Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное   
учреждение высшего образования

Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

**Отчет по лабораторной работе № 2**

**«Среда исполнения программ на языке Pascal--»**

**Выполнили**:

студенты группы 3823Б1ФИ1

Акимов И.А., Гусев Д.А.

**Проверил**:

доцент кафедры ВВиСП, к.т.н.,

Сысоев А.В.

Нижний Новгород

2025

Содержание

[Введение 3](#_Toc167581733)

[1. Постановка задачи 5](#_Toc167581734)

[2. Руководство пользователя 6](#_Toc167581735)

[3.1. Описание структуры программы 7](#_Toc167581736)

[3.2. Описание алгоритмов 8](#_Toc167581737)

[4. Результаты экспериментов 11](#_Toc167581738)

[Заключение 13](#_Toc167581739)

[Литература 14](#_Toc167581740)

[Приложение 15](#_Toc167581741)

# Введение

**Описание предметной области**

Язык Pascal-- является упрощённой версией классического языка Pascal, ориентированной на образовательные цели и исследование основных принципов компиляции и интерпретации. В рамках разработки программирования и компиляторных технологий Pascal-- используется для демонстрации следующих ключевых моментов: построения лексического анализатора (лексера), синтаксического анализатора (парсера) методом рекурсивного спуска, представления программы в виде абстрактного синтаксического дерева (AST), преобразования выражений в постфиксную форму (алгоритм сортировочной станции Дейкстры) и интерпретации полученного дерева разбора.

Интегрированная среда разработки (IDE) для Pascal-- объединяет все перечисленные компоненты в единую систему, позволяя пользователю последовательно:

* вводить и редактировать код (консольный интерфейс);
* выполнять лексический анализ исходного текста, разбивая его на токены;
* строить синтаксическое дерево с помощью парсера, реализованного по грамматике Pascal--;
* представлять выражения в виде обратной польской записи и вычислять их значение (реализован калькулятор на основе постфиксной формы);
* хранить символы (переменные, константы) в таблице (реализована хеш-таблица);
* выполнять программу, обходя AST и обрабатывая операторы присваивания, условного ветвления, циклов, ввода/вывода.

Такая среда служит универсальной платформой для изучения основ компиляторостроения, разработки систем программирования и отладки алгоритмов обработки языков высокого уровня. В рамках лабораторной работы №2 создаётся консольная IDE, реализующая перечисленные компоненты на языке C++. Пользователь может загружать текст программы, проверять синтаксис, запускать интерпретацию и наблюдать состояние переменных через команду ***vars***.

**Актуальность задачи**

Современное образование в области программной инженерии и компиляторных технологий требует наличия наглядных, доступных инструментов для изучения этапов трансляции и исполнения программ. Специально упрощённые языки типа Pascal-- позволяют сосредоточиться на фундаментальных алгоритмах и структурах данных — лексическом анализе, построении AST, алгоритме сортировочной станции, управлении таблицей символов и интерпретации без избыточной сложности полнофункциональных языков.

Реализация интегрированной среды разработки для Pascal-- особенно актуальна по следующим причинам:

* Образовательная ценность. Наличие рабочего прототипа IDE помогает студентам и начинающим инженерам визуализировать процесс трансляции, отладки и исполнения кода на каждом этапе (лексический анализ, парсинг, генерация постфиксной формы, интерпретация). Это усиливает понимание теоретических основ компиляторостроения и разработки языков программирования.
* Отладка и тестирование. Консольный интерфейс с возможностью интерактивного редактирования и запуска ускоряет выявление ошибок, позволяет наглядно проверять работу модулей (лексера, парсера, интерпретатора) и добавлять модульные тесты (Google Test) для автоматической валидации корректности работы ключевых компонентов.
* Практическое применение. Изучение Pascal-- и разработка соответствующей IDE создают фундамент для переноса полученных знаний на более сложные задачи: разработка языков доменной специфики, создание компиляторов и интерпретаторов реальных языков, оптимизация механизмов хранения символов (различные структуры данных — упорядоченные таблицы, деревья поиска, хеш-таблицы).

Таким образом, разработка IDE для Pascal-- является не только учебным упражнением, но и базовым шагом к созданию полноценных средств разработки, способствует углублённому пониманию внутренних механизмов трансляции и исполнения программ, а также формирует навыки объёмного проектирования архитектуры системы.

# Постановка задачи

Разработать интегрированную среду исполнения (IDE) для программ на языке Pascal-- с консольным интерфейсом, которая должна обеспечивать следующие функции и компоненты:

1. Консольный пользовательский интерфейс
2. Лексический анализатор

* Реализовать класс *Lexer*, который получает на вход исходный текст и возвращает и последовательно список токенов (структура *Token*).
* Распознавать все ключевые слова Pascal--.
* Обрабатывать операторы и разделители.
* Распознавать числовые литералы, строковые литералы, булевы константы, идентификаторы.

1. Синтаксический анализатор

* Реализовать класс *Parser*, выполняющий нисходящий рекурсивный разбор по всей грамматике Pascal--.
* Построение абстрактного синтаксического дерева (AST) из узлов *ASTNode*, где каждый узел имеет тип *ASTNodeType* и список дочерних узлов.
* Поддерживать следующие конструкции Pascal--: программа целиком, секция констант, секция переменных, операторы, выражения.

1. Абстрактное синтаксическое дерево (AST)

* Структура *ASTNode*, где каждый узел хранит: тип узла, значение, вектор дочерних узлов.
* Построить AST таким образом, чтобы задачи семантического анализа и интерпретации выполнялись путём рекурсивного обхода дерева.

1. Постфиксный калькулятор

* Реализовать класс *PostfixCalculator*, который: метод, преобразующий узел AST в вектор лексем в постфиксной форме; метод, вычисляющий значение выражения с учётом переменных; метод, реализующий арифметические операции, сравнительные, логические, унарный минуc; обеспечить поддержку конкатенации строк и преобразования типов.

1. Интерпретатор

* Реализовать класс *Interpreter*, использующий: таблицу символов, методы обхода AST и выполнение операторов по типу узла, вычисления выражения.

1. Таблица символов

* Реализовать собственную хеш-таблицу *SymbolTable*.
* Межмодульное взаимодействие.

1. Главный модуль (файл *main.cpp*)

* Инициализация консоли (кодировки, очистки).
* Главный цикл с отображением меню и обработкой команд.

1. Модульное тестирование (Google Test)

Таким образом, **постановка задачи** состоит в поэтапной разработке всех перечисленных модулей (лексер, парсер, AST, постфиксный калькулятор, интерпретатор, таблица символов, консольный интерфейс) и обеспечении их взаимодействия в рамках консольной IDE для языка Pascal--, с обязательным покрытием ключевых функций модульными тестами и логированием.

# Руководство пользователя

Данное руководство описывает работу с консольной интегрированной средой разработки (IDE) для языка Pascal--. Пользователь взаимодействует с программой через терминал (консоль) и управляет кодом при помощи простых текстовых команд.

После запуска программы пользователь попадает в интерактивное меню, где доступны следующие команды:

* new — создать новую программу в буфере. Построчно вводите код на Pascal--. Чтобы закончить ввод, введите строку *end*.
* open <имя\_файла> — загрузка программы из файла.
* save <имя\_файла> — сохранение программы в файл.
* edit — редактирование существующего кода в буфере. Введите программу построчно (как в командe *new*). Для завершения ввода снова напишите end на отдельной строке.
* check — проверка синтаксиса кода.
* run — выполнение (интерпретация) программы.
* vars — просмотр текущих значений переменных.
* exit — выход из IDE.

1. **Руководство программиста**

## Описание структуры программы

Программа предназначена для интерпретации и выполнения кода на языке Pascal--. Она выполняет лексический анализ, синтаксический разбор и интерпретацию Pascal-программ. Программа принимает исходный код на языке Pascal--, преобразует его в абстрактное синтаксическое дерево и выполняет соответствующие операции. Все этапы обработки кода выполняются через классы и функции, обеспечивающие модульность и расширяемость кода.

Проект включает в себя следующие основные элементы:

1. Основные компоненты системы:

* Лексический анализатор (Lexer) — преобразует исходный код в последовательность токенов.
* Синтаксический анализатор (Parser) — строит абстрактное синтаксическое дерево на основе токенов.
* Постфиксный калькулятор (PostfixCalculator) — преобразует выражения в постфиксную форму и вычисляет их.
* Интерпретатор (Interpreter) — выполняет программу, обходя узлы AST.
* Таблица символов (SymbolTable) — хранит информацию о переменных и их значениях.
* Система отчетов об ошибках (ErrorReporter) — обрабатывает ошибки и предупреждения.

1. Классы для представления данных:

* Token — представляет токены исходного кода (ключевые слова, идентификаторы, литералы и т.д.).
* ASTNode — узлы абстрактного синтаксического дерева различных типов.
* Value — универсальная структура для хранения значений разных типов (Integer, Real, Boolean, String).

1. Поддержка конструкций языка Pascal--:

* Ключевые слова: program, var, const, begin, end, if, then, else, while, do, for, to, downto и др.
* Типы данных: integer, real, boolean, string.
* Операторы: присваивание (:=), арифметические (+, -, \*, /, div, mod), логические (and, or, not), сравнения (=, <>, <, >, <=, >=).
* Управляющие конструкции: if-then-else, while-do, for-to-do, for-downto-do.
* Ввод/вывод: read, readln, write, writeln.

1. Интерфейсы для обеспечения модульности:

* ICompilerComponent — базовый интерфейс для всех компонентов компилятора.
* IErrorReporter — интерфейс для системы отчетов об ошибках.
* ILexer — интерфейс для лексического анализатора.
* IParser — интерфейс для синтаксического анализатора.
* IInterpreter — интерфейс для интерпретатора.

1. Обработка ошибок и исключений:

* Проверка и сообщения о синтаксических ошибках.
* Проверка и сообщения о семантических ошибках.
* Информативные сообщения с указанием строки и столбца в исходном коде.
* Обработка ошибок времени выполнения (деление на ноль, неопределенные переменные и т.д.).

1. Тесты:

* Тесты лексического анализатора (test\_lexer.cpp):
* Проверка токенизации всех ключевых слов языка.
* Проверка распознавания идентификаторов, чисел и строк.
* Проверка распознавания операторов и разделителей.
* Тестирование обработки пробелов и табуляции.
* Проверка корректной обработки комментариев.
* Тестирование токенизации управляющих конструкций (if, while, for).
* Проверка токенизации логических операторов и булевых констант.
* Тестирование обработки вложенных комментариев.
* Проверка распознавания вещественных чисел.
* Тесты синтаксического анализатора (test\_parser.cpp):
* Проверка построения AST для различных конструкций языка.
* Тестирование обработки объявлений переменных и констант.
* Проверка парсинга программы, операторов и выражений.
* Тестирование обработки ошибок синтаксиса.
* Тесты интерпретатора (test\_interpreter.cpp):
* Проверка выполнения пустой программы.
* Тестирование операций присваивания и получения значений переменных.
* Проверка ветвления if-else.
* Тестирование циклов while и for (с to и downto).
* Проверка вычисления выражений разных типов.
* Тестирование операций с разными типами данных (целые, вещественные, строки).
* Проверка арифметических и логических операций.
* Тестирование унарных операций (отрицание, логическое not).
* Проверка операторов ввода/вывода (read, write, readln, writeln).
* Интеграционное тестирование выполнения простых программ.
* Тесты постфиксного калькулятора (test\_postfix.cpp):
* Проверка преобразования выражений в постфиксную форму.
* Тестирование вычисления постфиксных выражений.
* Проверка приоритета операций.
* Тестирование обработки скобок и вложенных выражений.
* Тесты таблицы символов (test\_symbol\_table.cpp):
* Проверка добавления и получения символов.
* Тестирование областей видимости.
* Проверка объявлений и переопределений переменных.
* Тестирование работы с константами.
* Обработка ошибок и исключений:
* Проверка и сообщения о синтаксических ошибках.
* Проверка и сообщения о семантических ошибках.
* Информативные сообщения с указанием строки и столбца в исходном коде.
* Обработка ошибок времени выполнения (деление на ноль, неопределенные переменные и т.д.).

Таким образом, проект представляет собой полноценный интерпретатор языка Pascal--, включающий все необходимые компоненты для анализа и выполнения программ. Особенно стоит отметить обширную тестовую базу, покрывающую все аспекты работы интерпретатора, что обеспечивает высокую надежность и корректность его работы.

## Описание алгоритмов

Программа состоит из реализации классов для создания интерпретатора языка Pascal--, включая лексический анализатор, синтаксический анализатор, интерпретатор и вспомогательные компоненты. Каждый из этих классов использует специфические алгоритмы для обработки и выполнения кода. Рассмотрим основные алгоритмы по каждому из ключевых компонентов системы.

1. Алгоритмы лексического анализатора (класс Lexer)

Алгоритм токенизации исходного кода (tokenize):

* Последовательно обрабатывается каждый символ исходного текста программы.
* В зависимости от текущего символа вызывается соответствующий метод для обработки специфического типа токена.
* Пробелы, табуляции и переносы строк пропускаются.
* Комментарии в фигурных скобках обрабатываются как специальный случай и игнорируются.
* Для каждого распознанного токена запоминается его тип, значение и позиция в тексте (строка и столбец).

Алгоритм распознавания идентификаторов и ключевых слов:

* Считывание последовательности символов, начинающейся с буквы и продолжающейся буквами, цифрами или знаком подчеркивания.
* Проверка полученной строки на соответствие ключевым словам языка.
* Если строка является ключевым словом, создаётся токен соответствующего типа.
* В противном случае создаётся токен типа "идентификатор".

2. Алгоритмы синтаксического анализатора (класс Parser)

Алгоритм построения абстрактного синтаксического дерева (parse):

* Использование метода рекурсивного спуска для обработки токенов согласно грамматике языка.
* Для каждой языковой конструкции (выражения, операторы и т.д.) вызывается специальный метод.
* Проверка правильности синтаксиса и формирование соответствующих узлов AST.
* Обработка ошибок синтаксиса с выводом информативных сообщений.

Алгоритм разбора выражений:

* Построение выражений с учетом приоритета операторов.
* Сначала обрабатываются первичные выражения (литералы, идентификаторы, скобочные выражения).
* Затем обрабатываются выражения более высокого уровня с учетом ассоциативности операторов.
* Для каждого оператора создаётся соответствующий узел AST с левым и правым операндами.

3. Алгоритмы интерпретатора (класс Interpreter)

Алгоритм выполнения программы (run):

* Рекурсивный обход AST, начиная с корневого узла.
* В зависимости от типа узла вызывается соответствующий метод для его обработки.
* Результаты выполнения узлов сохраняются в таблице символов или используются для дальнейших вычислений.

Алгоритм выполнения условного оператора (executeIf):

* Вычисление значения условия.
* Если условие истинно, выполняется ветка then.
* Если условие ложно и есть ветка else, выполняется она.

Алгоритм выполнения цикла while (executeWhile):

* Вычисление условия цикла.
* Если условие истинно, выполняется тело цикла, затем условие проверяется снова.
* Цикл продолжается до тех пор, пока условие остается истинным.

Алгоритм выполнения цикла for (executeFor):

* Инициализация переменной цикла начальным значением.
* Проверка, является ли цикл возрастающим (to) или убывающим (downto).
* Цикл выполняется, пока переменная не достигнет конечного значения.
* После каждой итерации переменная цикла увеличивается или уменьшается на 1.

4. Алгоритмы постфиксного калькулятора (класс PostfixCalculator)

Алгоритм преобразования инфиксного выражения в постфиксное:

* Использование алгоритма сортировочной станции (Shunting-yard algorithm).
* Обработка операндов, операторов и скобок с учетом их приоритета.
* Операнды сразу добавляются в выходную очередь.
* Операторы добавляются в стек, с учетом их приоритета.
* Когда встречается оператор с более высоким приоритетом, операторы из стека переносятся в выходную очередь.

Алгоритм вычисления постфиксного выражения:

* Использование стека для хранения промежуточных результатов.
* Последовательное чтение элементов выражения.
* Если элемент — операнд, он помещается в стек.
* Если элемент — оператор, извлекаются нужные операнды из стека, выполняется операция, и результат помещается обратно в стек.

5. Алгоритмы работы с таблицей символов (класс SymbolTable)

Алгоритм добавления и получения переменных:

* Использование хеш-таблицы (словаря) для быстрого доступа к переменным по имени.
* При объявлении переменной проверяется, не была ли она уже объявлена ранее.
* При доступе к переменной проверяется её наличие в таблице символов.

Эти алгоритмы обеспечивают эффективную и корректную работу интерпретатора Pascal--, позволяя ему выполнять полный цикл обработки программного кода от исходного текста до получения результатов выполнения. Модульная структура системы обеспечивает чёткое разделение ответственности между компонентами, что упрощает разработку, тестирование и дальнейшее расширение функциональности.

# Результаты экспериментов

Для оценки работы интерпретатора Pascal-- были проведены эксперименты с различными компонентами системы: лексическим анализатором, синтаксическим анализатором, интерпретатором и постфиксным калькулятором. Оценивалась корректность выполнения всех этапов обработки кода, распознавание языковых конструкций и выполнение программ.

1. Токенизация исходного кода (Lexer)

В ходе эксперимента были проанализированы различные фрагменты кода Pascal--, содержащие ключевые слова, операторы, идентификаторы и литералы.

Результат: Лексический анализатор успешно распознавал все ключевые слова языка (program, var, begin, end и др.), корректно идентифицировал операторы и разделители, а также правильно обрабатывал идентификаторы, числовые и строковые литералы. Комментарии в фигурных скобках корректно игнорировались, в том числе вложенные. Токенизация происходила с правильным сохранением информации о позиции (строке и столбце) каждого токена в исходном тексте.

2. Построение абстрактного синтаксического дерева (Parser)

Эксперименты включали анализ программ с различными конструкциями языка: объявлениями переменных, операторами присваивания, условными операторами, циклами и выражениями разной сложности.

Результат: Синтаксический анализатор корректно строил AST для всех поддерживаемых конструкций языка. Выражения правильно обрабатывались с учетом приоритета операторов и скобок. При обнаружении синтаксических ошибок формировались информативные сообщения с указанием места и характера ошибки. Структура построенного AST полностью соответствовала грамматике языка Pascal--.

3. Интерпретация программ (Interpreter)

Для тестирования интерпретатора использовались программы с различными операторами и конструкциями: присваиванием, ветвлением, циклами, вводом/выводом и выражениями разных типов.

Результат: Интерпретатор успешно выполнял все типы операторов, корректно обрабатывал выражения и управляющие конструкции. Переменные правильно сохранялись в таблице символов, и их значения корректно обновлялись при выполнении операций присваивания. Операторы ввода/вывода (read, write) работали должным образом, позволяя взаимодействовать с пользователем. Циклы for и while выполнялись ожидаемое количество раз с правильным обновлением переменных цикла.

4. Вычисление выражений (PostfixCalculator)

Эксперименты включали преобразование и вычисление арифметических и логических выражений разной сложности, включая выражения со скобками и операторами разного приоритета.

Результат: Постфиксный калькулятор корректно преобразовывал инфиксные выражения в постфиксную форму с учетом приоритета операторов. Вычисление выражений производилось без ошибок для всех поддерживаемых типов данных: целых, вещественных, логических и строковых. Результаты вычислений совпадали с ожидаемыми математическими результатами.

5. Работа с переменными и таблицей символов (SymbolTable)

Тестирование включало объявление, инициализацию и обновление переменных различных типов, а также проверку на повторное объявление и доступ к несуществующим переменным.

Результат: Система корректно добавляла переменные в таблицу символов при их объявлении, сохраняя информацию о типе. Значения переменных успешно обновлялись при выполнении операторов присваивания. При попытке доступа к неопределенной переменной генерировалось соответствующее сообщение об ошибке. Система типов правильно обрабатывала преобразования между различными типами данных, когда это было возможно.

6. Выполнение условных операторов (Interpreter::executeIf)

Проведены эксперименты с условными операторами различной сложности, включая вложенные условия и различные комбинации условий.

Результат: Интерпретатор правильно вычислял условия и выполнял соответствующие ветки кода (then или else). Конструкции с вложенными условными операторами обрабатывались корректно, с соблюдением правильной последовательности выполнения. Логические операции (and, or, not) в условиях вычислялись правильно.

7. Выполнение циклов (Interpreter::executeWhile, executeFor)

Были протестированы циклы while и for с различными условиями и телами, включая вложенные циклы и циклы с условиями выхода внутри тела.

Результат: Все типы циклов выполнялись корректно. Цикл while правильно продолжал выполнение, пока условие оставалось истинным. Циклы for в обоих вариантах (to и downto) корректно инициализировали переменную цикла и выполняли нужное количество итераций. Вложенные циклы работали без ошибок, с правильным обновлением переменных на каждом уровне вложенности.

Эти эксперименты подтвердили, что среда выполнения программ Pascal-- корректно выполняет все этапы обработки кода и обеспечивает правильное выполнение программ на этом языке. Система обладает надежностью в обработке различных конструкций языка и обеспечивает достаточную информативность при возникновении ошибок.

# Заключение

В данной программе был реализована полноценная среда исполнения программ языка Pascal--, обеспечивающая весь цикл обработки программного кода: от лексического анализа до выполнения инструкций. Интерпретатор поддерживает основные типы данных (integer, real, boolean, string) и широкий спектр языковых конструкций, включая условные операторы, циклы и операции ввода-вывода.

Программа предоставляет пользователю возможность писать, отлаживать и выполнять код на языке Pascal--, получая детальные сообщения об ошибках с указанием их местоположения в исходном коде. Модульная архитектура на основе интерфейсов обеспечивает гибкость системы и позволяет легко модифицировать или заменять отдельные компоненты без изменения остальной части кода.

Основные этапы работы программы включают:

1. Лексический анализ исходного кода и разбиение его на токены.
2. Синтаксический анализ и построение абстрактного синтаксического дерева (AST).
3. Интерпретация AST с выполнением соответствующих операций и управляющих конструкций.
4. Обработка операций ввода-вывода для взаимодействия с пользователем.

Предложенная программа демонстрирует хорошие возможности для расширения и модификации. Например, можно добавить поддержку массивов, процедур и функций, модулей или реализовать дополнительные оптимизации при выполнении кода. В перспективе возможно расширение функциональности до полноценной интегрированной среды разработки (IDE) с подсветкой синтаксиса, автодополнением кода и более продвинутыми средствами отладки.

Разработанная среда исполнения программ Pascal-- является ценным образовательным инструментом, который может использоваться для обучения основам программирования и принципам работы компиляторов и интерпретаторов. Обширная тестовая база обеспечивает надежность и корректность работы всех компонентов системы, что делает программу подходящей как для учебных целей, так и для решения практических задач программирования.

# Литература

1. Документация GoogleTest – <https://google.github.io/googletest/>.
2. Альфред Ахо, Моника Лам, Рави Сети, Джеффри Ульман. "Компиляторы: принципы, технологии и инструментарий" (2-е издание) – фундаментальная работа по теории и практике построения компиляторов и интерпретаторов.
3. Никлаус Вирт. "Построение компиляторов" – классическая работа создателя языка Pascal по разработке компиляторов.
4. Теренс Парр. "The Definitive ANTLR 4 Reference" – современное руководство по созданию парсеров и обработке языков.
5. Рональд Мак. "Написание компиляторов и интерпретаторов: практический подход" – практическое руководство с примерами реализации интерпретаторов.
6. Торбьёрн Гранлунд и др. "GNU Coding Standards" – <https://www.gnu.org/prep/standards/> – стандарты кодирования, полезные для организации крупных проектов.
7. Эрих Гамма, Ричард Хелм, Ральф Джонсон, Джон Влиссидес. "Приемы объектно-ориентированного проектирования. Паттерны проектирования" – важная книга для понимания архитектуры сложных программных систем.
8. Стивен Прата. "Язык программирования C++. Лекции и упражнения" – подробное руководство по C++, языку реализации интерпретатора.
9. Документация по CMake – <https://cmake.org/documentation/> – официальная документация по системе сборки, используемой в проекте.
10. Andrew W. Appel. "Modern Compiler Implementation in C++" – современный подход к реализации компиляторов на C++.
11. Бьёрн Страуструп. "Язык программирования C++. Специальное издание" – классический труд создателя C++ с подробным описанием возможностей языка.
12. ISO/IEC 10206:1990 "Information technology – Programming languages – Extended Pascal" – официальный стандарт языка Pascal, полезен для понимания синтаксиса и семантики языка.

# Приложение

Код: <https://github.com/Akimov52/lab2-pascal_execution_env>