таблице.

Также необходимо предусмотреть механизм обработки лексических ошибок, которые могут возникать при неверных последовательностях символов, таких как неправильно сформированные идентификаторы, неверные операторы или некорректные константы. Лексический анализатор должен быть способен выявить такие ошибки и сообщить о них, предоставляя пользователю подробную информацию о месте и типе ошибки.

1. описание работы программы

В ходе выполнения лабораторной работы был разработан лексический анализатор для подмножества языка программирования *F#*, предназначенного для работы с выражениями, включающими арифметические операции, скобки, константы и идентификаторы [2]. Задача заключалась в том, чтобы разобрать исходный код, выделить лексемы и классифицировать их по типам: ключевые слова, операторы, разделители, константы, а также выполнить проверки на некоторые лексические ошибки.

Первоначально был создан набор ключевых слов, операторов и разделителей, который позволил корректно идентифицировать основные элементы языка. Ключевые слова, такие как «*if*», «*while*», «*for*», и операторы вроде «+», «-», «\*», а также разделители, такие как «;», «,», «(», «)», были занесены в соответствующие списки. Эти элементы впоследствии использовались для того, чтобы на стадии лексического анализа определять, к какому типу принадлежит каждое встреченное слово.

Для проверки значений, встречающихся в программе, был реализован механизм, который проверяет, был ли такой элемент уже занесен в таблицу. Каждое новая константа или оператор, встреченный в коде, проверялись на наличие в таблице. Если элемент не был найден, то добавлялся в таблицу с уникальным индексом.

После реализации лексического анализа и обработки таблиц была добавлена функция обработки ошибок. В частности, было предусмотрено распознавание некорректных шаблонов, например, неверных последовательностей символов Программы анализировали некорректные операторы инкремента/декремента и ошибки в идентификаторах.

Для удобства отслеживания работы программы был реализован механизм логирования. Все действия, включая создание токенов, обработку ошибок и запись результатов, фиксировались в файле *log.txt*. Кроме того, результаты лексического анализа записывались в файл *output.txt*, а также выводились в виде таблицы в файл *outputTable.txt*

В результате выполнения лабораторной работы был получен лексический анализатор, который выполняет разбор исходных программ, классифицирует лексемы, заполняет таблицы, а также выявляет и сообщает о лексических ошибках в исходном коде [3].

1. ход выполнения программы

На рисунке 3.1 представлен вывод таблицы идентификаторов в результате выполнения программы.

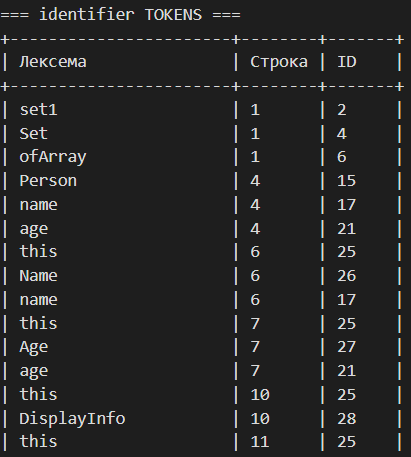


Рисунок 3.1 – Таблица идентификаторов

На рисунке 3.2 показана часть вывода таблицы с ключевыми словами.

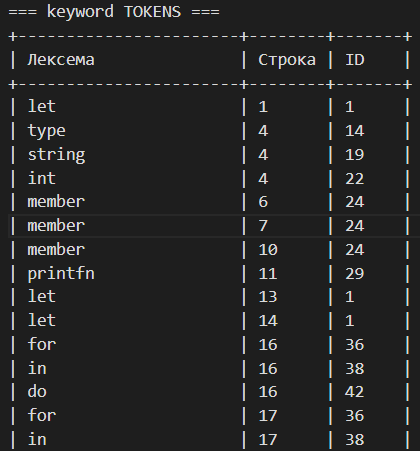


Рисунок 3.2 – Таблица ключевых слов

Пример обработки ошибок представлен на рисунке 3.3.

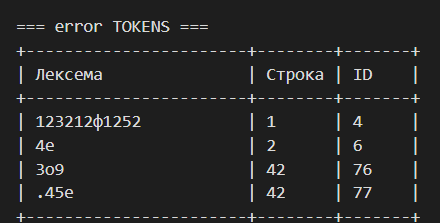


Рисунок 3.3 – Таблица ошибок

Заключение

В ходе выполнения лабораторной работы был разработан лексический анализатор на языке *Swift* для подмножества языка программирования *F#*. Разработанный анализатор выполняет разбор входного кода, выделяет ключевые лексемы, включая ключевые слова, операторы, разделители и константы, а также формирует соответствующие таблицы для их хранения и обработки.

Для организации хранения данных были созданы таблицы констант, ключевых слов, операторов и разделителей, что позволило минимизировать дублирование информации и упростить обработку лексем. Также реализован механизм проверки наличия элементов в таблице перед их добавлением, что способствует оптимизации работы анализатора.

Дополнительно были реализованы модули для обработки строковых и символьных констант с использованием регулярных выражений, что позволило корректно обрабатывать текстовые значения и добавлять их в таблицу констант

Реализованный алгоритм анализа позволил не только классифицировать элементы кода, но и выявлять лексические ошибки. В частности, были обработаны такие ошибки, как отсутствие закрывающих кавычек в строковых литералах, некорректные логические операторы, ошибки в именах идентификаторов и неверные шаблоны операторов инкремента. Анализатор предоставляет детализированные сообщения об ошибках, указывая их местоположение в коде, что упрощает процесс отладки.

Кроме того, в ходе разработки анализатора была предусмотрена возможность расширения функционала. Это включает в себя добавление новых типов лексем и поддержку более сложных синтаксических конструкций. Функциональность анализатора может быть улучшена за счет интеграции с системой автоматического тестирования, что позволит проверять его работу на различных тестовых случаях и обеспечивать высокое качество кода.

Таким образом, разработанный лексический анализатор представляет собой мощный инструмент для работы с кодом на языке *F#*, обеспечивая высокую точность и надежность при анализе. Его возможности по выявлению ошибок и адаптивность к изменениям в требованиях делают его ценным дополнением к инструментам разработчика. В дальнейшем планируется продолжить работу над улучшением функционала и производительности анализатора, что позволит расширить его применение в различных областях программирования.

Список литературных источников

1. Swift "Swift documentation" – [Электронный ресурс]. – Режим доступа: https://www.swift.org/documentation/.
2. F# "Документация по F#" – [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/dotnet/fsharp/.
3. GURU99 "Лексический анализ" – [Электронный ресурс]. – Режим доступа: https://www.guru99.com/ru/compiler-design-lexical-analysis.html.

ПРИЛОЖЕНИЕ А

(обязательное)  
Листинг программного кода

// Типы токенов

enum TokenType: String {

case keyword = "KEYWORD"

case identifier = "IDENTIFIER"

case number = "NUMBER"

case floatNumber = "FLOAT\_NUMBER"

case stringLiteral = "STRING\_LITERAL"

case charLiteral = "CHAR\_LITERAL"

case operatorToken = "OPERATOR"

case attribute = "ATTRIBUTE"

case error = "ERROR"

}

// Структура токена

struct Token {

let type: TokenType

let value: String

let line: Int

let id: Int

}

// Ключевые слова F#

let fsharpKeywords: Set<String> = [

"let", "rec", "fun", "match", "with", "if", "then", "else", "elif", "for", "to", "do", "while",

"type", "module", "namespace", "open", "exception", "try", "finally", "raise", "begin", "end",

"in", "of", "when", "as", "val", "mutable", "lazy", "async", "yield", "return", "use", "new",

"interface", "inherit", "abstract", "default", "member", "static", "override", "private",

"public", "internal", "base", "null", "true", "false", "and", "or", "not", "upcast", "downcast",

"int", "int32", "int64", "float", "double", "decimal", "bool", "string", "char", "unit", "obj",

"byte", "sbyte", "int16", "uint16", "uint", "float32", "single", "printfn", "downto", "printf",

"class"

]

// Операторы F#

let fsharpOperators: Set<String> = [

"+", "-", "\*", "/", "%", "=", "<", ">", "<=", ">=", "<>", "&&", "||", "!", "|>", ">>", "<<",

"::", "@", "^", "~", "?", ":", "->", "<-", "|", "&", ";;", "(", ")", "[", "]", "{", "}", ",",

".", "..", ";", "\*\*", "not", "&&&", "|||", "^^^", "~~~", "<<<", ">>>", "$",

]

class Lexer {

private let source: String

private var pos: String.Index

private var line: Int

private var nextTokenId: Int

private var tokens: [Token]

private var tokenIds: [String: Int]

// Основная функция для токенизации

func tokenize() -> [Token] {

tokens = []

while pos < source.endIndex {

let currentChar = source[pos]

Logger.log("Текущая позиция: \(source.distance(from: source.startIndex, to: pos))")

Logger.log(

"Текущий символ: \(source[pos]) (\(source[pos].unicodeScalars.first!.value))")

if currentChar.isWhitespace {

if currentChar == "\r\n" {

line += 1

}

pos = source.index(after: pos)

} else if currentChar == "["

&& (source.index(after: pos) < source.endIndex

&& (source[source.index(after: pos)] == "<"))

{

tokens.append(consumeAttribute())

} else if currentChar.isLetter || currentChar == "\_" {

tokens.append(consumeIdentifierOrKeyword())

} else if currentChar.isNumber

|| (currentChar == "." && source.index(after: pos) < source.endIndex

&& source[source.index(after: pos)].isNumber)

{

tokens.append(consumeNumber())

} else if currentChar == "\"" {

tokens.append(consumeStringLiteral())

} else if currentChar == "'" {

tokens.append(consumeCharLiteral())

} else if (currentChar == "/"

&& (source.index(after: pos) < source.endIndex

&& (source[source.index(after: pos)] == "/"

|| source[source.index(after: pos)] == "\*")))

|| (currentChar == "("

&& (source.index(after: pos) < source.endIndex

&& source[source.index(after: pos)] == "\*"))

{

consumeComment()

if source[pos] == ")" {

pos = source.index(after: pos)

}

} else if isOperator(String(currentChar)) {

tokens.append(consumeOperator())

} else {

tokens.append(createToken(type: .error, value: String(currentChar)))

pos = source.index(after: pos)

}

}

return tokens

}

func main() {

} else {

// for token in tokens {

// let outputLine =

// "Токен: \(token.type.rawValue) Лексема: \"\(token.value)\" Строка: \(token.line) Id: \(token.id)"

// Logger.log(outputLine)

// print(outputLine)

// }

// Группируем токены по их типу

let groupedTokens = Dictionary(grouping: tokens, by: { $0.type })

// Открываем файл для записи

if let outputFile = FileHandle(forWritingAtPath: outputFileNameTable) {

for (type, tokens) in groupedTokens {

let header = "\n=== \(type) TOKENS ===\n"

outputFile.write(header.data(using: .utf8)!)

let tableHeader = """

+-----------------------+--------+-------+

| Лексема | Строка | ID |

+-----------------------+--------+-------+

"""

Logger.log(tableHeader)

outputFile.write(tableHeader.data(using: .utf8)!)

outputFile.write("\n".data(using: .utf8)!)

for token in tokens {

let lexeme = token.value

let lines = stride(from: 0, to: lexeme.count, by: 21).map {

String(lexeme.dropFirst($0).prefix(21))

}

for (index, part) in lines.enumerated() {

let formattedLexeme = part.padding(toLength: 21, withPad: " ", startingAt: 0)

let row: String

if index == 0 {

// Первая строка с номером строки и ID

row = String(format: "| %@ | %-6d | %-5d |\n", formattedLexeme, token.line, token.id)

} else {

// Последующие строки без номера строки и ID

row = String(format: "| %@ | | |\n", formattedLexeme)

}

Logger.log(row)

outputFile.write(row.data(using: .utf8)!)

}

}

let tableFooter = "+-----------------------+--------+-------+\n"

outputFile.write(tableFooter.data(using: .utf8)!)

Logger.log(tableFooter)

}

print("Таблицы токенов записаны в \(outputFileNameTable)")

outputFile.closeFile()

} else {

print("Ошибка: нельзя открыть файл для вывода")

}

}

} else {

print("Ошибка: нельзя открыть файл для вывода")

}

}

main()